Universität Bielefeld
Fakultät für Linguistik und Literaturwissenschaft
Fachbereich Computerlinguistik
Gutachter: Andreas Witt, Dr. Hans-Jürgen Eikmeyer

XML zur Strukturierung von Prosatexten

Eine XML-basierte Editierumgebung als Voraussetzung für das Single-Source-Publishing

- Magisterarbeit -

vorgelegt von:
Maik Stührenberg

Maik Stührenberg
Ernst-Rein-Str. 26
33613 Bielefeld
Matrikel-Nr.: 1214598

Inhaltsverzeichnis

1	**Einleitung**	**1**
2	**Warum XML, warum nicht Microsoft Word? - Die Vorteile von Markup-Systemen**	**4**
2.1	Word .doc - ein proprietäres Format	5
2.2	LaTeX - ein Satzsystem mit elektronisch-spezifischen Markup	7
2.2.1	LaTeX Markup	8
2.2.2	LaTeX als Ausgangspunkt für Single-Source-Publishing	9
3	**SGML und XML - generisches Markup**	**11**
3.1	SGML	12
3.2	Die eXtensible Markup Language	12
3.2.1	Das Prinzip hinter SGML und XML	14
3.2.1.1	Deskriptives Markup	14
3.2.1.2	Dokumenttypen	15
3.2.1.2.1	Elemente	15
3.2.1.2.2	Attribute	17
3.2.1.2.3	Entitäten	18
3.2.1.3	Datenunabhängigkeit	19
3.2.1.4	Wohlgeformtheit und Gültigkeit	19
4	**XML in der Anwendung: die Magisterarbeit**	**20**
4.1	Die Document Type Definition	20
4.1.1	meta	21
4.1.2	body	24
4.1.3	appendix	29
4.1.4	bibliography	31
4.1.5	Modularität	32
5	**XML-Werkzeuge**	**34**
5.1	XMetaL	34
5.2	Emacs	36
6	**Transformation von SGML und XML**	**38**
6.1	DSSSL	38
6.2	XSL - die eXtensible Style Sheet Language	39

6.2.1	XSLT(ransform)	40
6.2.1.1	Verwendung von XSLT Style Sheets	41
6.2.1.2	Aufbau eines XSLT Style Sheets	42
6.2.1.2.1	XSL Templates	43
6.2.1.2.2	Das `xsl:apply-templates` Element	43
6.2.1.2.3	Das Attribut `select`	44
6.2.1.2.4	Verarbeitung mit Hilfe von `xsl:value-of`	44
6.2.1.2.5	Verarbeitung mehrerer Elemente mit `xsl:for-each`	45
6.2.1.2.6	Vergleichsmuster zur Elementauswahl	45
6.2.1.2.7	Steuerung der Ausgabe	45
6.2.1.2.8	Zählen von Knoten mit `xsl:number`	46
6.2.1.2.9	Sortieren der Ausgabe	47
6.2.1.2.10	Treffen einer Auswahl mit `xsl:if` und `xsl:choose`	47
6.2.2	XSL-F(ormatting) O(bjects)	48
6.2.2.1	Seitenlayout	49
6.2.2.1.1	Definition von Seiten mit `fo:layout-master-set`	50
6.2.2.2	Block Elemente	51
6.2.2.3	Inline Elemente	53
6.2.2.4	Out-of-line Elemente	55
6.2.2.5	Attribute (properties)	56
7	**XSL in der Anwendung**	**58**
7.1	Überführung nach HTML	58
7.1.1	Vorüberlegungen	58
7.1.2	Das XSLT Style Sheet für die Transformation nach HTML	59
7.2	Überführung nach XSL-FO	72
7.2.1	Vorüberlegungen	73
7.2.2	Das XSLT Style Sheet für die Überführung nach XSL-FO	73
8	**Printausgabe mit FOP und Acrobat Reader**	**89**
9	**Theorie und Praxis - die Probleme bei der Benutzung neuer Technologien**	**91**
10	**Schluss**	**93**
A	**Anhang**	**95**
A.1	Die DTD	95

A.1.1	magisterarbeit.dtd	95
A.1.2	glossary.ent	97
A.1.3	abbrevations.ent	97
A.1.4	bibliography.ent	98
A.2	Die XSLT Style Sheets	99
A.2.1	Überführung nach HTML	99
A.2.2	Überführung nach XSL-FO	113
	Abkürzungsverzeichnis	**138**
	Abbildungsverzeichnis	**140**
	Literaturverzeichnis	**141**

1 Einleitung

Single-Source-Publishing ist ein Ausdruck, hinter dem ein Verfahren und eine Technologie stehen, die in Zeiten von Internet, Palmsize Computern und WAP-Handys - kurz: einer Vielzahl an möglichen Ausgabeformaten für Informationen aller Art - immer mehr an Bedeutung gewinnen. Die Idee hinter Single-Source-Publishing ist, dass man ein Ausgangsdokument hat, aus dem mehr oder weniger automatisch (bzw. mittels einzelner Umformungsschritte) mehrere Zieldokumente entstehen können, die alle bestmöglich an das Produkt angepasst sind, mit dem sie betrachtet werden. Nehmen wir nur die oben genannten: ein Text, der im Internet veröffentlicht werden soll, ist in einer Auszeichnungssprache mit Namen HTML, HyperText Markup Language, verfasst. HTML wurde in den 90er Jahren des 20. Jahrhunderts konzipiert, um Texte über das weltweit umspannende Internet zu publizieren. Ein Text auf einer Seite im Internet sollte idealerweise folgende Voraussetzungen erfüllen:

- Die Schriftgröße und Schriftart müssen den Anforderungen zum Lesen auf dem Bildschirm genügen, da das Lesen vom Monitor die Augen sehr viel mehr anstrengt, als es das Lesen vom Papier tut. Standardmäßig wird für Bildschirmtexte eine serifenlose Schrift, wie zum Beispiel Verdana oder Arial verwendet. Die Schrift selbst sollte groß genug sein, damit aus einer typischen Entfernung zum Monitor noch gut gelesen werden kann.
- Aus den gleichen Gründen sollte der Kontrast zwischen Schriftfarbe und Hintergrund nicht zu stark sein.
- Die Informationsmenge, die gleichzeitig am Bildschirm präsentiert wird, sollte möglichst gering gehalten werden, oder zumindest sinnvoll portioniert.

Informationen, die auf Geräten mit kleinem und oft auch monochromen Bildschirmen, wie sie bei Handys und PDAs der Fall sind, konsumiert werden, stellen dagegen andere Anforderungen:

- Die Informationsmenge muss noch stärker komprimiert werden, da auf dem Display nur sehr wenig Textzeilen auf einmal Platz haben.
- Auf Layout-orientierte Gestaltung sollte tunlichst verzichtet werden, da dadurch nicht nur wichtiger Platz, sondern auch Bandbreite und somit unnötige Kosten entstehen.
- Aus dem gleichen Grund sollten auch Grafiken und andere Multimedia-Objekte vermieden werden.

Printtexte stellen weit weniger Anforderungen an den Produzenten: Natürlich ist auch hier ein wohl gewählter Kontrast sinnvoll, und im Gegensatz zu Bildschirmtexten sollten Serifenschriften gewählt werden, ansonsten ist aber (fast) alles erlaubt, vieles aber nicht möglich, was interaktive Medien ermöglichen.

Für alle diese Ausgabemedien war es bisher unvermeidlich jeweils verschiedene, auf den speziellen Zweck zugeschnittene Produkte zu produzieren, zumindest, wenn man den Kunden

- und als solcher sollte ein Leser eigentlich immer gelten - ein optimales Ergebnis präsentieren wollte. Jeder, der schon einmal eine Internetseite ausgedruckt hat, hat sich darüber geärgert, dass Links auf dem Ausdruck ihre Funktion verlieren und sich das auf dem Computer strukturierte Dokument in einen langen Fließtext verwandelt hat - kurz gesagt: alles, was den Reiz eines Hypertextes ausmacht, geht verloren. Nebenbei sehen solche ausgedrucken HTML-Seiten alles andere als besonders gut aus - zumindest, wenn man ein normales Printprodukt dagegen hält. Das liegt daran, dass HTML dafür entwickelt wurde, Information in Hypertexten, also miteinander verlinkten, referenzierbaren Textteilen, zu präsentieren. Design und auch Multimedia-Elemente gehörten zu Beginn nicht oder nur zu sehr geringen Teilen zum Standard. Erst mit den Cascading Style Sheets (CSS) ist es möglich, mit geringem Aufwand HTML-Seiten ein recht ansprechendes Ergebnis zu verschaffen. WAP dagegen - die Technologie, die zum Einsatz kommt, wenn man mit dem Handy durch das Internet surft, ist noch viel stärker auf den Inhalt fokussiert und bietet nahezu keinerlei am Design orientierte Elemente.

Seit 1986 ist es mit SGML möglich, Daten zu strukturieren und ein solcher Art annotiertes Dokument mit Hilfe von DSSSL in ein Printdokument oder auch nach HTML zu überführen. Dazu mehr in den entsprechenden Teilen dieser Arbeit.

Die Arbeit mit SGML hat einen entscheidenden Nachteil: SGML ist zu komplex und es gibt keine Software, die in der Lage ist, den vollen ISO-Standard zu unterstützen. Und obwohl SGML auf ASCII-Text aufbaut, also SGML-Dokumente prinzipiell mit jedem Texteditor zu erstellen und zu bearbeiten sind, gibt es nur wenig frei verfügbare Software, die SGML unterstützt. Kommerzielle Software ist allein schon wegen der Anschaffungskosten eher in Großverlagen zu finden.

XML dagegen ist ein Versuch des W3C, die Vorzüge von SGML mit der leichteren Erlernbarkeit von HTML zu kombinieren, ein *SGML for the Web*. XML ist daher die Grundlage für diese Arbeit.

Wenn im Titel dieser Arbeit von Entwicklungsumgebung gesprochen wird, dann ist der Terminus in der Art zu verstehen, als dass in dieser Arbeit schrittweise die Mittel und Werkzeuge vorgestellt werden, mit denen es möglich ist, ein auf XML basierendes Single-Source-Publishing durchzuführen. Ziel dieser Arbeit ist also die nachvollziehbare Darstellung des Aufbaus und der Durchführung eines Single-Source-Publishing Projekts. Auch am Ende dieser Arbeit wird der Leser nicht vor einer Entwicklungsumgebung im Sinne eines Softwareprodukts aus einem Guss, einer IDE, für die Erstellung und Bearbeitung von XML-annotierten Daten stehen. Allerdings wird man nach Lektüre dieser Arbeit in der Lage sein, die hier vorgestellten Software-Produkte in sinnvoller Art und Weise zu kombinieren und für eigene Zwecke zu benutzen.

Im ersten Teil der Arbeit soll die Motivation dargelegt werden, die mich dazu veranlasst

hat, diese Magisterarbeit und die damit verbundene Entwicklungsarbeit anzugehen. Hier soll auf die generellen Unterschiede zwischen klassischer Textverarbeitung und Markup-Modellen hingewiesen werden. Anhand einiger ausgewählter Markup- und Metasprachen soll das Prinzip, das dieser Arbeit zu Grunde liegt, vorgestellt werden. An diesen Überblick schließt sich eine Darstellung des technischen Hintergrunds dieser Arbeit - XML und die flankierenden Standards - im Hauptteil der Arbeit an. Neben diesen theoretischen Grundlagen geht es um den konkreten Fall einer Magisterarbeit. Es werden die Schritte aufgezeigt, die nötig waren, um diese Arbeit als XML-Dokument entstehen zu lassen. Ausgehend davon werden einige Werkzeuge gewürdigt, die bei der Arbeit mit XML hilfreich sind und die stellvertretend für die wachsende Zahl an XML-Software stehen sollen.

Nach der Produktion des XML-Dokuments beginnt der zweite Abschnitt der Entwicklungsarbeit: die Überführung in eine Print- und eine Onlineversion. Eine Einführung in den hierzu verwendeten Standard XSL und die praktische Demonstration bilden den Kern dieses Abschnitts.

Eine Darstellung zu den Problemen, die während dieser Arbeit auftraten, aber auch ein Ausblick auf die Zukunft, sind Inhalt des letzten Teils dieser Arbeit.

An dieser Stelle noch ein Hinweis: Englischsprachige Begriffe sind in weiten Teilen dieser Arbeit vorhanden. Das liegt daran, dass XML und die damit verbundenen Standards noch relativ jung sind. Deutsche Literatur ist dementsprechend meistens eine Übersetzung amerikanischer Werke und oftmals nicht auf dem aktuellsten Stand. Ebenso sind die W3C Standards alle in englisch verfasst, was mich dazu veranlasst hat, in der Mehrzahl der Fälle bei der originären Bezeichnung zu bleiben. Vor allem das Nachlesen in der angegebenen Print- und Online-Literatur wird dadurch erleichtert.

2 Warum XML, warum nicht Microsoft Word? - Die Vorteile von Markup-Systemen

Generell bestehen Texte aus zwei Teilen: einem grafischen Layout und einer inhaltlichen Struktur. Die grafische Oberfläche ist der logischen inhaltlichen Struktur untergeordnet.

Im klassischen Sinne sind Inhalt und Layout von einander getrennt. Der Autor schreibt den Text und ein Setzer kümmert sich um dann um Schriftgrößen, Absätze und Spaltenzahl. Durch die weite Verbreitung von Computern ist der Autor auch Setzer in einer Person geworden. Eine Lösung, die nicht nur Vorteile mit sich bringt, wie der folgende Abschnitt zeigen soll. Der weitverbreiteten Textverarbeitung Microsoft Word werden Lösungen gegenübergestellt, die mit verschiedenen Methoden der Textauszeichnung, also Markup arbeiten. Bevor auf die einzelnen Markup-Systeme eingegangen wird, zuerst eine kurze Definition des Begriffs Markup:

Geschichtlich gesehen beschreibt Markup die Auszeichnung eines Textes mit bestimmten Befehlen, die dem Setzer mitteilen, wie er den Text zu layouten hat. Seit jeher haben Autoren Notizen und Anmerkungen auf ihren Manuskripten gemacht, um die Arbeit des Setzers zu erleichtern (oder zu erschweren). Diese Notizen und Anmerkungen bezeichnet man als Markup. Eine Sammlung solcher Anmerkungen, die alle einer bestimmten Syntax und Grammatik angehören, nennt man Sprache. Als einfachste Form einer Markupsprache kann die Interpunktion in Texten gelten, die dem Leser verdeutlichen soll, wie der Text zu lesen ist.[1]

Einhergehend mit der Technisierung dieser Arbeit bezeichnet man mit Markup allgemein alle Formen von Symbolen und Ausdrücken, die die Interpretation eines Textes explizit machen. Sogar der von allen Computern unterstütze ASCII-Standard besitzt Symbole, die ursprünglich als Markup gedacht waren, jedoch wenig Verbreitung gefunden haben. Die bekanntesten dieser Symbole sind die Zeichen zur Anzeige eines Zeilenendes. Die Betriebssysteme MS-DOS und Windows haben das Zeichenpaar `CR-LF` (`carriage-return`, `line-feed`) von der Schreibmaschine übernommen, Unix arbeitet mit einem einfachen `LF`, Computer auf MacOS-Basis benutzen ein einfaches `CR`. Die Folge kennt jeder, der mit jemand anderem Texte austauscht, die auf einem anderen Computersystem erstellt wurden: ein einfacher Texteditor kann die Erzeugnisse der Gegenseite nicht korrekt darstellen - obwohl beide auf dem selben Standard ASCII beruhen.

Man unterscheidet zwischen mehreren Typen von Markup: Hardcopy-Markup, elektronisch-spezifisches Markup und generisches Markup.[2] Zwei dieser Markup-Typen soll in dieser Arbeit Platz eingeräumt werden: dem elektronisch-spezifischen Markup, das in Form des Satzsystems LaTeX vorgestellt wird, und dem generischen Markup, auf dem SGML und

[1] XMLPRO S. 9
[2] BOCK Abschnitt "Markup"

XML beruhen. Vorher sollen allerdings die Nachteile einer klassischen Textverarbeitung für eine Single-Source-Publishing Anwendung verdeutlicht werden.

2.1 Word .doc - ein proprietäres Format

Geht es um das Schreiben von Texten, ist meistens eine Textverarbeitung die erste Wahl. Aufgrund der Quasi-Monopolstellung des amerikanischen Software-Giganten Microsoft ist ein solches Programm üblicherweise Microsoft Word in eines seiner zahlreichen Versionen. Beim Kauf eines PCs kommt man gar nicht darum herum, eine Version dieses Produkts zu erwerben, Nun ist dieses Programm nicht zu Unrecht umstritten, abgesehen von Abstürzen, Problemen bei der Fußnotenverwaltung und eingebundenen Grafiken, sprechen aus computerlinguistischer Sicht weitere Punkte gegen den Einsatz einer solchen Textverarbeitung. Auch wenn das Format, in dem Word seine Texte speichert, das doc-Format, durch die weite Verbreitung recht populär und ein Quasi-Standard geworden ist - es bleibt proprietär. Das heißt, nur Personen, die über die gleiche Software verfügen, können den Text so lesen, wie es der Autor gewollt hat. Versucht man dagegen, einen Text im doc-Format mit einem normalen Texteditor zu öffnen, sieht das Ergebnis aus wie in der Abbildung.

Abbildung 1: *Word-Dokument im Texteditor Emacs*

Der eigentliche Text ist zwar sichtbar, aber umgeben von vielen Zeichen, mit denen ein menschlicher Leser nichts anfangen kann. Selbst wenn sich ein Leser die Mühe machen

wollte, den eigentlichen Text zu entziffern, so wäre es zum einen sehr mühsam und zum anderen würde er nicht wissen, welcher Teil des Textes eine Überschrift oder ein anderes logisches Textelement darstellt. Es fehlen schlichtweg sämtliche strukturellen Informationen, auf die ein menschlicher Leser Wert legt.

Das ist aber nicht der einzige Grund, warum normale Textverarbeitungen für Single-Source-Publishing unbrauchbar sind. Wer schon einmal versucht hat, Texte aus Word in ein anderes Format, wie zum Beispiel HTML, zu konvertieren, wird nur anfangs angenehm überrascht sein. Zwar bieten fast alle gängigen Textverarbeitungen die Überführung ihrer Texte in HTML an, sieht man sich aber das Ergebnis genauer an, erkennt man, dass die Ausgabe einiges ist - nur kein Standard konformes HTML. Aus einer einfachen Seite, die nichts anderes enthält, als den Satz "Dies ist eine Seite", erzeugt Word das HTML-Dokument, das in der Abbildung zu sehen ist.

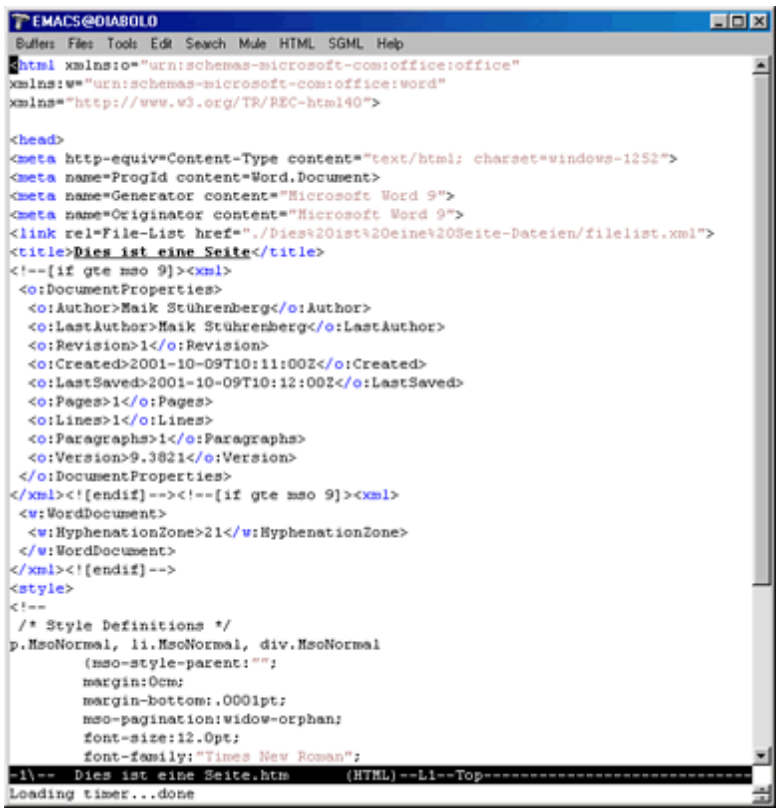

Abbildung 2: *eine Word HTML-Datei*

Das Dokument ist mit Informationen angereichert, die nur den Zweck haben, das Dokument wieder in das doc-Format zurück überführen zu können. Es finden sich die für Word typischen Information wie Autor, letzter Autor, Revision, Erstellungsdatum und vieles mehr. Daran schließen sich Microsoft-eigene Style Sheet Angaben an, die nicht zum HTML-Standard gehören. Die Ausgabedatei wird dadurch um ein Vielfaches größer als normale HTML-Dokumente. Für das Publizieren im Internet sind solche Seiten nicht zu gebrauchen. Auch in

Zeiten von Flatrates ist ein solches Dokument für Internetnutzer auf Grund der damit verbundenen langen Ladezeiten inakzeptabel.

Für Single-Source-Publishing ist eine klassische Textverarbeitung daher alles andere als ein sinnvolles Werkzeug zur Informationsspeicherung. Dazu werden andere Werkzeuge benötigt: Markup-Systeme.

2.2 LaTeX - ein Satzsystem mit elektronisch-spezifischen Markup

LaTeX (das "X" wird wie ein "ch" ausgesprochen) ist ein Textsatzsystem, das auf TeX aufbaut. TeX ist keine WYSIWYG-Textverarbeitung wie Word, es besitzt noch nicht einmal eine grafische Oberfläche. Dennoch kann man mit TeX hohe typographische Ansprüche erfüllen. Unübertroffen sind TeX und LaTeX trotz ihres hohen Alters[3] vor allem im Satz von mathematischen Formeln.

Ein TeX- oder LaTeX-System ist kein einzelnes Stück Software, sondern vielmehr das Zusammenspielen von einem Editor (der üblicherweise nicht zum TeX-System dazugehört), dem TeX-Programm, das für die Formatierung sorgt, einem Programm zur Darstellung der Ausgabedatei (einem Previewer) und einem Druckertreiber zur Druckausgabe.[4] Dabei werden - ähnlich der Arbeitsweise einer Programmiersprache - aus jedem dieser einzelnen Bestandteile Ausgabedateien erzeugt, die vom folgenden Werkzeug eingelesen und weiter verarbeitet werden. Aus diesem Grund sind TeX und LaTeX nicht interaktiv wie eine klassische Textverarbeitung, die jede Formatierung sofort anzeigt. Um Änderungen sichtbar zu machen ist immer ein erneuter Aufruf des TeX-Interpreters nötig.

TeX enthält über 900 Befehle, darunter 300 primitive Basisbefehle zum Setzen von Text. Das erschwert das Erlernen und den Einsatz im Privatgebrauch. Allerdings enthält TeX auch eine Makrosprache, mit deren Hilfe Funktionssammlungen auf höheren Abstraktionsebenen geschaffen werden können. Eine solche Sammlung an Funktionen und Formatvorlagen ist LaTeX, 1984 von Leslie Lamport entwickelt. Seit einiger Zeit arbeitet ein Team von unabhängigen Entwicklern an der Verbesserung und Erweiterung des Pakets. LaTeX hat es in kurzer Zeit geschafft, zur weltweit häufigsten Benutzerschnittstelle zur TeX-Welt zu werden. Umsetzungen von LaTeX gibt es für alle gängigen und darüber hinaus noch für viele exotische Betriebssysteme. Unter Windows ist MikTeX *(http://www.miktex.org)* eine weit verbreitete Distribution.

Was LaTeX im Zusammenhang mit dieser Arbeit interessant macht, sind zweierlei Aspekte:

[3] Eine erste funktionierende Version von TeX wurde von Donald E. Knuth schon 1978 herausgegeben. Seit 1982, in der Computerhistorie wahrlich ein biblisches Alter, gilt TeX als stabil.
[4] LATEX S. 5 ff.

- LaTeX ist eine Markupsprache. Einige der Konzepte, die LaTeX benutzt, um Texte zu strukturieren, wurden auch bei der Erstellung dieser Arbeit berücksichtigt.
- LaTeX ist um neue Dokumentklassen erweiterbar. Damit ähnelt das Konzept dem der Metasprachen SGML und XML.

2.2.1 LaTeX Markup

Das mächtige strukturelle Markup von LaTeX ist seine Stärke. Der Benutzer wählt aus einer der bereitgestellten Dokumentklassen eine passende aus und wird vom Layout vollkommen freigestellt. Auf Wunsch können allerdings explizite optische Markup-Informationen hinzugefügt werden. Zur Auswahl stehen verschiedene Layoutklassen, je nach Umfang und Verwendungszweck. Vom `article` über `report` bis hin zu `book`. Daneben gibt es noch Standardklassen für Briefe, Folien oder auch Poster. Da LaTeX vor allem im wissenschaftlichen Bereich eingesetzt wird, sind die Klassen vor allem für Dokumente dieser Zielgruppe optimiert.

Die LaTeX Tags unterscheiden sich von denen in HTML oder SGML/XML. In den zuletzt genannten Markupsprachen besteht ein Element aus einem öffnenden und schließenden Tag, jeweils durch "<" und ">" umschlossen. LaTeX-Tags werden dagegen üblicherweise durch den Backslash ("\") eingeleitet (von eingen wenigen Ausnahmen abgesehen). Die Bezeichnung der Tags ist in der Regel englisch und deskriptiv. Parameter, die den Attributen in anderen Markup- und Metasprachen vergleichbar sind, werden in geschweiften Klammern "`{`" bzw. "`}`" eingeschlossen, optionale Argumente in eckigen Klammern "`[`" bzw. "`]`".[5] Ein Beispiel-Dokument ist in der Abbildung zu sehen. LaTeX unterteilt den Text in Abschnitte verschiedener Hierarchieordnungen (`section`, `subsection`, `subsubsection`). Der Titel und der Autor eines Dokuments werden entsprechend ausgezeichnet.

[5] LATEX S. 11 ff.

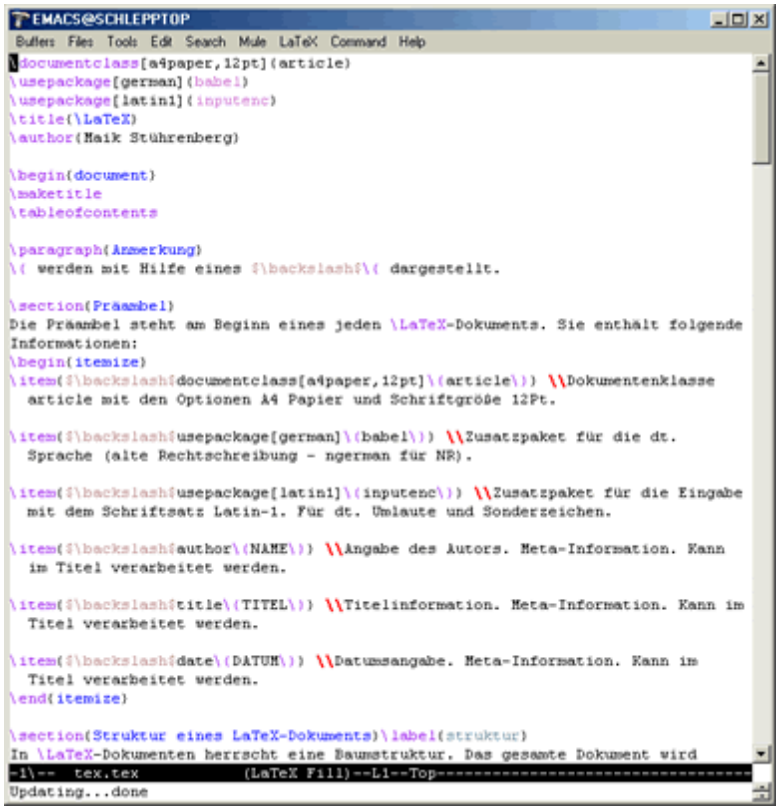

Abbildung 3: *ein LaTeX-Dokument*

Einige der hier angesprochenen Konstrukte werden in späteren Abschnitten dieser Arbeit wieder auftauchen.

2.2.2 LaTeX als Ausgangspunkt für Single-Source-Publishing

Da LaTeX ein sehr viel offeneres System ist als eine konventionelleTextverarbeitung, bietet es sich als Ausgangsbasis für Single-Source-Publishing an. Die Printausgabe von LaTeX ist über jeden Zweifel erhaben, was auf den ersten Blick fehlt, ist die Möglichkeit, Online-Versionen von LaTeX-Dokumenten zu erstellen. Das ist allerdings nicht korrekt: es gibt mehrere Möglichkeiten, LaTeX ins Web zu bringen. TeX4ht *(http://www.cis.ohio-state.edu/~gurari/TeX4ht/mn.html)* oder LaTeX2HTML *(http://www.tug.org/latex2html)* sind zwei Tools, die eine Konvertierung von LaTeX nach HTML ermöglichen. Dabei werden Inhaltsverzeichnisse und bibliographische Einträge in HTML-Links umgewandelt, die Funktionalität bleibt also erhalten und wird sogar erweitert.[6]

Eine solche Transformation hat allerdings auch Nachteile. Da LaTeX vor allem auf Grund seiner Fähigkeiten im mathematischen Formelsatz eingesetzt wird, müssen solche Formelausdrücke zu Grafiken umformatiert werden, da HTML nur wenige der verwendeten Sonderzeichen unterstützt. Das wiederum verlängert bei einem umfangreichen Dokument die Ladezeiten erheblich. Eine zufriedenstellende Alternative wird erst mit der breiten Unterstützung von

[6] TEXWEB S. 16 ff.

Unicode *(http://www.unicode.org)* oder MathML in Webbrowsern vorhanden sein. Bei Ausgabe und Konvertierung von normalen Texten erzeugen die vorhandenen Konverter allerdings schon jetzt überzeugende Ergebnisse. Dennoch ist selbst nach Aussage von Goossens und Rahtz XML die flexiblere Basis für Single-Source-Publishing.[7] Ein Grund sich XML und den Vorgänger SGML näher anzusehen.

[7] TEXWEB S. 23 f.

3 SGML und XML - generisches Markup

Mit der Verabschiedung SGMLs als ISO-Standard 8879:1986 wurde eine neue Technologie ins Leben gerufen, die den Austausch von Informationen revolutioniert hat. SGML bezeichnet ein standardisiertes Verfahren zur inhaltlichen Markierung elektronisch erstellter Texte und die Speicherung und den Austausch von Daten. Dazu wird in den Dokumenten die logische Struktur der Texte markiert, man spricht vom generischen Markup.

Die Arbeiten zu generischem Markup reichen bis in die 60er Jahre des 20. Jahrhunderts zurück. 1967 veröffentlichten William Tunnicliffe von der Graphic Communications Association und Stanley Rice parallel ihre Vorstellungen, Text unabhängig von seinem Aussehen auszuzeichnen.[8]1969 betrat nicht nur der erste Mensch den Mond, sondern Ed Mosher, Ray Lorie und Charles F. Goldfarb erfanden auch in den IBM Forschungszentren die erste moderne Markupsprache, die Generalized Markup Language (GML). GML war eine Metasprache, um andere Markupsprachen zu kreieren. Aus ihr wurde 1986 SGML, die Standard Generalized Markup Language.[9]

Ein derart annotierter Text wird von einem Programm, einem SGML-Parser, gelesen und in einer auf das gewünschte Ausgabemedium angepassten Form wiedergegeben. Damit lassen sich aus SGML-Dokumenten Texte für die unterschiedlichsten Zwecke erstellen, ein ideales Mittel für Single-Source-Publishing. Henning Lobin spricht von einer *neuen Zeitrechnung*, in der *die Informationstechnologie aus den Fesseln der technischen Abhängigkeiten befreit* wurde und *Information ausschließlich auf der Grundlage ihrer inneren Gesetzmäßigkeiten und ihrer Funktion* modelliert und verarbeitet werden.[10] Dieser Aussage ist sicherlich zu folgen, vor allem wenn man an den Siegeszug des World Wide Web denkt, der fast ausschließlich dem Einsatz der leicht erlernbaren und schnell weit verbreiteten SGML-Anwendung HTML zu verdanken ist. Auch wenn der SGML-Hintergrund für einen Großteil der Anwender nicht oder kaum bekannt ist - HTML gilt als die killer application von SGML.[11]

Das was SGML und XML so mächtig macht, ist die Möglichkeit, dass jeder Benutzer sich seine eigene Markupsprache für jede Art von Dokumenten kreieren kann, mit denen er arbeitet. Dabei kann das Hauptaugenmerk auf die Benutzerfreundlichkeit der Elemente gelegt werden, was das Verstehen und die Verarbeitung durch Menschen stark vereinfacht. Durch intelligente Editoren (siehe dazu auch die Ausführungen im entsprechenden Kapitel dieser Arbeit), die in der Lage sind, das Markup teilweise oder komplett vor dem Benutzer zu verbergen, ist

[8] XMLPR Abschnitt 2.3 "Die neue alte Idee: Strukturorientiert schreiben"
[9] XMLPRO S. 10
[10] INFXML S. 1
[11] INFXML S. 1

es für Autoren sehr viel einfacher, syntaktisch korrekte Dokumente zu erstellen.

3.1 SGML

SGML ist eine extrem mächtige, aber auch sehr komplexe Metasprache, die nicht nur bei der US-Regierung weite Verbreitung fand.[12] Vor allem im Verlagswesen sind SGML und die flankierenden Standards wie HyTime (ISO 19744:1997) und DSSSL (ISO 10179:1996) nicht mehr weg zu denken. Die Verfügbarkeit von Daten in elektronischer Form macht es möglich, aus einem Datensatz mehrere Produkte zu erzeugen. Aus einer Adressenliste kann ein gedrucktes Verzeichnis, eine CD-ROM, eine Datenbank für den internen Geschäftsverkehr oder aber auch ein webbasiertes Angebot werden. Natürlich können aus den Datensätzen auch ganz banale Dinge wie Adressenaufkleber gedruckt werden - kurz gesagt: die Liste der Möglichkeiten ist nur durch den Einfallsreichtum der Anwender begrenzt.

Aber was macht SGML zu etwas Besonderem, wenn es um die Entwicklung von Markup(-Sprachen) geht? Drei Eigenschaften sind hervor zu heben: das deskriptive Markup, das Konzept der Dokumenttypen und die Unabhängigkeit in Bezug auf Hardware und Software.[13] Alle diese Eigenschaften finden sich auch in XML wieder. Daher werden sie im entsprechenden Abschnitt näher erläutert.

3.2 Die eXtensible Markup Language

1996 begann das World Wide Web Consortium W3C mit der Entwicklung einer erweiterbaren Markupsprache. Sie sollte die Flexibilität und Mächtigkeit von SGML mit der umfassenden Akzeptanz von HTML verbinden, nachdem man festgestellt hatte, dass die Bandbreite von SGML zu groß war und die Sprache weder einfach zu lernen noch zu implementieren war.[14] Das Ergebnis dieser Anstrengungen ist XML, eine Untermenge von SGML, mit einigen wenigen zusätzlichen Komponenten[15]. Das hat den Vorteil, dass sich das Team, das mit der Entwicklung der Sprache beschäftigt war, sich ganz auf die Merkmale konzentrieren konnte, die die Sprache leichter als ihre Vorgängerin machen sollen. Zehn Designvorgaben gab es für XML[16]:

1. XML sollte direkt über das Internet nutzbar sein. XML nutzt die Möglichkeiten, die das Internet seit der Einführung von SGML bietet: existierende und weit verbreitete Internetprotokolle, wie zum Beispiel die URLs und URIs und den internationalen Zeichensatz Unicode.
2. XML sollte eine große Anzahl von Anwendungen unterstützen. Im Gegensatz zu HTML

[12] XMLPRO S. 11
[13] TEIG Part I, 2.1 "What's Special about SGML?"
[14] WEBXML S. 249
[15] TEXWEB S. 248
[16] TEXWEB S. 249

sollte XML nicht nur für Browser optimiert sein, sondern auch für andere, nicht webbasierte Applikationen wie zum Beispiel Authoring, Präsentation und Datenbanken nutzbar sein.

3. XML sollte kompatibel zu SGML sein, um vorhandene Anwendungen weiter nutzen zu können.
4. Es sollte einfach sein, Parser und XML Prozessoren zu schreiben. Die Verarbeitung von SGML-Dokumenten durch Parser ist auf Grund der Komplexität der Sprache sehr schwierig, XML-Parser dagegen gibt es heute kostenlos und in reicher Auswahl.
5. Optionale Komponenten sollten in XML so wenig wie möglich oder gar nicht vorhanden sein. Optionale Komponenten bleiben nur so lange optional, bis sie wirklich implementiert werden. Ab da an müssen Routinen geschrieben werden, die diese Komponenten verarbeiten. Deshalb gibt es keinerlei solcher Komponenten in XML. Jeder XML-Parser soll in der Lage sein, jedes XML-Dokument zu verarbeiten.
6. XML-Dokumente sollen auch von Menschen lesbar und verstehbar sein. Durch Verwendung der offenen Standards ASCII und Unicode ist es möglich, den Editor seiner Wahl zu benutzen um XML-Dateien zu bearbeiten.
7. Das XML-Design sollte schnell verfügbar sein um inkompatible Erweiterungen wie im Fall von HTML gar nicht erst entstehen zu lassen.[17]
8. Das XML-Design sollte formal und prägnant sein.
9. XML-Dokumente sollten einfach zu erstellen sein.
10. Klarheit in der Darstellung des XML-Markups steht vor Kürze. Damit wird gewährleistet, dass XML leicht lesbar und zudem gut implementierbar ist.

Kurz zusammen gefasst: XML bietet 90% der Funktionalität von SGML mit nur 10% an Komplexität.[18]

Seit seiner Standardisierung als W3C Recommendation *(http://www.w3c.org/TR/REC-xml)* im Februar 1998 hat XML einen beeindruckenden Siegeszug in der Internet Community angetreten. Literatur dazu erscheint in rascher Folge.[19] Weitere, flankierende Standards wurden ebenso verabschiedet, namentlich vor allem XSL, das während der Entstehung dieser Arbeit zur W3C Recommendation *(http://www.w3.org/TR/2001/REC-xsl-20011015/)* geadelt wurde, XLink und XPointer. XML ist nicht erschaffen worden, um bestehende Standards im Web zu ersetzen (was viele anfangs befürchteten), sondern um dem Treiben im Internet eine solidere und gleichzeitig flexiblere Grundlage zu schaffen. Zu zeigen, dass eben dieses mit XML möglich ist, ist Teil dieser Arbeit.[20]

Wie auch SGML ist XML trotz des Namens keine Markupsprache, sondern ein Werkzeug-

[17] LXML S. ix
[18] TEXWEB S. 249
[19] TEXWEB S. 249
[20] LXML S. ix

kasten, um eigene Markupsprachen zu entwerfen. Jedes Dokument, das eine von einem Entwickler definierte Menge an Tags benutzt, ist demnach eine eigenständige XML-Anwendung und damit ein Objekt einer eigenen Metasprache. Ein Beispiel dafür ist HTML, dass als XHTML *(http://www.w3.org/TR/xhtml1)* in XML neu definiert wurde[21]. Ein wichtiger Ausblick auf die Zukunft von HTML ist allerdings sicherlich die Möglichkeit, XHTML-Dokumente genaus so modular *(http://www.w3.org/TR/xhtml1/#mods)* zu konstruieren, wie es mit XML möglich ist.

3.2.1 Das Prinzip hinter SGML und XML

XML wird bezeichnet als Werkzeug zur Datenspeicherung, als frei konfigurierbares Vehikel für Informationen jeder Art und als offener, sich entwickelnder Standard für jeden - vom Banker hin zum Webmaster.[22] Wenn man sich ein XML- oder SGML-Dokument ansieht, wird schnell ersichtlich, warum das so ist: Ein XML-Dokument ist einem HTML-Dokument nicht unähnlich. Ein Beispiel:

```
<?xml version="1.0"?>
<magisterarbeit>
<meta>
<title>Eine Magisterarbeit</title>
</meta>
<body>
<chapter>Dies ist der erste Satz einer Magisterarbeit.</chapter>
</body>
</magisterarbeit>
```

Selbst wenn man die Syntax nicht kennt, kann man doch erahnen, um was es hier geht, da die Tags, die Elemente, die den Inhalt umschließen, ihren Inhalt genauer beschreiben. Es dreht sich allen Anschein nach um eine Magisterarbeit, die aus Metadaten, hier einem Titel "Eine Magisterarbeit", und einem Hauptteil besteht. Zu dieser Syntax von XML-Dokumenten wird näheres im Abschnitt zur Document Type Definition gesagt. Hier wird ein Merkmal von XML deutlich: sich selbst beschreibende Daten (self-describing data), also deskriptives Markup.[23]

3.2.1.1 Deskriptives Markup

Beim deskriptiven Markup wie es SGML und XML erlauben, haben die Elemente (bestehend aus einer Anfangskennung, dem Elementeinhalt und einer Endkennung) beschreibenden Merkmale über ihren Inhalt.[24] Prozedurales Markup dagegen definiert Funktionen wie "an dieser Stelle zwei Zentimeter nach rechts einrücken".[25]

[21] XHTMLWD
[22] LXML S. 1
[23] XMLPRO S. 15
[24] XMLPRO S. 15
[25] TEIG Part I, 2.1.1 "Descriptive Markup"

XML trennt - genau wie SGML - streng zwischen Inhalt und Formatierung. Solche Angaben finden sich dann in Style Sheets, wie zum Beispiel in DSSSL. Durch die Trennung kann das Dokument von verschiedener Software verarbeitet werden, die jeweils die für sie relevanten Teile bearbeitet, ausgehend von der Bezeichnung der Elemente. Sieht man sich die für diese Arbeit entwickelte DTD an, erkennt man das Prinzip des deskriptiven Markups. Auch wenn niemand dazu gezwungen ist, ein Element, das Informationen über einen Namen enthält, `name` zu nennen, ist es doch eine große Hilfe, vom deskriptiven Markup Gebrauch zu machen.

Alle Regeln des Markups finden sich in einem zu definierendem Regelwerk wieder: der Dokumenttyp Definition DTD.

3.2.1.2 Dokumenttypen

Mit SGML wurde der Dokumenttyp und die ihn deklarierende Deklaration (Document Type Definition - DTD) eingeführt. Ein Dokument besitzt demnach einen Typ, ausgehend von seiner inneren Struktur. Ein kleines Abstract hat eine andere Struktur als ein Buch, daher sollte es auch einen eigenen Dokumenttypen haben. Die DTD ist das Regelwerk (die Grammatik), das den Aufbau des Dokuments beschreibt.[26] Parser können dazu eingesetzt werden, um zu kontrollieren, ob ein Dokument einem bestimmten Dokumenttypen auch wirklich entspricht, und ob die Anordnung der Informationseinheiten korrekt ist. Komplexe Entwicklerwerkzeuge unterstützen den Autor eines Dokuments, indem sie jeweils nur das Einfügen eines Elements erlauben, das an der Stelle zulänglich ist, und auf Fehler hinweisen. Zu einigen dieser Entwicklerwerkzeuge wird an anderer Stelle dieser Arbeit noch etwas gesagt werden. Eine Standard-Textverarbeitung kann diesen Komfort nicht bieten, da sie nie weiß, welche Art von Text der Autor schreiben möchte. Daher stehen hier immer alle Funktionen zur Verfügung, auch wenn bei moderner Software dieser Art nur der geringste Teil davon wirklich sinnvoll, geschweige denn notwendig ist.

3.2.1.2.1 Elemente

Eine Dokumenttyp Deklaration (DTD) ist wie eine kontextfreie Grammatik aufgebaut, in der Regeln festgelegt werden, wie die abstrakten und konkreten Informationseinheiten, die Elemente, angeordnet werden sollen. Darüber hinaus geben DTDs die Typen von Elementen an, die an der jeweiligen Stelle im Dokument eingesetzt werden dürfen. Diese Elemente werden benannt und weiter spezifiziert und durch Regeln in Verbindungen mit einander gebracht. Mit Hilfe von DTDs ist also eine *textuelle Informationsmodellierung* möglich.[27]

Es gibt zwei Formen von Elementen:

1. Daten-Elemente, die konkrete Informationen enthalten.
2. Container-Elemente, die andere Elemente beinhalten.

[26] WEBXML S. 26
[27] INFXML S. 3 f.

Die Daten, die in Daten-Elementen vorkommen dürfen, müssen spezifiziert werden. Der einfachste Fall von Daten sind Zeichenfolgen, in SGML und XML als Parsed Character Data, `#PCDATA` bezeichnet.[28] Durch die Bezeichnung der Elemente mit einem eindeutigen Namen **kann** die in ihnen enthaltene Information näher beschrieben werden. Sicher zu stellen dass die Information in einem Element `name` auch wirklich ein Name ist, dafür gibt es weder in SGML noch in XML Mittel.

Die Deklaration eines Daten-Elements, das eine Zeichenfolge enthalten kann, sieht so aus:

```
<!ELEMENT name (#PCDATA)>
```

Es handelt sich also um ein Daten-Element mit Namen `name`, das eine beliebige Zeichenfolge enthalten kann.

An dieser Stelle eine Anmerkung zur Notation: alle DTDs, die in dieser Arbeit gezeigt werden, sind nach der XML-Notation verfasst. Das hat zwei Gründe: Zum einen sind die Notationen beider Metasprachen sehr ähnlich und in weiten Teilen identisch. Zum anderen liegt das Hauptaugenmerk dieser Arbeit auf XML.[29]

Container-Elemente enthalten keine konkreten Daten, sondern andere Elemente. Ein Beispiel:

```
<!ELEMENT name (first_name, surname)>
```

Hier wird ein Element `name` definiert, das wiederum aus zwei Elementen besteht: dem Vornamen und dem Nachnamen. An dieser Stelle wird nichts darüber ausgesagt, welchen Elementtyps `first_name` und `surname` sind. Beide Elemente müssen zwingend an anderer Stelle der DTD definiert werden.

Innerhalb der runden Klammern ist das Inhaltsmodell, in dem Angaben darüber gemacht werden, wie sich die beiden Elemente zueinander verhalten. Im obigen Beispiel sind beide Elemente `first_name` und `surname` obligatorisch, das heißt sie treten beide genau ein einziges Mal auf. Außerdem folgt `surname` auf `first_name`.

Neben dem obligatorischen Elementstatus gibt es weitere:

- `first_name?` besagt, dass das Element `first_name` entweder einmal oder gar nicht auftreten kann, also fakultativ ist.
- `first_name+` besagt, dass das Element mindestens einmal, aber auch öfter auftreten kann.
- `first_name*` besagt, dass das Element beliebig oft auftreten kann, auch gar nicht.

Neben der Anordnung der Elemente durch Kommata gibt es auch noch den Konnektor "`|`", der einem exklusiven Oder entspricht. In SGML gibt es darüber hinaus noch den Konnektor `&`, der die Reihenfolge der beiden Elemente offen lässt.[30]

[28] INFXML S. 10. Das # in #PCDATA markiert ein vordefiniertes Schlüsselwort. Neben #PCDATA sind in SGML auch noch weitere Datentypen erlaubt, die es in XML nicht gibt.
[29] INFXML S.4. Für weitere Einzelheiten zur Unterscheidung siehe auch die entsprechenden Standards.

Weiter soll an dieser Stelle auf Elemente nicht eingegangen werden. Durch die kommentierte Darstellung der dieser Arbeit zu Grunde liegenden DTD sollten sich dem Leser weitere Möglichkeiten erschließen.

3.2.1.2.2 Attribute

Neben den Elementen gibt es als zweite Informationsstruktur die Attribute. Attribute machen in Form von Meta-Informationen Aussagen über Elemente.[31] Dadurch erlauben sie es, über einzelne Knoten im Dokumentbaum Aussagen zu machen. Ein Beispiel:

```
<!ELEMENT person (#PCDATA)>
<!ATTLIST person sex (male|female) #REQUIRED>
```

Hier wird ein Daten-Element `person` definiert, das vermutlich Informationen über eine Person, beispielsweise ihren Namen enthält. Zusätzlich erhält das Element ein Attribut `sex`, das Auskunft über das Geschlecht einer Person geben soll. Das geschieht durch das Schlüsselwort `ATTLIST`, dem Namen des Elements und dem Namen des oder der Attribute. Jedes Attribut muss einen Wert erhalten. Hier gibt es die Auswahl aus den beiden Werten `male` oder `female`. Einen anderen Wert kann das Attribut nicht erhalten. Mit Hilfe von `#REQUIRED` wird darüber hinaus festgelegt, dass das Attribut `sex` unbedingt bestimmt werden muss. Neben `#REQUIRED` gibt es auch noch das Schlüsselworte `#FIXED`, gefolgt von einem Wert, um Attributwerte als unveränderbare Standardwerte zu definieren. Ein Beispiel für die Verwendung von `#FIXED` in Attributen findet sich in dieser Arbeit beim Element `link`. Auch ist es möglich mit `#IMPLIED` festzulegen, dass das Attribut fakultativ ist. SGML kennt darüber hinaus noch den vierten Wert `#CURRENT`.[32] Innerhalb eines XML-Dokuments wird ein Attribut wie folgt eingefügt:

```
<person sex="male">Maik</person>
```

Im obigen Beispiel konnte der Wert des Attributs aus zwei vorgegebenen Möglichkeiten ausgewählt werden. Es sind aber auch andere Werte möglich.

```
<!ATTLIST person address CDATA #IMPLIED>
```

Mit `CDATA` kann als Attributwert eine beliebige Zeichenkette (Character Data String) eingesetzt werden. Die einzige Ausnahme ist, dass ein Attributwert vom Typ `CDATA` kein Markup enthalten darf.[33]

Die Verwendung von `ID`, `IDREF` und `IDREFS` erlaubt es, Beziehungen zwischen einzelnen Knoten im Baum herzustellen.

```
<!ATTLIST person personalnr ID #REQUIRED>
```

Hier hat jede `person` eine `personalnr`. Der Wert eines Attributs vom Typ `ID` muss einmalig im gesamten Dokument sein und darf auch bei anders benannten Attributen nicht ver-

[30] INFXML S. 10 f.
[31] TEIG Part 1, 2.6 "Attributes"
[32] INFXML S. 26
[33] XMLPRO S. 85

wendet werden. Soll eine Beziehung zwischen zwei Elementen hergestellt werden, kann ein Attribut vom Typ `IDREF` oder `IDREFS` benutzt werden.

```
<!ATTLIST mother mother_of IDREF #REQUIRED>
```

Mit `mother_of` kann eine Beziehung zwischen zwei Personen hergestellt werden in der Art, dass das hier nicht definierte Element `mother` als Attribut eine Referenz auf ein Element mit einem Attribut vom Typ `ID`, zum Beispiel `personalnr` im Element `person`, hat.

```
<person personalnr="m123">Klaus</person>
<mother mother_of="m123">Erika</mother>
```

`Ein Attribut vom Typ IDREFS` erlaubt eine Referenz auf mehrere Elemente, die ein Attribut vom Typ `ID` haben.

Durch die Möglichkeit per `ID` und `IDREF` sehr einfach Beziehungen zwischen Elementen herzustellen, sind Attribute ein Mittel *unabhängig von der Baumstruktur weitere Strukturierungen über die Informationseinheiten* zu legen.[34]

Ob Informationen besser in `#PCDATA`-Elementen oder in `CDATA`-Attributen gespeichert werden ist sicherlich eine Frage, die man nicht eindeutig beantworten kann. Durch das Informationsmodell der Elemente können Reihenfolgebeziehungen dargestellt werden, Attribute dagegen erlauben eine Beschränkung der Auswahlmöglichkeiten, wichtig zum Beispiel bei einem automatischen (Aus-)Sortieren. Eine Hilfe zur Entscheidung stellt sicherlich die weit verbreitete Auffassung dar, Attribute als *Informationen über Informationen* [35] zu sehen. Darüber hinaus geben Travis und Waldt weitere Richtlinien an, die bei der Entscheidung Element oder Attribut helfen können. So spricht für Attribute auch, dass ihre Werte durch einen Parser validiert werden können.[36]

3.2.1.2.3 Entitäten

Als dritten Typ zur Strukturierung von Informationen gibt es die Entitäten. Entitäten sind Platzhalter für Sonderzeichen, einzelne Wörter oder ganze Texte und Dateien. Das hat den Vorteil, dass sich der Autor eines SGML- oder XML-Dokuments Arbeit sparen kann, wenn er schon vorhandene Dokumentteile durch die entsprechende Entität mehrfach nutzen kann.

Entitäten sind vor allem da anzutreffen, wo es darauf ankommt, neben `#PCDATA` und `CDATA` weitere Datentypen in ein SGML- oder XML-Dokument einzubinden, zum Beispiel Bilder oder Grafiken. Es wird zwischen verschiedenen Entitäten unterschieden:[37]

- allgemeine- und Parameter-Entitäten
- interne und externe Entitäten

Allgemeine- und Parameter-Entitäten können sowohl intern als auch extern definiert werden.

[34] INFXML S. 29
[35] INFXML S. 32
[36] SGMLMIG S. 305 f.
[37] TEXWEB S. 263 ff.

Die Definition einer allgemeinen Entität kann nur innerhalb der DTD auftreten:

```
<!ENTITY XML "Extensible Markup Language">
```

Die Referenz auf eine solche Entität kann dagegen sowohl in der DTD als auch in einer Dokumenteninstanz durch `&XML;` erfolgen.

Parameter-Entitäten werden auch in dieser Arbeit benutzt. Ihre Definition ist ebenfalls nur innerhalb der DTD möglich.

```
<!ENTITY % bibtext "#PCDATA | link | quotation | abbrevation |
code">
```

Im Gegensatz zu den allgemeinen Entitäten ist auch der Aufruf einer Parameter-Entität auf die DTD beschränkt. Er erfolgt durch `%bibtext;`.

Interne Entitäten sind innerhalb ein und des selben Dokumentes. Sie werden bei der Laufzeit geparst. Ein Beispiel dafür ist die obige Abkürzungsverwendung. Die Entitätenreferenz bewirkt, dass die Referenz durch den Inhalt der Entität ausgewechselt wird. Externe Entitäten werden dagegen benutzt, um andere Daten zu referenzieren. Ein Beispiel dafür ist folgende Definition:

```
<!ENTITY bibliography SYSTEM "bibfile.xml">
```

Neben dem Schlüsselwort `SYSTEM` ist es auch möglich mit Hilfe von `PUBLIC` und der Angabe eines public identifiers und der optionalen Angabe einer URI Dateien über das Internet einzubinden.

3.2.1.3 Datenunabhängigkeit

Bei der Entwicklung von SGML sollte sichergestellt sein, dass Daten - unabhängig von der verwendeten Hard- und Software - leicht auf andere Systeme portierbar waren, ohne dass Informationen verloren gingen. Das wird durch zwei Merkmale von SGML realisiert: zum einen sind SGML-Dokumente ASCII-Dateien, dass heißt, diese Dateien sind auf jedem Rechner weltweit darstellbar und mit einem einfachen Texteditor zu öffnen. Zum anderen bietet SGML die Möglichkeit der Stringsubstitution durch Verwendung von Entitäten.[38]

3.2.1.4 Wohlgeformtheit und Gültigkeit

SGML-Dokumente haben ein Kriterium, dem sie genügen müssen: sie müssen gültig im Sinne der DTD sein, nach der sie erstellt wurden. Das gleiche gilt selbstverständlich auch für XML-Dokumente. Allerdings gibt es bei XML noch ein etwas schwächeres Kriterium: die Wohlgeformtheit. Von einem wohlgeformten (well-formed) XML-Dokument spricht man, wenn die Syntax eines XML-Dokuments den Anforderungen des W3C Standards, der XML Recommendation *(http://www.w3c.org/TR/REC-xml)* , genügt. Somit kommt eine große Anzahl an Anwendungen mit XML-annotierten Informationen zurecht.[39]

[38] TEIG Part 1, 2.1.3 "Data Independence"
[39] WEBXML S. 25

4 XML in der Anwendung: die Magisterarbeit

Nachdem über die theoretischen Grundlagen von XML genug gesagt worden ist, ist es an der Zeit, eine praktische Anwendung von XML zu erstellen: die Magisterarbeit. In diesem Kapitel wird die dieser Arbeit zu Grunde liegende DTD erläutert.

4.1 Die Document Type Definition

Die Document Type Definition (DTD) ist der Bauplan einer XML-Datei. In der DTD werden die zentralen Elemente, Attributlisten und Entitäten, die in einer nach dieser DTD erstellten XML-Datei vorkommen dürfen, genannt und beschrieben.

Die Entwicklung einer eigenständigen DTD ist nicht zwingend erforderlich. Es gibt mehrere frei verfügbare SGML und XML-DTDs, die sehr weit entwickelt sind. Zwei wichtige DTDs sind die der Text Encoding Initiative *(http://www.tei-c.org)* und DocBook *(http://www.docbook.org)*, das vor allem für Dokumentationen im OpenSource-Umfeld weit verbreitet ist. Neben den schon genannten Technologien, wie HTML und LaTeX, haben auch diese DTDs Anteil an der Erstellung dieser Magisterarbeit gehabt.

Die hier benutzte DTD ist modular. Das ermöglicht es, Teile, die sich schnell ändern können, wie zum Beispiel die Art ein Literaturverzeichnis oder ein Glossar zu erstellen, auszutauschen, ohne die zentrale DTD zu ändern. Da auch im XML-Dokument die entsprechenden Teile als Entitäten ausgelagert sind, ist es sehr einfach, inhaltlich wie strukturell neue Angaben zu machen.

Ausgangspunkt für diese DTD war die Überlegung eine Magisterarbeit komplett als XML-Datei zu realisieren. Diese Magisterarbeit sollte sowohl in eine Online- als auch in eine Printversion überführt werden können. Die Onlineversion sollte multimedial angereichert werden können, dass heißt, die Einbindung von Bildern, Audio- und Videodateien sollte möglich sein. Der Grund dafür ist, dass die hier verwendete DTD ursprünglich Hintergrund einer weiteren Magisterarbeit sein sollte, deren HTML-Version stark mit Multimedia-Elementen angereichert werden sollte. Überlegungen in diese Richtung finden sich vor allem im Element `asset`, das zur Einbindung externer multimedialer Daten dient.

Die DTD baut sich wie folgt auf. Das Wurzelelement (root) ist natürlich `magisterarbeit`. Die erste Frage, die es zu beantworten galt, war die nach dem Aufbau einer Magisterarbeit. Im Normalfall liegt eine Magisterarbeit in gedruckter Form vor, die Prüfungsordnung schreibt eine solche zwingend vor. Eine solche Arbeit besitzt zu allererst ein Deckblatt, auf dem der Titel der Arbeit, der Name des Autors, seine Anschrift und seine Matrikel-Nummer, sowie weitere formale Angaben vermerkt sind. Daran schließt sich die eigentliche Arbeit an, unterteilt in einzelne Kapitel wie Einleitung und Schluss. Am Ende der Arbeit steht ein Literaturverzeichnis, davor eventuell noch Anhänge. Strukturell ist eine Magisterar-

beit also in folgende Teile unterteilt:

- Informationen für die Titelseite
- die eigentliche Arbeit
- Anhang
- Literaturverzeichnis

Die Informationen, die sich später im Titel der Arbeit wieder finden, sind allesamt Meta-Informationen über die eigentliche Arbeit. Daher liegt es nahe, hierfür ein Element `meta` zu definieren. Die eigentliche Arbeit wird als ein weiteres Element, nennen wir es `body`, definiert. Für Anhänge wird ein Element `appendix` kreiert. Zu guter Letzt folgt das Literaturverzeichnis, `bibliography`.

```
<!ELEMENT magisterarbeit (meta, body, appendix, bibliography)>
```

Die einzelnen Elemente werden im folgenden näher erläutert.

4.1.1 meta

Das Container-Element `meta` enthält den Titel der Magisterarbeit. Daran schließt sich ein fakultativer Untertitel an (obwohl bei Magisterarbeiten ein Untertitel üblich ist). Informationen über den oder die Autoren der Arbeit dürfen ebenso wenig fehlen. Zu diesem Zweck gibt es ein Container-Element `authors`. Da es sich bei Instanzen dieser DTD um universitäre Arbeiten handelt, sollte auch die Angabe der Universität, an der diese Arbeit geschrieben wird, nicht fehlen. Das gleiche gilt für die Fakultät, den Fachbereich und den oder die Gutachter. Bei der Produktion der Arbeit ist eine Information über den aktuellen Stand sinnvoll, ein Element `revision`. Die Revisionsnummer dient vor allem internen Zwecken, eine Versionskontrolle ist damit leichter zu bewerkstelligen. Da eine fertige Magisterarbeit eine komplett abgeschlossene Arbeit sein sollte, ist das Element `revision` ebenfalls fakultativ ausgelegt.

```
<!ELEMENT meta (title, subtitle?, authors, university?, fa-
culty?, department?, supervisors?, revision?)>
```

Der Titel einer Arbeit besteht normalerweise aus `#PCDATA`, also einer Kombination aus Zeichen. Das Element `title` soll aber nicht nur den Titel der Magisterarbeit beinhalten - Titel finden sich schließlich auch bei der verwendeten Literatur oder bei einzelnen Abschnitten der Arbeit. `#PCDATA` als Elementinhalt ist allerdings noch zu unspezifisch. Ein Titel kann zum Beispiel ein Zitat oder einen Hyperlink enthalten. Dazu an anderer Stelle mehr. Zuerst soll `#PCDATA` genügen.

```
<!ELEMENT title (#PCDATA)>
```

Das Element `subtitle` ist nur ein semantisches Element. Rein formal ist ein Untertitel nichts anderes als ein Titel, allerdings ist aus strukturellen Aspekten die Unterscheidung wünschenswert. Für `subtitle` gilt natürlich ebenso das Gesagte wie für `title`. `#PCDATA` soll allerdings auch hier als Übergangslösung erst einmal genügen.

```
<!ELEMENT subtitle (#PCDATA)>
```

Wie oben schon angesprochen, ist das Element `authors` ein Container für einen oder mehrere Autoren. Innerhalb dieses Elements muss mindestens ein Autor vorkommen.

```
<!ELEMENT authors (author+)>
```

Der einzelne Autor hat verschiedene Merkmale, die wichtig sind. So hat jeder Autor einen Namen. Auch die Angabe einer Adresse ist bei Abgabe einer Magisterarbeit vorgeschrieben. Ebenso möglich ist die Auflistung von Telefonnummer(n), eMail-Adresse(n), und natürlich der Matrikelnummer. Allerdings wäre es äußerst kurzsichtig, ein Element `author` zu entwerfen, das nur auf den Autoren einer Magisterarbeit zutrifft und nur einen solchen zutreffend beschreibt. Eine Magisterarbeit beinhaltet auch immer Material, Quellen und Zitate von anderen Autoren. Also sollten auch diese Verfasser vom Element `author` erfasst werden. Bei den Autoren von Büchern, Zeitschriftenartikeln oder Internetveröffentlichungen sind jedoch nicht alle der oben genannten Angaben vorhanden. Daher sollten alle erweiterten Angaben optional sein.

Einige der Literaturangaben, die in dieser Arbeit verwendet werden, haben als Autorenangabe keine Einzelpersonen, sondern sind von Organisationen erstellt worden. Das trifft auf die W3C-Standards zu, die zwar jeweils von einem oder mehreren Autoren verfasst werden, aber als Veröffentlichung des gesamten Konsortiums gelten. Diesem Umstand sollte das `author` Element ebenfalls Rechnung tragen.

```
<!ELEMENT author ((name, address*, phone*, e-mail*, matri-
kel_nr?) | organization)>
```

Ein Name besteht aus Vor- und Nachnamen. Eine Person kann aber auch mehrere Vornamen haben. Allerding muss die Notation flexibel sein. Die Nennung des Namens auf dem Deckblatt einer Magisterarbeit besteht aus dem Vornamen und dem Nachnamen im Anschluss daran. Bei Nennung eines Namens im Literaturverzeichnis steht dagegen der Nachname vor dem Vornamen, durch Komma abgetrennt. Da in einer XML-DTD der Konnektor "&" für beliebige Reihenfolge fehlt, muss die geforderte Flexibilität anders erstellt werden.[40] Ein Name besteht also aus dem Vornamen (`first_name`), gefolgt vom Nachnamen (`surname`), oder aus dem Nachnamen, gefolgt vom Vornamen.

```
<!ELEMENT name ((first_name+, surname) | (surname,
first_name+))>
```

Mit den Elementen `first_name` und `surname` sind die ersten Daten-Elemente erstellt: eine weitere Unterteilung macht keinen Sinn.

```
<!ELEMENT surname (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
```

[40] INFXML S. 11

Bei der Nennung einer Organisation ist es sinnvoll, wahlweise ein Element `link` mit anzugeben. Schließlich verfügen fast alle größeren Institutionen über eine Webadresse.

```
<!ELEMENT organization (#PCDATA | link)*>
```

Die Angabe der Adresse auf dem Deckblatt erfordert die Nennung von Straße und Hausnummer, zusätzlich die Postleitzahl und den Wohnort. Eine Straße ohne Hausnummer oder nur eine Hausnummer machen wenig Sinn, daher ist eine Kopplung beider Angaben miteinander sinnvoll. Möglich ist auch die Nennung des Landes, in dem der Autor lebt, falls mehrere Autoren aus verschiedenen Ländern an der Arbeit beteiligt waren.

```
<!ELEMENT address ((street, nr)*, postal_code*, town*,
country*)>
```

Damit sind mit den Unterelementen von `author` und `address` weitere Terminale entstanden:

```
<!ELEMENT street (#PCDATA)>
<!ELEMENT nr (#PCDATA)>
<!ELEMENT postal_code (#PCDATA)>
<!ELEMENT town (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT e-mail (#PCDATA)>
<!ELEMENT matrikel_nr (#PCDATA)>
```

Zurück zu den anderen Elementen, die in `meta` enthalten sind. Der Großteil davon ist recht unspektakulär. Analog zu `authors` ist auch `supervisors` ein Container-Element für einzelne Gutachter.

```
<!ELEMENT university (%bibtext;)*>
<!ELEMENT faculty (%bibtext;)*>
<!ELEMENT department (%bibtext;)*>
<!ELEMENT supervisors (supervisor+)>
<!ELEMENT supervisor (name)>
<!ATTLIST supervisor grade CDATA #IMPLIED>
```

Wie schon angesprochen besitzt die DTD ein Element zur Versionskontrolle: `revision`. Eine solche Revisionsnummer wird als Datum definiert.

```
<!ELEMENT revision (date)>
```

Das Datum setzt sich aus Tag, Monat und Jahr zusammen. Bei der hier beschriebenen DTD wird das deutsche Datumsformat verwendet (Tag, Monat, Jahr). Bei der Nennung des Datums in Literaturangaben wird sich nur in den seltensten Fällen eine Angabe zu Tag und Monat finden. Daher sollten beide Angaben optional sein.

```
<!ELEMENT date (day?, month?, year)>
<!ELEMENT day (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

Vom Element `link` einmal abgesehen, sind damit alle Elemente für Meta-Angaben definiert. Dank XLink stehen in XML mächtige Linkingkonstrukte zur Verfügung, die weit über die Möglichkeiten der HTML-Links hinaus gehen.[41] Momentan kann die softwareseitige Unterstützung dieser Konstrukte allerdings nur als kaum vorhanden bezeichnet werden. Allenfalls ein simple link wird von einigen Browsern unterstützt. Daher kann der Wert des Attributs `xlink:type` fest auf `simple` gesetzt werden, was auch die Überführung nach HTML erleichtert. Ein Linkziel in Form von`xlink:href` ist zwingend vorgeschrieben, ein `xlink:title` sollte vorhanden sein.

Auch für die beiden Attribute `xlink:show` und `xlink:actuate` sind die Werte fest vorgegeben. Das Attribut `xlink:show` legt fest, ob das Linkziel im gleichen Fenster oder in einem neuen Fenster dargestellt werden soll. Im Beispiel wird bei jedem Link ein neues Fenster geöffnet.

`xlink:actuate` bestimmt, wann ein Linkziel angezeigt wird. Mögliche Werte sind neben dem hier verwendeten `onRequest`, also nach Anforderung durch den Benutzer, zum Beispiel durch Mausklick, auch `onLoad`.

Ebenfalls soll angegeben werden, ob der Link intern oder extern ist. Diese Unterscheidung ist sinnvoll, um in der HTML-Version mit Hilfe eines Cascading Style Sheets externe Links für den Benutzer besonders zu kennzeichnen. Als Inhalt kann ein `link` jedes andere Element beinhalten.

```
<!ELEMENT link ANY>
<!ATTLIST link
xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
xlink:type (simple) #FIXED "simple"
xlink:href CDATA #REQUIRED
xlink:title CDATA #IMPLIED
xlink:show (new) #FIXED "new"
xlink:actuate (onRequest) #FIXED "onRequest"
range (intern | extern) #REQUIRED>
```

4.1.2 body

Das Element `body` bildet den Hauptteil der Arbeit. Hier stellt sich zuerst die Frage, in welche Untereinheiten der Text zu untergliedern ist. Möglich wäre eine Unterteilung wie in HTML. Hier gibt es sechs Hierarchie-Ebenen, vorgegeben durch die Überschriften `<h1>` bis

[41] XMLPRO S. 322; für ausführlichere Informationen zu XLink S. 295 ff.

h6>. Eine solche Vorgehensweise hätte allerdings den Nachteil, dass nur die Überschriften den Text strukturieren würden. Besser ist da der Weg, den LaTeX geht. Der Text wird untergliedert in `\section`, `\subsection` und `\subsubsection`, dazu kommen je nach Dokumentklasse weitere Ebenen. Als sinnvoll hat sich eine Begrenzung der Hierarchisierung auf maximal fünf Ebenen erwiesen. Das bedeutet für eine Magisterarbeit: die größte Einheit ist das Kapitel, dann kommen Unterkapitel, in denen Abschnitte enthalten sein können. Diese wiederum können Unterabschnitte enthalten, die aus Unterunterabschnitten bestehen können.

Alle diese Einheiten müssen einen Titel besitzen. Notwendigerweise sollte vor einer weiteren Untergliederung mindestens ein Absatz stehen, in dem einleitende Worte oder die Gründe für eine weitere Unterteilung des Textes genannt werden sollten. Zusätzlich erhalten die Elemente eine `id`.

Für das Kapitel-Element `chapter` bedeutet das:

```
<!ELEMENT chapter (title, paragraph+, subchapter*)>
<!ATTLIST chapter id ID #REQUIRED>
```

In ähnlicher Weise können dann auch die anderen Einheiten definiert werden:

```
<!ELEMENT subchapter (title, paragraph+, section*)>
<!ATTLIST subchapter id ID #REQUIRED>
```

```
<!ELEMENT section (title, paragraph+, subsection*)>
<!ATTLIST section id ID #REQUIRED>
```

```
<!ELEMENT subsection (title, paragraph*, subsubsection*)>
<!ATTLIST subsection id ID #REQUIRED>
```

```
<!ELEMENT subsubsection (title, paragraph+)>
!ATTLIST subsubsection id ID #REQUIRED>
```

Alle diese Elemente besitzen den oben angesprochenen Absatz `paragraph`. Als kleinste Struktureinheit besteht eine `subsubsection` nur aus Absätzen. Daher sollte als nächstes `paragraph` definiert werden.

```
<!ELEMENT paragraph (#PCDATA)>
```

Spätestens an dieser Stelle fällt auf, was schon bei der Definition des Elements `title` angesprochen wurde: `#PCDATA` ist für einen ganzen Absatz zu unspezifisch. Einzelne Wörter sollten in ihrer Bedeutung ausgezeichnet werden können. Zum Beispiel muss es Elemente geben, die ein Zitat kennzeichnen, oder die für betonte Passagen in einem Text stehen. Quelltexte sollten in einer anderen Schriftart wiedergegeben werden. Als unverzichtbare Elemente einer Magisterarbeit haben sich die nachfolgenden heraus gestellt. Dabei wurde zu großen Teilen auf schon vorhandene Elemente aus `HTML` zurück gegriffen.

Ein Element `emph` für betonte Wörter:

```
<!ELEMENT emph (#PCDATA)>
```

Ein Element `strong` für besonders betonte Wörter:

```
<!ELEMENT strong (#PCDATA)>
```

Da nicht jeder Text einer Magisterarbeit vom Autor derselben geschrieben ist:

```
<!ELEMENT quotation (#PCDATA)>
```

Ein Element `code` um Quelltexte und Programmcode zu erfassen:

```
<!ELEMENT code (#PCDATA)>
```

Ein weiteres Element für Quelltexte, dieses Mal allerdings für eine komplette Zeile:

```
<!ELEMENT codeline (#PCDATA)>
```

Neben diesen Elementen erschien es sinnvoll, noch einige weitere zu definieren. Abkürzungen sollten als solche gekennzeichnet sein. Damit verbunden war der Wunsch nach einem Abkürzungsverzeichnis, ähnlich einem Glossar. Dazu besitzt das Element `abbrevation` ein Attribut `long_form`, das vom Typ `IDREF` ist. Damit ist es möglich, auf ein vorher definiertes Abkürzungsverzeichnis Bezug zu nehmen, das die ausgeschriebene Form der Abkürzung beinhaltet.

Dazu an dieser Stelle eine Anmerkung: Es gibt zwei Möglichkeiten, eine Beziehung zwischen der Abkürzung im Text und einem Abkürzungsverzeichnis herzustellen. Zum einen besteht die Möglichkeit, einer Abkürzung die ausgeschriebene Form als Element oder Attribut mitzugeben, die dann in ein automatisch generiertes Abkürzungsverzeichnis eingetragen wird. Zum anderen ist es möglich, ein Abkürzungsverzeichnis parallel zum Text zu erstellen, in dem die Abkürzungen und die ausgeschriebene Form erfasst werden. Ich habe mich für die zweite Variante entschieden, da es so genügt, ein Abkürzungsverzeichnis nur ein einziges Mal zu erstellen und beim Vorkommen einer Abkürzung im Text nur die jeweilige Referenz einzutragen. Per Konvention ist diese Referenz die Abkürzung selbst. Das bedeutet weniger Schreibaufwand als die komplett ausgeschriebene Form bei jedem Vorkommen einer Abkürzung im Text erneut zu erfassen.

```
<!ELEMENT abbrevation (#PCDATA)>
<!ATTLIST abbrevation long_form IDREF #REQUIRED>
```

Ebenso wird mit Ausdrücken verfahren, die in einem Glossar erklärt werden können. Das Element `term` bezeichnet einen Begriff, der im Glossar der Arbeit (sofern es denn eines gibt) näher erläutert wird. Dazu gibt es auch hier ein Attribut vom Typ `IDREF`.

```
<!ELEMENT term (#PCDATA)>
<!ATTLIST term definition IDREF #REQUIRED>
```

Listen sind ebenso ein wichtiger Bestandteil von schriftlichen Arbeiten. Sie bestehen aus einer optionalen Überschrift und mehreren Listenpunkten. Davon sollte mindestens einer vorhanden sein. Die hier verwendeten Listen sind analog zu denen in LaTeX und HTML.

```
<!ELEMENT list (title*, item)+>
<!ELEMENT item (#PCDATA)>
```

Allerdings sagt das bisherige Element `list` nur sehr wenig aus. Es ist nicht bekannt, ob es sich um eine nach Zahlen geordnete oder eine ungeordnete, sogenannte Bullet-List handelt. Um nicht wie in HTML zwei eigenständige Elemente (`ul` für die ungeordnete und `ol` für die geordnete Liste) definieren zu müssen, wird ein Attribut `type` eingefügt.

```
<!ATTLIST list type (ordered | unordered) #REQUIRED>
```

Nicht nur für das Element `quotation` wäre es sinnvoll, ein Element zu haben, das Referenzen definiert, ähnlich der Fußnote im gedruckten Text. Ein solches Element könnte auch weitere Anmerkungen beinhalten, die nicht direkt im Fließtext stehen sollen. Das hier verwendete Element heißt `reference`. Es beinhaltet als weiteres Element den eigentlichen Text. In der Printausgabe erscheint dieser Text als Fußnote am Ende der Seite, in der HTML-Ausgabe wird der Text zu Endnoten umgewandelt.

```
<!ELEMENT reference (reftext)>
```

Ähnlich wie bei den Abkürzungen kam auch hier schnell der Wunsch auf, eine Beziehung zwischen solchen Referenzen und den Angaben im Literaturverzeichnis herzustellen. Dazu dient ein Attribut `refered_to`, das vom Typ `IDREF` ist.

```
<!ATTLIST reference refered_to IDREF #IMPLIED>
```

Da `reference` auch Anmerkungen aufnehmen soll, die sich nicht auf eine bestimmte Quelle im Literaturverzeichnis beziehen, ist `refered_to` nicht als `#REQUIRED` angegeben. Durch die Beziehung zwischen den Elementen `reference` im `body` und dem entsprechenden `biblio_item` (definiert im Abschnitt zur `bibliography`) ist sichergestellt, dass nur aus solchen Quellen zitiert werden kann, die auch im Literaturverzeichnis eingetragen sind. Bleibt nur noch der Referenztext:

```
<!ELEMENT reftext (#PCDATA)>
```

Um multimediale Objekte wie Bilder, Sounds und Videos in den Text mit aufnehmen zu können, bedarf es weiterer Elemente. In HTML steht dafür das Element `img` zur Verfügung, das Bilder einbindet. Audio- und Videodaten werden eher stiefmütterlich behandelt und über Elemente wie `object`, `embed` (proprietär) oder per allgemeinem Link integriert. In dieser DTD werden alle drei Objekte mit einem Element in den Text integriert: `asset`. Dazu erhält `asset` einige Attribute. Mit Hilfe von `id` kann das jeweilige Objekt eindeutig identifiziert werden. Das ist nötig, um ein Abbildungsverzeichnis zu generieren. Die Angabe der Quelle und der Typ des Objekts sind ebenso notwendig. Mit Hilfe von `format` können die Dateiendungen angegeben werden. Dazu muss am Beginn der DTD jeweils eine Notation Deklaration für die hier aufgeführten Dateiformate stehen.[42]

[42] XMLPRO S. 88 f.

Aufgeführt sind hierbei die wichtigsten Multimedia-Dateitypen.

- gif - Das von Compuserve entwickelte Graphics Interchange Format, das sich im Internet vor allem für grafische Elemente wie Buttons durchgesetzt hat.
- jpg - Das von der Joint Photographics Expert Group entworfene Format, das das gängige Format für Fotos und andere Halbtonbilder in HTML-Dateien ist.
- rm und ram - RealMedia- bzw. Real Audio Media-Dateien sind weitverbreitete Formate, die vom RealMedia Player wiedergegeben werden, den es als PlugIn für die meisten Browser gibt.
- avi - Von Microsoft entwickeltes Videoformat. Standard in der Windows-Welt.
- mov - Quicktime Movie. Entwickelt von Apple und auch als PlugIn für Browser erhältlich.
- wav - Ein von Microsoft mit Windows eingeführte unkomprimiert speichernde Soundformat.
- mp3 - MP3 (eigentlich MPEG 1 Layer 3) ist ein weitverbreitetes verlustbehaftet komprimierendes Audioformat, das ursprünglich von der Fraunhofer Gesellschaft entwickelt wurde und im Internet vor allem durch Musiktauschbörsen weltweit berühmt und berüchtigt wurde.

Zumindest für die Einbindung von Grafiken ist es sinnvoll, Angaben über die originale Abmessung eines Bildes in Form von Höhe und Breite in Pixeln anzugeben. Für eine spätere Überführung in eine Druckausgabe mit Hilfe von XSL-FO und FOP *(http://xml.apache.org/fop)* hat es sich darüber hinaus bewährt, zusätzlich die Maßen in Millimetern mit anzugeben. Traditionellerweise sollte auch ein alternativer Text für die Benutzer vorhanden sein, die auf Multimedia-Objekte verzichten wollen oder müssen. Insgesamt ergibt sich daraus folgende Definition von `asset`:

```
<!ELEMENT asset ANY>
<!ATTLIST asset
id ID #REQUIRED
source CDATA #REQUIRED
type (video | audio | image) #REQUIRED
format NOTATION (gif|jpg|rm|ram|avi|mov|wav|mp3) "gif"
width CDATA #IMPLIED
height CDATA #IMPLIED
width_in_mm CDATA #IMPLIED
height_in_mm CDATA #IMPLIED
alt CDATA #REQUIRED>
```

Damit wäre der Hauptteil der Arbeit erledigt. Zur Vervollständigung wird noch ein Element `comment` definiert. Es ist in XML zwar möglich, wie in HTML Kommentare durch `<!--` und `-->` zu kennzeichnen. Solche Kommentare erscheinen aber nicht im Text, wenn

man mit einem grafischen Editor wie zum Beispiel XMetaL *(http://www.softquad.com)* arbeitet. Für Anmerkungen wie Korrekturangaben, die im Laufe des Entstehungsprozesses einer Magisterarbeit immer wieder vorkommen, hat sich `comment` als sinnvoll erwiesen.

```
<!ELEMENT comment (#PCDATA)>
```

Der Großteil der Elemente einer Magisterarbeit ist damit definiert. Allerdings sind einige Elemente noch immer nicht zufrieden stellend gelöst. Abkürzungen, die innerhalb eines Zitates stehen, werden nicht erfasst. Ebenso wenig Links, die in einer Liste vorkommen können. Die Lösung dieses Problems ist die Definition von internen Parameter-Entitäten.

```
<!ENTITY % text "#PCDATA | link | asset | emph | strong | quota-
tion | abbrevation | code | codeline | reference | list | com-
ment ">
```

Nach Abänderung der Elemente `paragraph`, `emph`, `stong`, `quotation`, `abbrevation`, `item`, `reftext` und `comment` in

```
<!ELEMENT paragraph (%text;)*>
```

ist es möglich, auch verschachtelte Elemente zu behandeln.

Für die Elemente `title` und `subtitle` wird eine weitere Parameter-Entität definiert:

```
<!ENTITY % bibtext "#PCDATA | link | quotation | abbrevation |
code">
```

Damit wird sichergestellt, dass im Titel einer Literaturangabe sowohl keine betonten oder stark betonten Elemente, vor allem aber keine Listen und Multimedia-Objekte enthalten sind.

4.1.3 appendix

Im Anhang einer Magisterarbeit stehen Informationen, die zwar zu der Arbeit gehören, aber nicht im Hauptteil (`body`) stehen sollen. Dazu gehören in diesem konkreten Fall auf jeden Fall sämtliche der Arbeit zu Grunde liegenden Quelltexte, also die DTD und die Style Sheets zur Überführung in die Online- und Printausgabe.

Solche besonderen Informationen wurden in diesem Fall von den anderen Anhängen getrennt und in ein eigenes Element `apps` ausgelagert. Daran schließen sich optional ein Glossar `glossary`, das schon genannte (optionale) Abkürzungsverzeichnis `abbrevations` und ein Abbildungsverzeichnis `images`.

```
<!ELEMENT appendix (apps*, glossary?, abbrevations?, images?)>
```

Die sogenannten allgemeinen Anhänge `apps` sind aufgebaut wie ein normaler Teil der Arbeit, enthalten also mindestens ein Kapitel.

```
<!ELEMENT apps (chapter+)>
```

Das Glossar enthält einen Titel und besteht ansonsten aus mindestens einem zu erklären-

dem Begriff, hier `gloss_item` genannt. Wie die sonstigen strukturellen Teilelemente der Arbeit, `chapter`, `subchapter`, `section`, `subsection` und `subsubsection` enthält auch das Glossar eine eindeutige `id`, was die spätere automatische Generierung eines Inhaltsverzeichnis erleichtert.

```
<!ELEMENT glossary (title, gloss_item+)>
<!ATTLIST glossary id ID #REQUIRED>
```

Ein `gloss_item` besteht naturgemäß aus dem Begriff `term` und den ihn näher erläuternden Angaben, hier `definition` genannt.

```
<!ELEMENT gloss_item (term, definition)>
<!ATTLIST gloss_item def ID #REQUIRED>
```

Ein anderer gangbarer Weg wäre die Erweiterung des Elements `list` um den Typ `definition` gewesen, wie es ihn auch in HTML gibt. Da eine solche `definition list` allerdings zumindest in dieser Arbeit im Fließtext nicht genutzt wird, erschien es sinnvoller ein separates Element zu gestalten.

Der zu definierende Begriff `term` wurde ja bereits im `body` definiert. Er kann alle Elemente enthalten, die in der Parameter-Entität `%bibtext;` definiert sind.

Die Definition eines Begriffs kann dagegen auf den vollständigen Bestand an Elementen zugreifen. Das ist sinnvoll, da es durchaus vorkommen kann, dass zur Definition Listen oder auch grafische Elemente herangezogen werden.

```
<!ELEMENT definition (%text;)*>
```

Damit wäre ein Glossar komplett. Ganz ähnlich ist das Abkürzungsverzeichnis aufgebaut.

Es enthält ebenfalls einen Titel, sowie mindestens ein Abkürzungselement `abb_item`.

```
<!ELEMENT abbrevations (title, abb_item+)>
<!ATTLIST abbrevations id ID #REQUIRED>
```

Ein Abkürzungselement besteht aus der Abkürzung, die ja schon im `body` definiert wurde, und der ausgeschriebenen Form.

```
<!ELEMENT abb_item (abbrevation, longform)>
```

Die ausgeschriebene Form enthält die in der Parameter-Entität `%bibtext;` enthaltenen Elemente. Eine Verwendung von `%text;` ist hier nicht angeraten, da weder die Verwendung von multimedialen- noch List-Elementen nötig ist.

```
<!ELEMENT longform (%bibtext;)*>
```

Wichtig ist allerdings, dass `longform` eine eindeutige `id` erhält, anhand derer eine Referenz durch das Element `abbrevation` möglich ist.

```
<!ATTLIST longform id ID #REQUIRED>
```

Noch eine Anmerkung zum Abkürzungsverzeichnis und den darin enthaltenen Einträgen: im Rahmen dieser Arbeit wurden nur solche Abkürzungen in das Abkürzungsverzeichnis aufgenommen, die

1. mehrmals im Text vorkommen,
2. ohne eine Erklärung im unmittelbar anschließenden Text stehen und
3. nicht dem allgemeinen Sprachgebrauch des Deutschen entnommen sind.

Das bedeutet, dass Abkürzungen wie "z.B." für "zum Beispiel" nicht Eingang in das Abkürzungsverzeichnis gefunden haben.

Das Abbildungsverzeichnis `images` wird automatisch generiert. Dennoch erschien es aus zwei Gründen sinnvoll, ein Element `images` zu definieren: Zum einen wird durch ein Element `title` innerhalb von `images` gewährleistet, dass der Titel in der jeweiligen Sprache ausgegeben wird, in der auch das restliche Dokument verfasst ist. Mit einem automatisch generierten Titel wäre das nur durch einen Eingriff in die beiden XSLT Style Sheets gewährleistet. Zum anderen ist es so möglich, `images` aus den schon oben erwähnten Gründen ebenfalls eine eindeutige `id` zu vergeben.

```
<!ELEMENT images (title)>
<!ATTLIST images id ID #REQUIRED>
```

Mit der Definition von `images` ist der Anhang abgeschlossen. Als letztes Element der Magisterarbeit folgt das Literaturverzeichnis.

4.1.4 bibliography

Das Literaturverzeichnis am Ende einer Magisterarbeit beinhaltet sämtliche in der Arbeit genutzten Quellen, egal welcher Art. Dabei ist es vom Aufbau her identisch mit dem Glossar und dem Abkürzungsverzeichnis. Es enthält ebenso einen Titel, mindestens einen bibliografischen Eintrag und eine eindeutige `id`.

```
<!ELEMENT bibliography (title, biblio_item+)>
<!ATTLIST bibliography id ID #REQUIRED
```

Die einzelnen Einträge `biblio_item` beinhalten mehrere Angaben zur verwendeten Quelle. Auf jeden Fall müssen der oder die Autoren genannt werden, ebenso der Titel und ein eventuell vorhandener Untertitel. Bei Zeitschriftenartikeln reicht es nicht, Autor, Titel und Untertitel zu nennen, eine Angabe zur Quelle ist ebenso erforderlich. Ein Verlag, bei dem eine gedruckte Quelle erschienen ist und der Ort desselben sollten ebenso genant werden. Da bei Internetdokumenten in der Regel solche weiteren Angaben nicht vorhanden sind, sind die letzten genannten Elemente optional. Nicht optional ist dagegen die Angabe eines Datums. Bei Büchern sollte hier das Jahr des Drucks stehen, bei Zeitschriftenartikeln der Monat und das Jahr, bei Internetdokumenten ist es üblich, das Datum des letzten Besuchs auf der Seite zu vermerken.

```
<!ELEMENT biblio_item (authors, title, subtitle*, source*, pu-
blisher*, address*, date)>
```

Da mit dem Attribut `refered_to` des Elements `reference` Bezug auf ein `biblio_item` genommen werden soll, ist ein Attribut vom Typ `ID` zwingend erforderlich. Weiterhin ist es sinnvoll durch ein Attribut `type` die Quelle weiter zu spezifizieren.

```
<!ATTLIST biblio_item
designator ID #REQUIRED
type (Book | Inbook | Journalarticle | Proceedings | Inprocee-
dings | Dissertation | Phdthesis | Mastersthesis | Techreport |
Manual | Newspaperarticle | AV | Internet | Unpublished)
#REQUIRED>
```

```
<!ELEMENT source (%bibtext;)*>
<!ELEMENT publisher (%bibtext;)*>
```

4.1.5 Modularität

Eingangs wurde erwähnt, dass die hier benutzte DTD modular ist. Bisher war davon noch wenig zu sehen. Die Modularität wird dadurch erreicht, dass Teile der DTD in separate Dateien ausgelagert werden. Das ist möglich, indem man die entsprechenden Dateien per Parameter-Entität in die Basis-DTD einbindet. Bei der hier verwendeten DTD boten sich vor allem das Glossar (`glossary`), das Abkürzungsverzeichnis (`abbrevations`) und das Literaturverzeichnis (`bibliography`) dazu an.

Dazu werden zu Beginn der DTD `magisterarbeit.dtd` die entsprechenden Entitäten definiert:

```
<!ENTITY % gloss SYSTEM "glossary.ent">
<!ENTITY % abb SYSTEM "abbrevations.ent">
<!ENTITY % bib SYSTEM "bibliography.ent">
```

Durch Aufruf der Entitäten vor der Verwendung werden die entsprechenden Teile der DTD geladen. Es genügt also, vor der Definition des Elements `appendix` `%gloss;` `%abb;` und anschließend `%bib;` in die DTD einzufügen. Der Vorteil ist die strikte Trennung zwischen Inhalt im `body` Element und den formaleren Teilen wie Anhang und Literaturverzeichnis. Dadurch wird es sehr schnell und einfach möglich, diese Elemente neu zu formulieren, um sie an andere Gegebenheiten anzupassen.

Eine weitere Modularität findet sich auch beim XML-Dokument selbst. Die genannten Teile der Magisterarbeit, die auf separate Teile der DTD beruhen, sind ebenfalls als einzelne XML-Dateien ausgelegt, die per Entität in das Dokument eingefügt werden. Das erleichtert eine Neustrukturierung der betreffenden Elemente, da am Hauptteil weiterhin gearbeitet werden kann und nur kleinere Dateien geändert werden müssen.

Leider unterstützen nur wenige XML-Werkzeuge eine Einbindung von Teilen der DTD

und der XML-Datei durch Einsatz von Parameter-Entitäten. Exemplarisch seien zwei aus einer Vielzahl im nächsten Abschnitt vorgestellt.

5 XML-Werkzeuge

Wohl selten hat die Industrie und die freie Entwicklergemeinde auf eine neue Softwaretechnologie so dankbar und schnell reagiert wie auf XML. Schon nach kurzer Zeit gibt es eine (im Gegensatz zu SGML reiche) Auswahl an freier und kommerzieller Software, die bei der Erstellung und Erweiterung von XML-Dokumenten unterstützend zur Seite steht.

5.1 XMetaL

XMetaL von der Firma Softquad *(http://www.softquad.com)* (jüngst von der kanadischen Firma Corel *(http://www.corel.com)* aufgekauft) nimmt zu Recht eine herausragende Stellung unter den kommerziellen XML-Editoren ein - und das nicht nur aufgrund seines Preises. Wenn auch nicht so vielseitig wie zum Beispiel XMLSpy *(http://www.xmlspy.com)*, so ist XMetaL sehr hilfreich und durchdacht in genau einer Disziplin: Design und Überarbeitung von XML-Daten. Dem Benutzer stehen teils mächtige Werkzeuge zur Auswahl, die bei der Erstellung von annotierten Dokumenten sehr hilfreich sind. Eine Auswahl an verschiedenen Sichtweisen auf das Dokument erleichtert das Arbeiten. Neben einer reinen Textansicht und einer Vorschau im Browser sind vor allem die Darstellung mit hervorgehobenen Tags (siehe Screenshot), sowie eine Ansicht komplett ohne Tags (*Normal View*) sehr hilfreich. In beiden zuletzt genannten Darstellungen unterstützt XMetaL das Layouten mit Hilfe von Cascading Style Sheets. Dadurch können strukturelle Elemente schon bei der Eingabe besonders deutlich gemacht werden. Im Rahmen der Erstellung dieser Arbeit war es von großem Vorteil, Anmerkungen (`comment`) durch einen gelben Hintergrund hervorzuheben, damit sie nicht übersehen wurden.

Werkzeuge wie der *Attribute Inspector*, die *Element List* und die *Structure View* helfen ebenfalls, damit vor allem das Überarbeiten von vorhandenen Dokumenten mit möglichst geringen Zeitaufwand verbunden ist.

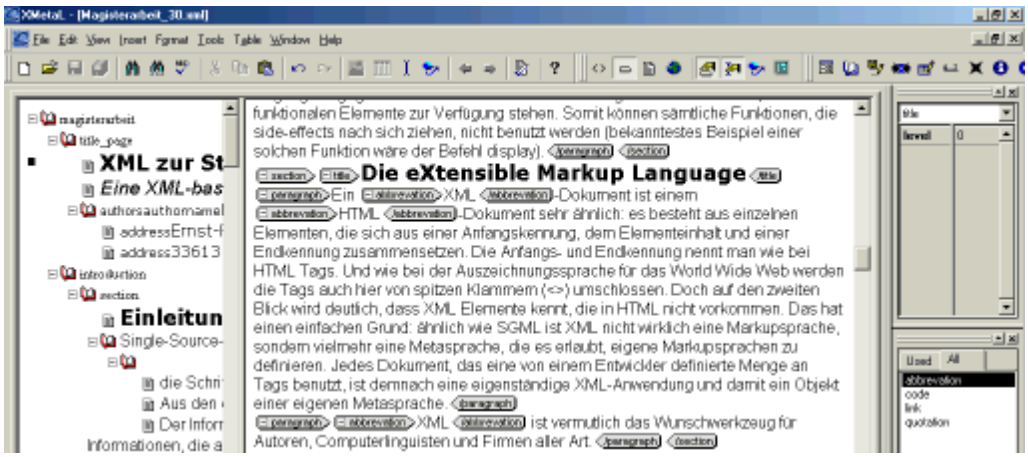

Abbildung 4: *XMetaL in der Ansicht "Tags On"*

Innerhalb von XMetaL ist es darüber hinaus möglich, Elementen bestimmte Funktionen zu zuweisen (*Treat As* im Untermenü *Customizations*). Ein Beispiel dafür ist die Funktion *Treat As Image*, so dass ein Benutzer ganz einfach per Schaltfläche in der Menüleiste ein Bild einfügen kann, ohne direkt das dahinter stehende Element zu bemühen. Neben der Arbeitserleichterung kann eine solche Anpassung nützlich sein, wenn der Elementname und seine inhaltliche Bedeutung nicht übereinstimmen.

Neben den genannten Aspekten fällt an XMetaL auch positiv auf, dass er Sonderzeichen wie "<" oder Umlaute maskiert, der Benutzer muss sich wie bei einem WYSIWYG HTML-Editor keine Gedanken darum machen. Allerdings kann auch ein Nachteil diese Maskierung betreffend nicht verschwiegen werden: versucht man in XMetaL eine öffnende spitze Klammer "<" über die Zwischenablage aus einem anderen Windowsprogramm einzufügen, stellt sich XMetaL stur. Das ist sehr ärgerlich, wenn XML- oder HTML-Quelltextzeilen wie in dieser Arbeit Teil des Dokuments sind. Sind mehrere Elemente in einer Zeile enthalten, fügt XMetaL auch nur bis zur öffnenden spitzenden Klammer "<" ein. Auch ist das Programm nicht in der Lage, solche Sonderzeichen (auch deutsche Umlaute) über die "Suchen"-Funktion im Dokument zu finden, da die interne Programmdarstellung auf dem XML-Quelltext im UTF-8 Format erfolgt. Komplette Unicode-Unterstützung (UTF-16) bietet XMetaL erst mit der seit August verfügbaren Version 2.1 SP1, allerdings auch nur unter Windows NT/2000.

Dafür ist XMetaL als einziger dem Autor bekannter XML Editor in der Lage, die in dieser Arbeit verwendeten Referenzen aufzulösen. Sowohl Emacs als auch XMLSpy in der Version 3.5 zeigen nur die Referenz (also "`&bibliography;`") an. XMetaL zeigt dagegen ein Rechteck mit dem Namen der Referenz und öffnet per Doppelklick das referenzierte Dokument.

Als Nachteil hat sich allerdings die nicht vorhandene Unterstützung von Namespaces durch XMetaL erwiesen. Dem wurde beim `link` Element begegnet, indem hier der Namespace fest als Attribut vorgegeben wird.

Im Gegensatz zu der ein oder anderen weit verbreiteten Textverarbeitung zeigt sich XMetaL als wahres Arbeitstier. Bei der gesamten Produktion dieser Arbeit kam es zu keinem einzigen Absturz der Software, so lange mit einem gültigen XML-Dokument gearbeitet wurde. Bei Änderungen an der DTD (und Löschen der zugehörigen internen .rlx-Datei) und Neueinlesen der XML-Datei stürzte XMetaL allerdings beim Versuch, die Datei zu validieren kommentarlos ab, sobald auch nur ein einziges nicht (mehr) erlaubtes Element in der Datei vorhanden waren. XMLSpy dagegen zeigte den Fehler korrekt an, weshalb es sich bei Umgestaltungen der DTD als erste Wahl erwies. Fairerweise muss man sagen, dass beim Arbeiten an einem Dokument die DTD normalerweise auch nicht mehr geändert wird.

Trotz der genannten Mängel bleibt festzustellen, dass XMetaL durch seine vielen durchdachten Funktionen vor allem bei der Produktion und Bearbeitung von XML-Dokumenten die erste Wahl ist.

5.2 Emacs

Der GNU Emacs und sein Vetter, XEmacs, im folgenden der Einfachheit halber nur Emacs genannt, haben sich dank einer rege tätigen Entwicklergemeinde als unglaublich vielseitige Werkzeuge des Programmier- und Editieralltags herausgestellt. Ursprünglich aus der UNIX-Welt stammend, ist Emacs auch in der Windows-Welt immer populärer geworden. Es ist sicherlich nicht übertreiben, wenn an dieser Stelle behauptet wird, das es wenig gibt, zu dem Emacs nicht im Stande ist. Das liegt daran, dass der Editor durch Module in der Programmiersprache LISP relativ einfach erweiterbar ist. Neue Module und ihr Verwendungszweck können dank einer textbasierten Konfigurationsdatei (.emacs bzw. _emacs) in LISP-Syntax sehr schnell in das System eingebunden werden. Das hat Emacs den Ruf des Schweizer Taschenmesser unter den Editoren eingebracht, wobei man fairerweise dazu sagen muss, dass es sich hier um ein Taschenmesser in der Größe einer Kettensäge handelt.[43]

Wenn im Zusammenhang mit der Bearbeitung von SGML- oder XML-Dokumenten der Emacs genannt wird, dann ist es eigentlich nicht der Editor selbst, der den Umgang mit derart annotierten Daten erleichtert, sondern vor allem der PSGML-Modus *(http//sourceforge.net/projects/psgml)* von Lennart Staflin. Dieser sogenannte Major-Mode bringt dem Emacs bei, SGML- und auch XML-DTDs und -Dokumente zu bearbeiten. Ist der Modus installiert, startet PSGML je nach Konfiguraton in der .emacs beim Laden eines Dokumentes. Neben einem einfachen DTD-Parser bringt das Modul eine eigene Menuzeile mit, in der Elemente und Attribute an der aktuellen Cursor-Position eingefügt werden können. Dazu liest PSGML die DTD auf Wunsch ein und speichert sie in einem internen Format. Ist der Modus aktiv, kann man mit seiner Hilfe einzelne Regionen des Textes ausblenden, um zum Beispiel die interne Struktur eines Dokuments besser darstellen zu können, oder - wie in XMetaL - die an dieser Stelle gültigen Tags anzeigen lassen.

[43] TEXWEB S. 272

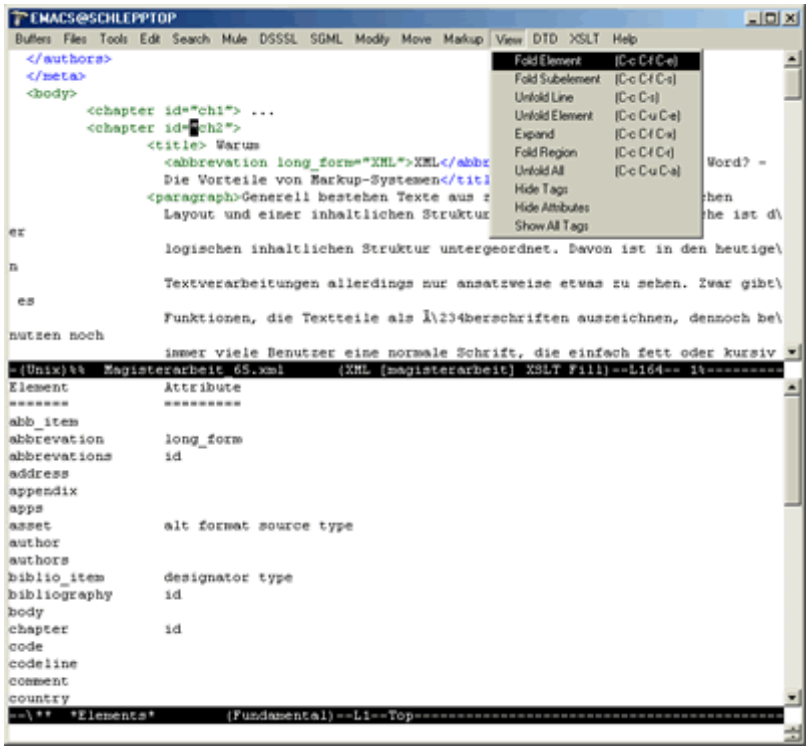

Abbildung 5: *Emacs im PSGML-Mode*

Aber es gibt noch weitere Helfer, die den Emacs zur Bearbeitung von SGML- und XML-Dokumenten erweitern. tDTD *(http://www.mulberrytech.com/tdtd/)* von Tony Graham ist ebenfalls ein sogenannter Major-Mode, der PSGML in manchen Teilen ergänzen kann. tDTD kann die Elemente der aktuellen DTD inkl. Typ anzeigen, ebenfalls die DTD parsen und allgemeine Informationen dazu ausgeben.

Zu guter Letzt kann auch die Verarbeitung von XSL-Style Sheets mit passenden Emacs-Modi erleichtert werden: xslide *(http://www.menteith.com/xslide/)* ist ebenfalls ein Major-Mode, der den Emacs um XSL-Funktionalität erweitert. Einen Schritt weiter geht XSLT-process *(http://sourceforge.net/projects/xslt-process)* von Ovisiu Predescu, ein Minor-Mode, der Emacs um gleich mehrere XSLT Prozessoren bereichert, die auch im Debugger-Modus arbeiten können. Damit kann die Transformation Schritt für Schritt mit verfolgt werden. Auch die Umwandlung nach XSL-FO und PDF ist per Menupunkt möglich.

Damit erweist sich Emacs in manchen Punkten als XMetaL ebenbürtig oder sogar überlegen, zumindest was die Vielfalt an Funktionen angeht. Unschlagbar ist natürlich das Preis-Leistungs-Verhältnis: schließlich sind sowohl Emacs als auch alle genannten Zusatzmodule kostenlos erhältlich. Einziger schwer wiegener Nachteil: die nicht vorhandene Unicode-Unterstützung.

6 Transformation von SGML und XML

Die in dieser Arbeit angesprochenen Markup-Sprachen SGML und XML machen beide keinerlei Angaben über die Art der Präsentation ihrer Elemente. Das machte es nötig, dass für die Ausgabe von derart annotierten Daten weitere Technologien entwickelt wurden. 1990 wurde die Formatting Output Specification Instance (FOSI) *(http://www-cals.itsi.disa.mil/core/standards/28001C.pdf)* vom amerikanischen Militär veröffentlicht. Der Standard erlaubte die Darstellung (das Rendern) von SGML-Dokumenten. Nach der Entwicklung von FOSI wurde das Engagement zur Verabschiedung eines Standards zur generellen Transformation von SGML-Dokumenten und deren Formatierung weiter verstärkt. Das Ergebnis dieser Anstrengungen war die Document Style Semantics and Specification Language DSSSL.

Mit Hilfe von DSSSL ist nicht nur möglich, Transformationen vorzunehmen, es sind auch Formatierungs- und Abfrage-Elemente in DSSSL enthalten. Da XML ein Subset von SGML ist, ist DSSSL auch zur Verarbeitung von XML-Dokumenten geeignet. Da sich allerdings DSSSL und DSSSL-online *(http://metalab.unc.edu/pub/sun-info/standards/dsssl/dssslo/do960816.htm)* , das speziell für Internet-spezifische Transformationen entwickelt worden ist, auch aufgrund der SCHEME-Syntax nicht durchsetzen konnten, wurde im August 1997 der Vorschlag einer eXtensible Style Sheet Language, XSL gemacht.[44]

In beide Technologien soll an dieser Stelle kurz eingeführt werden. Anhand dieser Arbeit wird die Überführung eines XML-Dokuments mit Hilfe von XSL ausführlich dargestellt.

6.1 DSSSL

DSSSL ist seit 1996 ISO-Standard (ISO/IEC:10179:1996) und besteht aus vier Sprachen:

- style language
- transformation language
- expression language
- query language (SDQL)

Diese vier Sprache erlauben es SGML-Texte zu annotieren, zu formatieren und gegebenenfalls zwischen verschiedenen DTDs zu transformieren.[45] Die expression language ist die Grundlage der anderen Sprachen. Ihre Syntax lehnt sich eng an SCHEME an, einem LISP-Dialekt. LISP ist als Programmiersprache unter Computerlinguisten recht bekannt und wird auch zu anderen Zwecken in der Computerlinguistik eingesetzt. Auch ist ausreichend Literatur vorhanden, um sich in die Sprache einzuarbeiten. Allerdings ist die DSSSL expression language gegenüber dem Standard-SCHEME eingeschränkt in der Art, dass nur die funktionalen

[44] TEXWEB S. 290
[45] TEXWEB S. 312

Elemente zur Verfügung stehen. Somit können sämtliche Funktionen, die side-effects nach sich ziehen, nicht benutzt werden (bekanntestes Beispiel einer solchen Funktion wäre der Befehl display).

Es ist auch möglich, DSSSL zu verwenden, ohne die expression language zu erlernen. Das Ergebnis ist dann allerdings eine Funktionalität, die ungefähr der von CSS entspricht.

Zentrales Objekt der DSSSL style language ist der Flow Object Tree, eine abstrakte Repräsentation, wie das SGML-Dokument und die Formatierungsanweisungen miteinander verbunden werden. Die Knoten dieses Baumes bilden Flow Objects. Sie beschreiben das Layout des Dokuments mit Hilfe von Konstrukten wie Seiten-Sequencen, Absätzen oder Tabellen. Jedes der Flow Objects hat Angaben zu Maßen, Abständen und Rändern, ähnlich wie sie in CSS und in XSL-FO vorhanden sind.

Wird ein Flow Object formatiert, ist das Ergebnis eine Abfolge von Areas. Areas sind rechteckige Boxen, in denen der Inhalt steht. Es gibt zwei Arten von Areas: display areas und inline areas. Die display areas erstellen Container mit vorgegebener oder variabler Größe, inline areas sind Teilelemente einer Zeile.[46] Beide Konstrukte sind in analoger Form auch in XSL-FO enthalten. Daher wird an dieser Stelle nicht näher darauf eingegangen.

Die transformation language ist eine Sprache, um SGML-Dokumentinstanzen einer DTD in Instanzen einer anderen DTD zu überführen. Das kann zum Beispiel eine Überführung nach HTML sein. Auch die transformation language arbeitet auf Bäumen. Die Datenstrukur heißt hier allerdings grove (DSSSL Graph Representation of Property Values). Der grove kann als Baum von Bäumen beschrieben werden, Mathematiker und Informatiker werden eher mit der Bezeichnung *possibly cyclid directed graph of nodes* warm.[47] Jeder Knoten im Baum hat ein Set von Eigenschaften, die entweder atomare Werte enthalten (dann handelt es sich um einen Endknoten vom Typ String, Boolean oder Integer) oder ebenfalls Listen von Knoten oder Referenzen auf solche sind.[48]

Da vor allem die DSSSL expression language auf Grund ihrer Komplexität nicht einfach zu implementieren ist, teilt DSSSL ein wenig das Schicksal von SGML.[49]

6.2 XSL - die eXtensible Style Sheet Language

XML als Möglichkeit der semantischen Auszeichnung eines Dokuments war Gegenstand

[46] TEXWEB S. 314
[47] TEXWEB S. 314
[48] TEXWEB S. 314 f.
[49] Eine rühmliche Ausnahme bildet Jade *(http://www.jclark.com/jade)*, die frei verfügbare DSSSL-Engine von James Clark *(http://www.jclark.com)*.

der bisherigen Ausführungen. Da das Ziel dieser Arbeit Single-Source-Publishing ist, wird es nun Zeit, etwas über die Möglichkeiten zu erfahren, die es im Umfeld von XML hierzu gibt.

Die eXtensible Style Sheet Language ist ähnlich wie DSSSL unterteilt: in eine Sprache zur Transformation (XSLT)[50] und eine Sprache zur Formattierung, die XSL Formatting Objects (XSL-FO)[51], beide als XML-Anwendung ausgeführt[52]. Während erstere schon seit einiger Zeit (November 2000) vom W3C als Recommendation verabschiedet und von mehreren Programmen unterstützt wird, sind die Formatting Objects Teil einer noch recht jungen Spezifikation, die erst nach und nach in Softwareprodukten umgesetzt wird. XSL ähnelt im Aufbau sowohl DSSSL als auch den CSS. Im Laufe der Entwicklung des Standards hat sich XSL, speziell XLSFO, mehr und mehr in Richtung der CSS entwickelt, was den Umstieg nicht allzu schwierig macht, wenn man sich an die Syntax von XML-Dokumenten im Allgemeinen gewöhnt hat.[53]

6.2.1 XSLT(ransform)

Es gibt viele Gründe, die es nötig machen, ein XML-Dokument zu transformieren. Es kann nötig sein, die Daten, die nach einer DTD strukturiert sind, in eine andere Struktur (nach Vorbild einer anderen DTD) zu bringen, einem Übersetzungsprozess gleich. Ein besonderer Anwendungsfall dieser Art ist die Ausgabe nach HTML, dazu später mehr. Auch ist es mit einer XSL-Transformation möglich, dynamischen Inhalt zu erzeugen, wie zum Beispiel XML-Dokumente, die sich nach Vorgaben des Benutzers neu ordnen oder sortieren lassen. Oder die Selektion einiger bestimmter Daten anhand vorgegebener Kriterien und die Ausgabe in ein weiteres Dokument. Diese Möglichkeiten machen XSLT zu einer viel benutzten Komponente im Rahmen von XML-basierten E-Commerce. Für diese Arbeit ist allerdings eine andere Möglichkeit entscheidend: die Überführung in eine Rendering- und Präsentationssprache.

Eine XSL Transformation findet nach folgendem Muster statt: ein XSLT Prozessor nimmt als Eingabe jeweils eine XML-Datei und ein XSLT Style Sheet. Aufgrund der Anweisungen in dem XSLT Style Sheet gibt der Prozessor entweder ein anderes XML-Dokument oder Teile davon aus. Dabei steht vor allem der Austausch und die Manipulation von Daten im Vordergrund, nicht die Anzeige (das "Rendern") für den Benutzer.

Ein wichtiger Grundsatz zum Verständnis von XSLT: **XSLT engines manipulieren nicht Dokumente, sondern Strukturen.** [54]

Die interne Struktur, das interne Modell einer XSLT engine ist der Baum, analog zum DSSSL grove.[55]XSLT operiert auf den XML-Dateibäumen und transformiert einen XML

[50] XSLTREC
[51] XSLREC Abschnitt "Formatting Objects"
[52] BIBLE17 Einleitung
[53] XMLPRO S. 373
[54] XMLPRO S. 374
[55] XSLTPR S. 50 ff.

Baum in einen anderen, daher nennt man die Transformationsteil von XSL auch *tree construction part*.[56] Dazu besitzt die XSL Transformation language Operatoren um Knoten im Baum auszuwählen, die Knoten neu anzuordnen und Knoten auszugeben. Falls ein Knoten ein Elementkonten ist, ist er ein Baum für sich.

Die Eingabe für einen XSLT Prozessor muss zwingend ein XML-Dokument sein, eine Überführung von anderen Standards ist weder möglich, noch angedacht. Ein Grenzfall sind HTML- und SGML-Dokumente, da beide Standards recht eng mit XML verwandt sind. XSLT kann auf diesen Dokumenttypen arbeiten, so lange sie wohlgeformt sind. Dennoch gilt: XSLT ist keine Allzweckwaffe mit regulären Ausdrücken zur Transformation von Rohdaten, sondern ein speziell an XML angepasstes Werkzeug.[57]

In der Regel ist die Ausgabe einer XSLT Transformation ebenfalls ein XML-Dokument. Allerdings kann es auch vorkommen, dass als Ausgabe nur ein Teilstück des Ursprungsbaumes gewünscht ist, das als externe Entität in einem anderen XML-Dokument genutzt werden soll. Dass bedeutet, dass die Ausgabe nicht zwingend ein wohlgeformtes XML-Dokument, wohl aber ein Teil eines wohlgeformten XML-Dokumentes ist.[58]

6.2.1.1 Verwendung von XSLT Style Sheets

Um XSLT-Style Sheets einzusetzen, gibt es drei Möglichkeiten: auf Client-Seite, auf Server-Seite oder separat durch ein externes Programm. Auf Client-Seite verschickt ein Webserver eine XML-Datei und ein XSLT Style Sheet. Der Webbrowser enthält einen XSLT Prozessor und ist in der Lage, aus den beiden Dateien eine Transformation durchzuführen und das Ergebnis dem Benutzer zu präsentieren. Serverseitig findet die Umwandlung schon auf dem Server statt: der Server transformiert die XML-Ausgangsdatei und schickt dem Browser eine HTML-Seite. Bei der separat durchgeführten Transformation wird die Umwandlung noch vor der Speicherung des Dokuments auf den Server vollzogen. Dieser Weg kommt vor allem für Benutzer in Betracht, deren Absicht es gar nicht ist, die Dokumente ins Internet zu stellen. Zwei weitere Gründe spielen ebenfalls eine Rolle, warum zumindest zur Zeit der dritte Weg (noch) die beste Methode ist: zum einen ist es viel weniger aufwendig mit einem XSLT Prozessor wie SAXON *(http://sourceforge.net/projects/saxon)* von Michael Kay *(http://users.iclway.co.uk/mhkay)*, XT *(http://www.jclark.com/xt)* von James Clark *(http://www.jclark.com)* oder XALAN *(http://xml.apache.org/xalan-j)* vom XML Apache Project *(http://xml.apache.org)* die Transformation durchzuführen, ohne dazu erst einen Webserver einzurichten. Zum anderen sind bisher nur wenige Webbrowser überhaupt in der Lage, die erste angesprochene Möglichkeit zu nutzen. Auch wenn Microsoft mit dem Internet Explorer 5.5 und 6.0 einen XSLT Prozessor mitbringt und eine neuere Version (MSXML4) zum freien Download *(http://msdn.microsoft.com/xml)* anbietet, bleiben die genannten Stan-

[56] BIBLE17 Abschnitt "Overview of XSL Transformation", "Trees"
[57] BIBLE17 Abschnitt "Overview of XSL Transformation", "Trees"
[58] BIBLE17 Abschnitt "Overview of XSL Transformation", "Trees"

dalone-Programme doch die erste Wahl, da sie den W3C Standard umfangreicher unterstützen und einfacher in der Handhabung und Konfiguration sind. Windows-Usern sei für den Einstieg eine besondere Version von SAXON sehr empfohlen: Instant SAXON *(http://sourceforge.net/projects/saxon)*. Dabei handelt es sich um eine kompilierte Windows Anwendung für die Kommandozeile. Da sowohl SAXON als auch XALAN auf Java aufsetzen, ist die Konfiguration bei beiden Programmen zumindest für unerfahrenere Anwender nicht ganz einfach. Instant SAXON wird einfach in ein Verzeichnis kopiert, das im `PATH` des Windows-Systems liegt und ist sofort einsatzbereit. Eine Transformation wird durch einfach Eingabe von `saxon -o out.html in.xml style.xsl` durchgeführt, wobei der Parameter `-o` lediglich dafür sorgt, dass die Ausgabedatei in diesem Fall `out.html` heißt und nicht den Namen der XML-Datei bekommt. Da sämtliche Transformationen im Rahmen dieser Arbeit mit SAXON durchgeführt wurden, wird an dieser Stelle auf Angaben zur Installation und Verwendung von XALAN verzichtet und auf die ausreichend dokumentierten Webseiten des XML Apache Projects und die zahlreichen Hilfsangebote in Newsgroups wie `comp.text.xml` verwiesen.

6.2.1.2 Aufbau eines XSLT Style Sheets

Zu aller erst ist ein XSLT Style Sheet eines: ein wohlgeformtes XML-Dokument.[59] Es enthält Templates, nach denen eine Transformation zu erfolgen hat. Eine solche Template Rule besteht aus einem Vergleichsmuster (Pattern), das den entsprechenden Knoten selektiert, auf den eine Verarbeitung stattfinden soll, und dem eigentlichen Template an Anweisungen, die ausgeführt und ausgegeben werden sollen, sobald das Vergleichsmuster zutrifft. Bei der Transformation durchläuft der XSLT Prozessor den XML Dokumentbaum und durchsucht jeden Knoten. Bei jedem gelesenen Knoten findet ein Vergleich mit allen Pattern statt, die im Style Sheet aufgeführt sind. Bei einem positiven Vergleich wird die Anweisungen im Template ausgeführt. Ein solches Template kann Markup, neu hinzugefügte Daten oder von anderer Stelle aus dem XML-Dokument kopierte Daten enthalten.[60]

Eingangs wurde schon darauf hingewiesen, dass ein XSLT Style Sheet ein wohlgeformtes XML-Dokument sein muss. Als Wurzelelement steht das `Style Sheet` Element. Zwingend vorgeschrieben ist die Angabe des Namespace `http://www.w3.org/1999/XSL/Transform`. Per Konvention sollte das Präfix `xsl:` gewählt werden (`xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`), der Standard erlaubt aber auch andere selbst gewählte Präfixe. Obwohl XSLT Prozessoren wie SAXON schon recht weit fortgeschritten sind, ist es angeraten, die Konventionen zu befolgen.

Jede Template Rule ist ein Element vom Typ `xsl:template`. Das zu suchende Pattern steht im Attribut `match`. Die Anweisungen, sind der Elementinhalt von `xsl:template`. Innerhalb dieses Templates stehen wiederum XSLT-Elemente, die alle das Präfix `xsl:` im Na-

[59] XMLPRO S. 373
[60] BIBLE17 Abschnitt "Overview", "XSLT style sheet documents"

men tragen und die Aufgaben übernehmen. Alle anderen Elemente (ohne das `xsl:` Präfix) werden als Teil des Ausgabe Baums übernommen. Das kann man sich zu Nutze machen, um zum Beispiel HTML-Elemente in den Ausgabe Baum mit zu übernehmen, die keine Entsprechung im XML-Baum haben. Außerdem wird es dadurch sehr leicht, XSLT Befehle (in Form von XSLT-Elementen) von reinem Text, der unbearbeitet in das Ziel wandert, zu unterscheiden.

In den folgenden Abschnitten werden einige Konstrukte von XSLT näher erläutert. Aus Platzgründen kann an dieser Stelle keine umfassende Darstellung der Materie erbracht werden. Das ist auch nicht Anspruch dieser Arbeit. Eher soll ein generelles Verständnis für die hier geschilderten Techniken vermittelt werden, dass bei der weiteren Einarbeitung mit Hilfe anderer Quellen von Nutzen ist.

6.2.1.2.1 XSL Templates

Wie schon oben angeführt, sind die Template Rules eines XSLT Style Sheets in `xsl:template` Elementen untergebracht. Diese Teile des Style Sheets sind die wichtigsten, da sie spezielle Eingaben mit speziellen Ausgaben verbinden. Die Eingaben in Form von zu verarbeitenden Knoten finden sie anhand des Patterns im `match`-Attribut.

Ein Beispiel:

```
<xsl:template match="/">
<html>
<head>
</head>
<body>
</body>
</html>
</xsl:template>
```

Das Pattern, auf das hier gematcht wird, ist das Wurzelelement "`/`". Der XSLT Prozessor würde in diesem Fall nichts anderes tun, als eine leere HTML-Datei auszugeben, da keinerlei XSLT-Befehle im Template stehen, die den Prozessor anweisen, tiefer in den Baum hinab zu steigen und weitere Anweisungen auszuführen.

6.2.1.2.2 Das `xsl:apply-templates` Element

Um auch andere Knoten außer dem Wurzelknoten erreichen und bearbeiten zu können, müssen die Knoten des Baums rekursiv durchlaufen werden - das ist die Aufgabe von `xsl:apply-templates`. Durch Aufruf des Elements `xsl:apply-templates` wird jeder Knoten als separater Baum behandelt.

Ein Beispiel:

```
<xsl:template match="/">
<html>
```

```
<head>
</head>
<xsl:apply-templates/>
</html>
</xsl:template>
<xsl:template match="title_page">
<body>
</body>
</xsl:template>
```

Zuerst wird das `root` Element bearbeitet und es werden die HTML-Tags `html`, `head` und `/head` ausgegeben. Anschließend wird mit Hilfe von `xsl:apply-templates` das Template `xsl:template match="title_page"` aufgerufen und abgearbeitet, indem `body` und `/body` in den Ausgabe Baum eingefügt werden. Zuletzt wird das schließende `/html` eingefügt.[61]

6.2.1.2.3 Das Attribut `select`

Um nur bestimmte Knoten eines Baumes auszuwählen, dient das `select` Attribut des `xsl:apply-templates` Elements. Auch dazu ein Beispiel:

```
<xsl:template match="title_page">
<body>
<xsl:apply-templates select="title"/>
</body>
</xsl:template>
```

Ist kein `select` Attribut angegeben, werden alle Kind-Elemente, Text, Kommentare und Processing Instruction Knoten ausgewählt.[62]

6.2.1.2.4 Verarbeitung mit Hilfe von `xsl:value-of`

Mit Hilfe des Elements `xsl:value-of` ist es möglich, den Inhalt eines Elements, also den Wert eines Knotens, in den Ausgabe Baum zu überführen.[63]

```
<xsl:template match="title_page">
<xsl:value-of select="title">
</xsl:template>
```

Das Element, dessen Wert ausgewählt wird, ist relativ zum gegenwärtigen Knoten, der vom jeweiligen Template ausgewählt wurde. Der Wert eines Knotens ist immer ein String, eine Zeichenkette (auch die leere Zeichenkette). Es kommt allerdings auf den Typ des Knotens an: in den meisten Fällen ist Knoten vom Typ Element. Der Wert eines solchen Knotens ist dann die Konkatenation der gesamten Zeichen (character data) - abgesehen vom Markup -

[61] Abschnitt "XSL Templates", "The xsl:apply-templates Element"
[62] XMLPRO S. 386
[63] XSLTPR S. 305

zwischen Start- und Endtag des Elements. Die anderen möglichen Typen eines Knotens und die entsprechenden Werte, die `xsl:value-of` zurück gibt, sind:[64]

- root - das Wurzelelement eines Dokuments
- Element - die Konkatenation aller "parsed character data", inklusive aller character data in irgendeinem Unterelement des Elements
- Text - der Text eines Knotens, der Knoten selbst
- Attribut - der Attributwert nach Auflösung aller vorhandener Entitäten, ohne den Namen des Attributs, das Ist-Gleich-Zeichen oder Anführungszeichen
- Namespace - die URI des Namespace
- Processing Instruction - die Daten innerhalb der Processing Instruction, ohne die Zeichen `<?` oder `?>`
- Kommentar - der Text zwischen den Zeichen `<!--` und `-->`

6.2.1.2.5 Verarbeitung mehrerer Elemente mit `xsl:for-each`

Oft kommt es vor, dass Elemente fakultativ sind, dass heißt, ihre Anzahl ist nicht eindeutig bestimmbar. Mit `xsl:value-of` würde allerdings nur das erste Element von mehreren ausgewählt werden. Neben der Verarbeitung mehrfacher Elemente durch Definition eines eigenen Templates gibt es noch das Element `xsl:for-each`, das kein eigenes Template erfordert und dennoch alle Elemente abarbeitet, die durch das `select` Attribut ausgewählt werden.[65]

```
<xsl:for-each select="authors/author">
<xsl:value-of select="."/>
</xsl:for-each>
```

6.2.1.2.6 Vergleichsmuster zur Elementauswahl

XSL unterstützt XPath Ausdrücke. XPath ist ein im November 1999 verabschiedeter W3C Standard *(http://www.w3.org/TR/1999/REC-xpath-19991116)* zur genauen Bestimmung von Teilen in einem Dokumentbaum. Mit Hilfe von XPath ist es nicht nur möglich einzelne Knoten eines Baumes zu selektieren (`match="body"`), sondern auch Tochterknoten (`match="body/chapter"`). Darüber hinaus bietet XPath Möglichkeiten, Elemente anhand ihrer `id` (`match="id(section1)"`) oder ihrer Attribute (`match="@designator"`) auszuwählen und unterstützt Wild Cards, also Platzhalter (`match="apps/*"`).

Das Attribut `select` unterstützt noch sehr viel weitere XPath Ausdrücke und Funktionen.[66] Eine recht bekannte und auch in dieser Arbeit benutzte Funktion ist `position()`, die eine Zahl als Rückgabewert hat, und benutzt wird, um unter anderem abzufragen, ob ein Element das letzte in einem Knoten des Dokumentbaumes ist.

6.2.1.2.7 Steuerung der Ausgabe

Bei der Überführung von XML nach HTML ist es mitunter wichtig, dass den Elementen

[64] BIBLE17 Abschnitt "Computing the Value of a Node with xsl:value-of"
[65] XSLTPR S. 201 ff.
[66] XMLPRO S. 326 ff.

der Ausgabe Attribute und ihre Werte übergeben werden. Dazu kann das Element `xsl:attribute` genutzt werden[67]. Ein Beispiel:

```
<xsl:template match="link"
<a>
<xsl:attribute name="href">
<xsl:value-of select="xlink:href"/>
</xsl:attribute>
</a>
```

Hier erhält das HTML-Element `a` das Attribut `href` mit dem Wert des Attributs `xlink:href` des XML-Elements `link`. Aus dem XLink konformen Link ist somit ein HTML-Link geworden.

Aber nicht nur für die Ausgabe von Attributen gibt es XSL-Elemente. Mit Hilfe von `xsl:text` kann Text an das Ausgabedokument übergeben werden. Dabei beachtet `xsl:text` Leerzeichen. In dieser Arbeit wird es daher unter anderem dazu benutzt, gezielt Leerzeichen auszugeben. Darüber hinaus ist es mit `xsl:text` möglich, Sonderzeichen in die Ausgabe zu portieren.[68]

6.2.1.2.8 Zählen von Knoten mit `xsl:number`

Ein unverzichtbares Element von XSL bei der Erstellung des Inhaltsverzeichnisses dieser Arbeit ist `xsl:number`. Es erzeugt eine Zahl vom Typ Integer Ganzzahl). Diese Zahl ist die Position des Knotens in Bezug zu Knoten gleichen Typs. Durch die Attribute `level`, `count` und `format` lässt sich die Ausgabe noch weiter spezifizieren. Normalerweise zählt `xsl:number` nur die Geschwisterknoten gleichen Typs. Wird das Attribut `level` auf den Wert `any` gesetzt, werden alle Knoten im Baum vom gleichen Typ gezählt. Das Attribut `count` kann dazu benutzt werden nicht nur Knoten des gleichen Typs wie der momentane Knoten zu zählen.[69] Ein Beispiel aus dem Inhaltsverzeichnis dieser Arbeit:

```
<xsl:for-each select="subchapter">
<xsl:number count="chapter"/>.<xsl:number count="subchapter"/>
<xsl:text> </xsl:text>
<xsl:value-of select="title"/>
</xsl:for-each>
```

Hier erhält jedes Element vom Typ `subchapter` vor seinem Titel die Anzahl der Knoten vom Typ `chapter` und die Anzahl der Knoten vom Typ `subchapter`, getrennt durch einen Punkt.

Das Attribut `format` kann den Ausgabewert der von `xsl:number` gelieferten Zahl anpassen. Mögliche Werte sind:[70]

[67] XSLTPR S. 155 ff.
[68] BIBLE17 Abschnitt "Deciding What Output to Include", "Generating text with xsl:text"
[69] XSLTPR S. 237 ff.

- `i` bzw. `I` für die römischen Zahlen `i`, `ii`, `iii` bzw. `I`, `II`, `III`
- `a` bzw. `A` für die Buchstabenreihe `a`, `b`, `c` bzw. `A`, `B`, `C`

6.2.1.2.9 Sortieren der Ausgabe

Bei den Angaben im Literaturverzeichnis ist es ratsam, eine gewisse Sortierung einzuhalten, normalerweise nach Alphabet. Da eine solche Sortierung manuell bei einer gewissen Anzahl an Einträgen etwas Arbeit erfordert, kann man `xsl:sort` die Arbeit überlassen. `xsl:sort` kann nach einem `xsl:apply-templates` oder `xsl:for-each` stehen und sortiert die Ausgabe nach dem Wert des `select` Attributs.[71]

```
<xsl:template match="bibliography">
<p>
<xsl:for-each select="biblio_item">
<xsl:sort select="@designator"/>
...
```

Im obigen Beispiel sortiert `xsl:sort` die Literaturangaben `biblio_item` anhand des Wertes ihres Attributs `designator`.

6.2.1.2.10 Treffen einer Auswahl mit `xsl:if` und `xsl:choose`

In den meisten prozeduralen Programmiersprachen finden sich `if` und `if/elseif` Konstrukte für bedingte Anweisungen. Diese finden ihre Entspechungen in den Elementen `xsl:if` und `xsl:choose`.[72] `xsl:if` entscheidet mit Hilfe des Attributs `test`, ob der Inhalt zwischen dem öffnenden und schließenden `xsl:if` Tag ausgegeben wird. Ist die Bedingung erfüllt, wird der Inhalt ausgegeben, sonst nicht. `xsl:if` in Verbindung mit der Funktion `position()` ergibt den weiter oben angesprochenen Test, ob ein Element das letzte im Teilbaum ist. Da es weder `xsl:else` noch `xsl:elsif` gibt, muss für erweiterte Abfragen `xsl:choose` benutzt werden.

`xsl:choose` legt den Rahmen für eine Reihe von möglichen Vorbedingungen. Jede dieser Vorbedingungen wird durch `xsl:when` eingeleitet, die letzte kann auch durch `xsl:otherwise` eingeleitet werden. Das Element `xsl:when` ist analog zu `xsl:if` aufgebaut und verlangt nach einem Attribut `test`, `xsl:otherwise` dagegen enthält keine weiteren Attribute.

Ein Beispiel ist die Transformation des Elements `link`:

```
<xsl:template match="link">
<xsl:choose>
<xsl:when test="@range='extern'">
<a class="extern">
```

[70] XSLTPR S. 242
[71] BIBLE17 Abschnitt "Sorting Output Elements"
[72] XMLPRO S. 400

```
<xsl:attribute name="href">
<xsl:value-of select="@xlink:href"/>
</xsl:attribute>
<xsl:attribute name="title">
<xsl:value-of select="@xlink:title"/>
</xsl:attribute>
<xsl:value-of select="."/>
</a>
</xsl:when>
<xsl:otherwise>
<a class="intern">
<xsl:attribute name="href">
<xsl:value-of select="@xlink:href"/>
</xsl:attribute>
<xsl:attribute name="title">
<xsl:value-of select="@xlink:title"/>
</xsl:attribute>
<xsl:value-of select="."/>
</a>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
```

6.2.2 XSL-F(ormatting) O(bjects)

Die XSL Formatting Objects (XSL-FO) sind Teil der eXtensible Style Sheet Language XSL. Das Hauptaugenmerk auf dieser XML-Anwendung liegt bei der Präsentation für den Leser[73]. Dazu werden XML-annotierte Daten mit Hilfe eines XSLT-Style Sheets in ein neues Format gebracht. Die Formatting Objects sind entgegen bisherigen Layout-Ansätzen wie Cascading Style Sheets bei HTML seitenorientiert, das heißt als Ausgangsmedium wurde bewusst eine Printausgabe angestrebt. Auch wenn die Elemente von XSL-FO in Gebrauch und Namen eine (beabsichtigte) Übereinstimmung zu den CSS-Gestaltungselementen haben, ist mit ihrer Hilfe ein viel weiter gehenderes und detaillierteres Layouten möglich.

Allerdings kann an dieser Stelle auch ein (momentaner) Nachteil von XSL-FO nicht verschwiegen werden: Momentan gibt es nur wenig Software, die die Möglichkeiten von XSL-FO annähernd unterstützt. Als Glücksfall hat sich das Engagement des XML Apache Projects erwiesen, das mit FOP den ersten XSL-FO Prozessor frei zur Verfügung stellt.

In den folgenden Abschnitten sollen die wichtigsten Elemente von XSL-FO vorgestellt werden. Ähnlich wie der Abschnitt zu XSLT stellt auch dieser Teil der Arbeit keinesfalls eine vollständige Aufstellung aller Möglichkeiten dar, die XSL-FO bietet. Angestrebt ist vielmehr,

[73] BIBLE18 Einleitung

dass der Leser einen Überblick über die Materie bekommt und in der Lage ist, die Ausführungen im Kapitel zum Style Sheet zur Überführung nach XSL-FO nachvollziehen zu können. Sowohl Herold[74] als auch Knobloch und Kopp[75] haben ausführlichere Darstellungen zum Thema, aus denen auch die hier gemachten Ausführungen stammen.

6.2.2.1 Seitenlayout

XSL-FO merkt man sowohl die Verwandtschaft zu DSSSL als auch zu den CSS an. Der Standard, der Teil der am 15. Oktober 2001 verabschiedeten XSL W3C Candidate Recommendation *(http://www.w3.org/TR/xsl)* ist, umfasst über 50 einzelne Formatting Objects mit rund 200 Attributen (hier properties genannt), die vielfach in Element- und Attributnamen an die Bezeichnungen der CSS angelehnt sind.[76] Sowohl an DSSSL als auch an CSS orientiert sich dagegen die Aufteilung der Elemente in Block und Inline, analog zu den Display areas und inline areas, die in DSSSL zur Verfügung stehen.

Vereinfacht ausgedrückt ist XSL-FO ein komplettes XML-Vokabular, um ein XML-Dokument auf eine Druckseite zu bringen.[77]

Das Wurzelelement eines XSL-FO Dokuments ist `fo:root` - wohlgemerkt, es geht hier um das Dokument, das nach der Anwendung eines XSLT Style Sheets aus einem XML-Dokument hervorgeht, nicht um das Style Sheet. Bei einem Style Sheet zur Umwandlung nach XSL-FO ist darauf zu achten, dass sowohl der Namensraum für XSLT (`xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`) als auch der für XSL-FO (`xmlns:fo="http://www.w3.org/1999/XSL/Format"`) angegeben werden. Das `fo:root` Wurzelelement ist vergleichbar mit dem `html` Element, das als Wurzelelement einer HTML-Dokuments ausgegeben wird.

`fo:root` umschließt ein `fo:layout-master-set` und mindestens ein `fo:page-sequence` Element. Im `fo:layout-master-set` werden Templates für die auszugebenden Seiten definiert. Hier kann also festgelegt werden, welche Maße die Seiten haben sollen. Im `fo:page-sequence` Element wird der Inhalt ausgegeben. Ein einfaches XSL-FO Dokument sieht demnach so aus:

```
<xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
<fo:layout-master-set>
...
</fo:layout-master-set>
<fo:page-sequence>
...
```

[74] BIBLE18
[75] WEBXML S. 183 ff.
[76] WEBXML S. 183
[77] BIBLE18 Abschnitt "Formatting Objects and Their Properties", "Transforming to formatting objects"

```
</fo:page-sequence>
</fo:root>
```

Ein solches Dokument wäre zweifelsohne sehr dünn. Also sehen wir uns an, wie man Vorlagen für Seiten erzeugt und sie mit Informationen füllt.

6.2.2.1.1 Definition von Seiten mit `fo:layout-master-set`

Innerhalb des `fo:layout-master-set` Elements werden mit Hilfe eines oder mehrerer `fo:simple-page-master` Elemente Seitendefinitionen erstellt.[78] Die Definition für eine typische A4-Seite sieht wie folgt aus:

```
<fo:layout-master-set>
<fo:simple-page-master master-name="a4"
page-height="297mm"
page-width="210mm"
margin-bottom="20mm"
margin-left="30mm"
margin-right="25mm"
margin-top="15mm">
<fo:region-body/>
</fo:simple-page-master>
</fo:layout-master-set>
```

Die Seite, die hier definiert wird, erhält als Namen (Attribut `master-name`) "a4". Ihre Seitenhöhe beträgt 297 Millimeter und ihre Breite 210 Millimeter. Am Fuß der Seite befindet sich ein Rand von 20 Millimetern (`margin-bottom`), links sind 30 Millimeter und rechts 25 Millimeter Rand vorhanden, während es am Kopf der Seite nur 15 Millimeter sind. Neben der Angabe in Millimeter sind auch Angaben in Zentimetern oder Inches möglich. Innerhalb des `fo:simple-page-masters` werden einzelne Seitenregionen durch die Elemente `fo:region-before`, `fo:region-after`, `fo:region-start`, `fo:region-end` und `fo:region-body` festgelegt. Wie das obige Beispiel zeigt, müssen nicht alle Regionen definiert sein, es reicht, wenn es einen `fo:region-body` gibt, in den anschließend der Inhalt ausgegeben wird.

`fo:region-before` bezeichnet die Kopfzeilenregion, `fo:region-after` die Fußzeile. `fo:region-start` entspricht der Seite, auf der angefangen wird zu schreiben, `fo:region-end` dem Gegenüber. In europäischen Sprachen wäre `fo:region-start` der linke Seitenrand, in arabischen Regionen der rechte, da hier von rechts nach links geschrieben wird. Die Verwendung der allgemeinen Bezeichnungen `start` und `end` hat den Vorteil, dass XSL-FO Dokumente international eingesetzt werden können, ungeachtet der Schreibkonvention.[79]

[78] WEBXML S. 186 ff.
[79] BIBLE18 Abschnitt "Page Layout", "Simple page masters"

Nach der Definition eines `fo:layout-master-set` Elements folgt mindestens ein `fo:page-sequence` Element. Jede `fo:page-sequence` verweist mit dem Attribut `master-name` auf den `fo:simple-page-master`, der das Layout der Sequenz festlegt. Der Inhalt innerhalb des `fo:page-sequence` Elements besteht aus:

- einem optionalen `fo:title` Element, das dem HTML `title` entspricht,
- einer beliebigen Anzahl an `fo:static-content` Elementen und
- einem `fo:flow` Element.

Der Unterschied zwischen `fo:static-content` und `fo:flow` besteht darin, dass der Inhalt von `fo:static-content` auf allen Seiten platziert wird.[80] Ein Beispiel dafür ist der Titel der Arbeit, der auf allen Seiten der Druckversion mit Ausnahme der Titelseite erscheint, ein weiteres die Angabe der Seitenzahlen in der Fußzeile.

Beide Elemente, `fo:flow` und `fo:static-content`, enthalten als Unterelemente Block Elemente. Beispiele für solche Block Elemente sind `fo:block`, `fo:list-block` oder `fo:table`. Daneben gibt es noch eine Reihe weiterer Block Elemente, die aber im Rahmen dieser Arbeit keine Rolle spielen.[81]

6.2.2.2 Block Elemente

Das Element `fo:flow` enthält den eigentlichen Inhalt. Innerhalb eines `fo:flow` Elements befinden sich Block Elemente (block-level formatting objects). Block Elemente ziehen jeweils einen eigenen rechteckigen Bereich, der von anderen durch einen Zeilenumbruch und evtl. weiteren Abstand abgetrennt ist.[82] So bestimmen sie maßgeblich das Layout einer Seite. Ein Beispiel:

```
<fo:page-sequence mster-name="a4">
<fo:flow flow-name="xsl:region-body">
<fo:block>
Ein Block
</fo:block>
</fo:flow>
</fo:page-sequence>
```

Zwingend ist die Angabe `flow-name`, die festlegt, in welcher der möglichen Regionen der Inhalt des `fo:flow` Elements platziert werden soll. Das gilt auch für das Element `fo:static-content`:

```
<fo:page-sequence master-name="a4">
<fo:static-content flow-name="xsl:region-before">
<fo:block>
Ein Block als statischer Inhalt
```

[80] BIBLE18 Abschnitt "Page Layout", "Page sequences"
[81] BIBLE18 Abschnitt "Page Layout", "Flows"
[82] BIBLE18 Abschnitt "Content", "Block-level formatting objects"

```
</fo:block>
</fo:static-content>
<fo:flow flow-name="xsl:region-body">
<fo:block>
Ein Block
</fo:block>
</fo:flow>
</fo:page-sequence>
```

Im obigen Beispiel würde auf allen Seiten dieser `fo:page-sequence` in der Kopfzeile der Satz "Ein Block als statischer Inhalt" erscheinen. Um in einem `fo:static-content` Element die Seitenzahlen anzeigen zu lassen, wird das Element `fo:page-number` benutzt.

Attribute im Element `fo:page-sequence` beeinflussen die Art der Seitennummerierung. In dieser Arbeit werden nur `initial-page-number` zur Angabe der Seitenzahl, mit der die Nummerierung begonnen werden soll, und `format="I"` genutzt, um für die Seitenzahlen des Inhaltsverzeichnisses römische Zahlen zu verwenden. Ohne das `format` Attribut werden arabische Zahlen ausgegeben.

```
<fo:page-sequence master-name="a4" initial-page-number="1" lan-
guage="de">
<fo:static-content flow-name="xsl-region-after">
<fo:block font-size="12pt" text-align="end">
<fo:page-number/>
</fo:block>
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
...
</fo:flow>
</fo:page-sequence>
```

Auch für Listen bietet XSL-FO ein äuqivalentes Element: `fo:list-block`. Ein solches Element besteht aus verschiedenen `fo:list-item` Elementen, die allerdings etwas komplizierter aufgebaut sind. In XSL-FO wird nicht wie in HTML eine Unterscheidung zwischen geordneten und ungeordneten Listen gemacht. Vielmehr enthält jedes `fo:list-item` Element ein `fo:list-item-label`, das aus einer Zahl oder einem beliebigen anderen Zeichen innerhalb eines `fo:block` Elements bestehen kann. Für eine geordnete Liste bietet es sich an, hier `xsl:number` zu bemühen. Nach dem `fo:list-item-label` folgt der `fo:list-item-body`, in dem der Inhalt des Listenpunkts enthalten ist.[83] Ein Beispiel:

```
<fo:list-block>
<fo:list-item>
<fo:list-item-label>
```

[83] BIBLE18 Abschnitt "Lists"

```
<fo:block>*</fo:block>
</fo:list-item-label>
<fo:list-item-body>
<fo:block>Ein Listenpunkt</fo:block>
</fo:list-item-body>
</fo:list-item>
</fo:list-block>
```

Ebenso wichtig wie die Listen sind die Tabellen. Auch wenn in der DTD kein Tabellenelement vorhanden ist, so werden Tabellen in XSL-FO benötigt, um Verzeichnisse wie das Inhaltsverzeichnis sauber formatiert darzustellen. Die Elemente `fo:table` und `fo:table-and-caption` erzeugen beide eine Tabelle, das zweite mit einer Überschrift oder Unterzeile in Form eines `fo:table-caption` Elements. Da in dieser Arbeit die Tabellen nur als Hilfsmittel genutzt werden, beschränkt sich die Darstellung auf `fo:table`.

Die XSL-FO Tabellen ähneln dem HTML Element `table`. Zuerst steht ein Element `fo:table`, gefolgt von einem optionalen `fo:table-header`, einem optionalen `fo:table-footer` und mindestens einem `fo:table-body` Element. Da FOP Tabellen nicht ohne feste Spaltenzuweisung darstellen kann, ist bis auf weiteres für jede Spalte ein ansonsten optionales `fo-table-column` Element mit dem Attribut `column-width` dem `fo:table-body` voranzustellen.[84]

Das `fo:table-body` Element ist wie in HTML durch Zeilen (`fo:table-row`) unterteilt, die jeweils einzelne Zellen (`fo:table-cell`) enthalten. Eine einzeilige Tabelle mit zwei 80 Millimeter breiten Spalten sieht demnach so aus:

```
<fo:table>
<fo:table-column column-width="80mm"/>
<fo:table-column column-width="80mm"/>
<fo:table-body>
<fo:table-row>
<fo:table-cell>eine Tabellenzelle</fo:table-cell>
<fo:table-cell>und noch eine Tabellenzelle</fo:table-cell>
</fo:table-row>
</fo:table-body>
</fo:table>
```

Neben den bisher vorgestellten Block Elementen gibt es noch Inline Elemente und Out-of-line Elemente

6.2.2.3 Inline Elemente

Im Gegensatz zu Block Elementen hat das Element `fo:inline` keinen Einfluss auf das Layout einer Seite. Es ist vergleichbar mit dem Container-Element `span` in HTML. Wie die-

[84] BIBLE18 Abschnitt "Tables"

ses kann auch `fo:inline` Formatierungsattribute wie `font-style` oder `color` enthalten. Einzelne `fo:inline` Elemente werden nicht durch Zeilenumbruch voneinander getrennt. So ist es möglich, innerhalb eines Satzes Wörter oder Buchstaben verschieden zu formatieren:[85]

```
<fo:block>
<fo:inline font-style="italic">
Dieser Text ist kursiv, während
</fo:inline>
<fo:inline font-style="normal" font-weight="bold">
der hier fett formatiert ist.
</fo:inline>
</fo:block>
```

Die Attribute zur Formatierung werden im entsprechenden Teil dieser Arbeit näher erläutert.

Ein weiteres Inline Element ist `fo:leader` zur Darstellung von formatierten Linien, das ebenfalls in dieser Arbeit benutzt wird. HTML kennt das Element `hr` (horizontal rule), das eine horizontale Linie zeichnet. Ähnlich funktioniert `fo:leader`. Allerdings sind außer horizontalen Linien auch gepunktete Verbindungen zwischen einem links- und einem rechtsausgerichtetem Textteil möglich, wie man sie aus dem Inhaltsverzeichnis von Büchern kennt - zumindest im Prinzip.[86] Denn FOP bietet dafür momentan keine Unterstützung an. Daher wird `fo:leader` in dieser Arbeit auch nur wie `hr` benutzt.

```
<fo:static-content flow-name="xsl-region-before">
<fo:block font-size="12pt"
text-align="start"
font-style="italic">
<xsl:value-of select="meta/title"/>
<fo:block>
<fo:leader
leader-pattern="rule"
leader-length="155mm"
rule-thickness="1pt"
color="black"/>
</fo:block>
</fo:block>
</fo:static-content>
```

Im Seitenkopf wird nach dem Titel der Magisterarbeit eine ein Punkt dünne Linie gezogen, um den Kopf vom Rest der Seite zu trennen. Durch die Attribute `leader-pattern`, `leader-length` und `rule-thickness` kann das Aussehen der Linie bestimmt werden.

[85] BIBLE18 Abschnitt "Inlines"
[86] BIBLE18 Abschnitt "Leaders and Rules"

`leader-pattern` kann unter anderem Werte wie `space`, `rule` oder `dots` annehmen. Mit `leader-length` wird die Länge bestimmt. Für ein einheitliches Layout empfiehlt es sich, die Länge an die Breite von `fo:region-body` anzupassen.

Mit dem Attribut `rule-thickness` kann die Dicke der Linie bestimmt werden, Standardwert ist hier ein Punkt. Mit `color` kann der Linie eine Farbe zugewiesen werden.

Auch `fo:external-graphic` zum Einbinden externer Grafiken, die nicht in XML-eigenen Grafikformaten wie SVG oder MathML vorliegen, ist ein Inline Element. FOP unterstützt Grafiken in den Formaten GIF und JPEG, beide Formate werden auch von Internetbrowsern unterstützt. `fo:external-graphic` ist aufgebaut wie das `img` Element in HTML. Wichtigstes Attribut beider Elemente ist `src`, das einen URI mit dem Ort der Grafik enthält.[87]

Mit den Attributen `width` und `height` können die vertikalen und horizontalen Abmessungen festgelegt werden. Standardmäßig skaliert FOP die Grafik im Seitenverhältnis soweit hoch, wie es die Abmessungen des `fo:region-body` Elements es erlauben. Das bedeutet, dass die Attribute mit Werten belegt werden müssen, um eine Grafik in den Originalabmessungen zu erhalten.

```
<fo:external-graphic src="images/screenshot.gif" width="102mm"
height="106mm"/>
```

6.2.2.4 Out-of-line Elemente

Es gibt zwei sogenannte Out-of-line Elemente: `fo:footnote` und `fo:float`. Letzteres wird leider von FOP noch nicht unterstützt, `fo:footnote` dagegen schon. Wie der Name schon sagt, erzeugt `fo:footnote` eine Fußnote. Das Element wird direkt im Fließtext an die Stelle platziert, an der die Fußnotenreferenz später erscheinen soll. `fo:footnote` enthält sowohl die Referenz als auch den eigentlichen Fußnotentext (`fo:footnote-body`), der vom XSL-FO Prozessor an das Ende der Seite gesetzt wird. Die Referenz für eine Fußnote ist genau wie beim `fo:list-item-label` dem Autoren überlassen. Für durchnummerierte Fußnoten ist auch hier wieder `xsl:number` zuständig.[88]

```
<fo:footnote>
<fo:inline font-size="8pt" vertical-align="super">
<xsl:number count="reference" level="any"/>
</fo:inline>
<fo:footnote-body>
<fo:inline font-style="italic" font-size="10pt">
Dazu siehe auch...
</fo:inline>
</fo:footnote-body>
</fo:footnote>
```

[87] BIBLE18 Abschnitt "Graphics"
[88] BIBLE18 Abschnitt "Footnotes"

Hier wird eine automatisch durchnummerierte Fußnote erzeugt (das Beispiel stützt sich auf die dieser Arbeit zu Grunde liegende DTD), die als acht Punkt große hochgestellte (`vertical-align="super"`) Zahl in den Text eingefügt wird. Der Fußnotentext am Ende der Seite wird in zehn Punkt großer Kursivschrift (`font-style="italic"`) dargestellt. In diesem Beispiel fehlt beim Fußnotentext der Bezug zur Fußnote. Dazu müsste ein weiteres Mal `xsl:number` benutzt werden.

6.2.2.5 Attribute (properties)

XSL-FO enthält eine Vielzahl von sogenannten properties, die den Attributen entsprechen. An dieser Stelle kann nur auf einige davon eingegangen werden.

Das Attribut `id` kann wie in HTML an jedes beliebige XSL-FO Element angefügt werden. Es entspricht in seinen Anforderungen allerdings dem XML Attributtyp ID.[89]

Mit Hilfe des Attributs `language` kann die Sprache zum Beispiel eines `fo:block` Elements spezifiziert werden. Als Werte können nur ISO 639 konforme Sprachen-Codes wie "de" für Deutsch oder "en" für Englisch benutzt werden. Damit entspricht `language` dem HTML Attribut `lang`.[90]

In Absätzen können desweiteren noch folgende Typen von Attributen stehen:

- break properties
- hyphenation properties
- indent properties

Die break properties (Umbruch-Attribute) teilen dem XSL-FO Prozessor mit, ob und wann ein Umbruch zu erfolgen hat. Es gibt hierfür folgende Attribute: `keep-with-next`, `keep-with-previous`, `keep-together`, `break-before` und `break-after`. Die beiden erstgenannten Attribute legen fest, wieviel Aufwand der XSL-FO Prozessor treiben soll, um das nachfolgende (`keep-with-next`) oder vorangegangene (`keep-with-previous`) Formatierungsobjekt mit dem aktuellen Element zusammen auf einer Seite unterzubringen. `keep-together` bezieht sich auf den Inhalt des aktuellen Formatierungsobjekts. Als Werte sind ganzzahlige Werte (je höher der Wert, desto höher der Aufwand) oder die Schlüsselworte "always" und "auto" möglich, wobei letzteres die Standardeinstellung ist.

Die Properties `break-before` und `break-after` dagegen erzwingen einen Umbruch. Mögliche Werte sind hier: `column`, für einen Umbruch in einer Textspalte, `page`, um auf die nächste Seite zu wechseln, `even-page`, um die aktuelle Seite umzubrechen und zur nächsten gerade nummerierten Seite zu wechseln, `odd-page`, um zur nächsten ungerade nummerierten Seite zu wechseln und `auto`, wobei hier der XSL-FO Prozessor freie Hand hat.[91]

[89] BIBLE18 Abschnitt "Formatting Properties", "The id property"
[90] BIBLE18 Abschnitt "Formatting Properties", "The language property"
[91] BIBLE18 Abschnitt "Formatting Properties", "Break properties"

Eine ebenfalls wichtige Gruppe von Attributen sind die hyphenation properties, die für die Wörtertrennung zuständig sind. Folgende Attribute sind für einen deutschsprachigen Text notwendig:

```
<fo:block language="de" text-align="justify" hyphenate="true"
hyphenation-character="-">
```

Mit `hyphenate="true"` wird die automatische Silbentrennung aktiviert. Durch das `language` Attribut weiß der XSL-FO Prozessor, welches Modul für Silbentrennung anzuwenden ist, `hyphenation-character` setzt den Bindestrich als Trennungszeichen fest. FOP bringt eine Vielzahl an Modulen zur Silbentrennung mit, darunter auch Deutsch.[92]

Die indent properties erlauben es, Textzeilen oder -blöcke einzurücken. Das Attribut `start-indent` legt eine Einrückung aller Zeilen des betreffenden Elements vom Textanfang fest. Eine Einrückung zum Textende hin wird mit `end-indent` erreicht. `text-indent` erlaubt es, nur die erste Zeile einzurücken, wie es auch bei der Druckversion dieser Arbeit der Fall ist, die letzte Zeile wird mit `last-line-end-indent` eingerückt.[93]

```
<fo:block text-indent="5mm">
die erste Zeile dieses Absatzes ist um fünf Millimeter nach
rechts eingerückt.
</fo:block>
```

Einige weitere Formatierungsattribute, die nicht nur in Block Elementen auftreten können, sollen an dieser Stelle noch kurz dargestellt werden. Um Text zu formatieren, bietet XSL-FO alle Eigenschaften an, die auch in den CSS enthalten sind. Die folgende Aufzählung berücksichtigt aus Platzgründen nur die in dieser Arbeit genutzten Attribute: `color`, `font-family`, `font-stlye`, `font-weight`, `line-height`, `text-align`, `text-align-last`, `background-color`. Bis auf `text-align-last` entsprechen sie alle ihren CSS-Namensvettern. Die Verwendung wird bei den Erläuterungen zum Style Sheet für die Transformation nach XSL-FO deutlich. `text-align-last` bestimmt die Ausrichtung der letzten Zeile eines Absatzes. So ist es sinnvoll, bei Blocksatz die letzte Zeile linksbündig auszurichten um zu stark gesperrten Satz zu vermeiden.

[92] BIBLE18 Abschnitt "Formatting Properties", "Hyphenation properties"
[93] BIBLE18 Abschnitt "Formatting Properties", "Indent properties"

7 XSL in der Anwendung

Nach den theoretischen Ausführungen der beiden Teilaspekte von XSL, XSLT und XSL-FO, soll nun anhand der Magisterarbeit der praktische Einsatz gezeigt werden. Dazu werden zwei XSLT Style Sheets entwickelt: eines dient der Transformation nach HTML, das andere der nach XSL-FO.

7.1 Überführung nach HTML

Ein wichtiger Zweck, zu dem XSL eingesetzt wird, ist die Überführung nach HTML. Eine solche Überführung bietet sich an, ist es doch - zumindest wenn man XHTML als Zielformat wählt - eine Überführung von einer XML DTD in eine andere. Aber auch wenn das Ziel HTML in der aktuellen Version 4.01 nach SGML-Standard ist, stellt eine Transformation keine großen Probleme dar, da die Unterschiede zwischen den beiden Spachversionen relativ gering sind. XHTML lässt wie XML keine offenen Tags zu, während HTML einige Elemente kennt, die nicht geschlossen werden müssen, wie zum Beispiel `br` für einen Zeilenumbruch.[94] Viele XSLT Prozessoren unterstützen die Überführung durch den Parameter `xsl:output method="html"`, der eine HTML-konforme Ausgabe erlaubt. Eine solche Erleichterung ist zwar nicht Teil des Standards, es kann jedoch davon ausgegangen werden, dass sie noch eine Weile in den Prozessoren zu finden ist - zumindest bis XHTML HTML ablöst.

7.1.1 Vorüberlegungen

Bevor es an die Formatierung geht, sollten erste Überlegungen aufgestellt werden. Schon beim praktischen Arbeiten mit XMetaL muss man sich früher oder später Gedanken darüber machen, welche Elemente wie dargestellt werden sollen. Werden CSS zur Darstellung in der *Normal View* benutzt, ist damit eine Entscheidung verbunden, ob ein Element als Block oder Inline dargestellt werden soll. Das heißt, soll das Element absatzbildend sein oder darf ein nachfolgendes Element direkt (oder durch hinzugefügte Zeichen getrennt) im Anschluss stehen. Alle diese Überlegungen sind hilfreich, wenn es an die Gestaltung des Style Sheets geht, dass ein XML-Dokument nach HTML überführen soll. Natürlich ist das Arbeiten mit CSS zur Ansicht im Browser oder in der *Normal View* in XMetaL nur ein erster Schritt, aber dennoch hilfreich, um eine erste, grobe Einteilung vorzunehmen.

Desweiteren hat es sich als nützlich erwiesen, eine Aufstellung zu erstellen, in der die XML-Elemente den entsprechenden HTML-Elementen gegenüber gestellt werden. Wenn man sich klar darüber ist, was in HTML und in CSS möglich ist, dann ist es recht einfach, sich die Darstellung der eigenen Arbeit vorzustellen.

Schon bei der Erstellung der DTD hat es einige Vorüberlegungen gegen, in wie weit es Sinn macht, bestimmte Elemente auszuzeichnen und andere nicht. Ein Beispiel dafür ist das

[94] XHTMLWD Abschnitt "Differences with HTML 4"

Element `abbrevation`. Es ist geplant, eine Eigenschaft des HTML Elements `a` zu nutzen, um bei Abkürzungen einen Mehrwert für den Benutzer zu erzeugen. Mit Hilfe von `a` werden in HTML Hyperlinks hergestellt. `a` kann dabei Ausgangspunkt (mit dem Attribut `href`) und Ziel (mit dem Attribut `name`) eines Links sein. Interessant für das Element `abbrevation` ist aber vor allem das Attribut `title`, mit dem bei einem Link ein Text angezeigt werden kann, der das Linkziel beschreibt. Die meisten Browser zeigen diese Information als kleines Fenster, einen sogenannten Tooltip, über dem Inhalt des `a` Elements an (wie in der Abbildung sichtbar), sobald der Mauszeiger über dem entsprechenden Element verweilt.

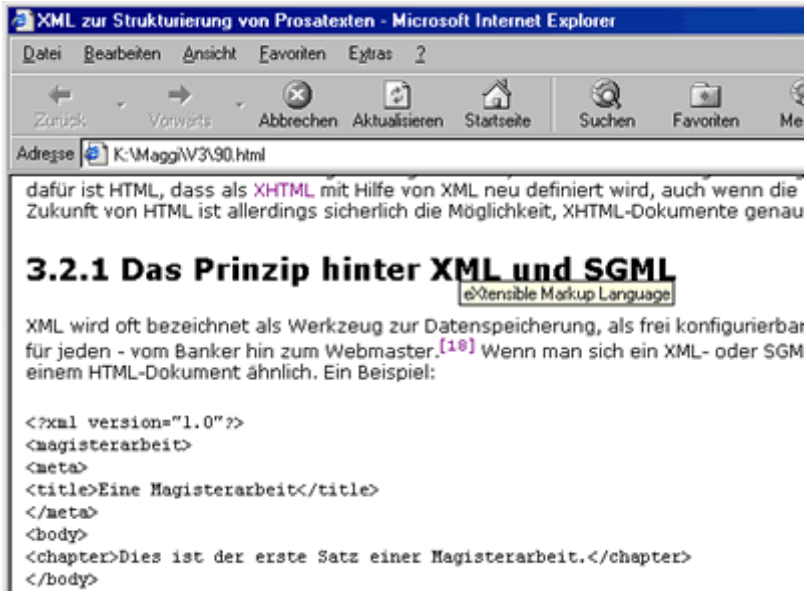

Abbildung 6: *Verhalten bei einer Abkürzung*

Um dieses Verhalten zu erreichen, reicht es also, alle gekennzeichneten Abkürzungen in ein `a title=""` Element zu überführen. Der Wert des Attributs `title` ist der Wert des Elements `longform`, das über das Attribut `long_form` anhand seiner `id` referenziert wird.

Eine weitere Überlegung betrifft die Fußnoten (Element `reference`). Da die Darstellung in HTML nicht seitenorientiert ist, werden Fußnoten üblicherweise in Endnoten umgewandelt.

Nach diesen beiden Beispielen sollen nun einzelne Templates des Style Sheets, das sich in vollständiger Länge im Anhang wieder findet, ausführlicher dargestellt werden.

7.1.2 Das XSLT Style Sheet für die Transformation nach HTML

Im Prolog der Datei steht die XML-Deklaration. Bei der Encodierung wurde Unicode gewählt, um auch Umlaute nutzen zu können. Daran schließt sich das `xsl:stylesheet` Element an, das zwingend mit der Angabe des Namensraumes (namespace) `xmlns:"http://www.w3.org/1999/XSL/Transform"` in einem XSL Style Sheet stehen muss. Zusätzlich ist die Angabe des namespaces von XLink erforderlich, da das `link` Ele-

ment auf XLink basiert.

```
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xlink="http://www.w3.org/1999/xlink" >
```

Es folgt das Element `xsl:output`, das einige wichtige Parameter für die Ausgabe enthält. Durch `method="html"` und die Angabe eines System Identifiers und eines Public Identifiers für die HTML 4.01 Transitional DTD wird die HTML-Version genau festgelegt. Die Attribute `encoding` und `indent` veranlassen den XSLT Prozessor, einen bestimmten Zeichensatz zu benutzen und die Ausgabe einzurücken.

```
<xsl:output method="html"
doctype-system="http://www.w3.org/TR/html4/loose.dtd"
doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"
encoding="ISO-8859-1"
indent="yes"/>
```

Soweit zu den allgemeinen Angaben. Das erste Template, das das Wurzel-Element bearbeitet, muss dafür sorgen, dass das HTML-Wurzel-Element ausgegeben wird und anschließend die anderen Templates aufrufen.

```
<xsl:template match="/">
<html>
<xsl:apply-templates/>
</html>
</xsl:template>
```

Damit steht schon der Rumpf der HTML-Datei. Noch ist aber nichts enthalten. Da ein HTML-Dokument aus `head` und `body` besteht, ist es ratsam, sich zuerst dem Kopf zu widmen. Dazu dienen die Angaben aus dem `meta` Element. Damit werden zuerst die Meta-Angabe `author` und das Element `title`, das für die Ausgabe des Titels im Browserfenster verantwortlich ist, mit Informationen gefüllt. Ebenso wird ein externe CSS-Datei eingebunden, die später für eine ansprechendere Formatierung sorgt. Durch Aufruf einer `xsl:for-each` Schleife wird sichergestellt, dass wirklich jeder Autor, der an der Arbeit beteiligt war, zu seinem Recht kommt. Ein zusätzlicher Test per `xsl:if` fügt an jeden Autoren außer dem letzten ein Komma und ein Freizeichen an. Das hat zur Folge, dass bei einem einzigen Autoren nur der Name ausgegeben wird, mehrere Autoren werden durch Kommata voneinander getrennt.

```
<xsl:template match="meta">
<head>
<meta name="author">
<xsl:attribute name="content">
<xsl:for-each select="authors/author">
```

```
<xsl:value-of select="name/first_name"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/surname"/>
<xsl:if test="position()!=last()">
<xsl:text>, </xsl:text>
</xsl:if>
</xsl:for-each>
</xsl:attribute>
</meta>
```

```
<link rel="stylesheet" type="text/css" href="mag.css"/>
<title>
<xsl:value-of select="title"/>
</title>
</head>
</xsl:template>
```

Der Kopf der HTML-Datei ist damit komplett. Mehr Arbeit macht der eigentliche Inhalt im `body` Element. Hier muss natürlich auch zuerst das öffnende `body` Tag ausgegeben werden. Anschließend wird mit Hilfe eines `div` Elements für eine zentrierte Ausgabe des Titels, Untertitels, der Fakultät, Universität, Gutachter und des Autor der Arbeit, mit den Angaben zu seiner Adresse und Matrikelnumer gesorgt. Dabei werden die einzelnen Angaben mit Überschriften verschiedenener Ordnung ausgegeben, die dann per CSS entsprechend formatiert werden. Die Überschrift erster Ordnung, `h1`, wird nur für den Titel der Arbeit benutzt. Die Ausgabe der Adresse wird dabei einem weiteren Template überlassen, da Angaben zur Adresse auch in den Literaturangaben auftauchen.

```
<xsl:template match="body">
<body>
<div align="center">
<h1><xsl:value-of select="//meta/title"/></h1>
<h2><xsl:value-of select="//meta/subtitle"/></h2>
<h3>Magisterarbeit der <xsl:value-of select="//meta/faculty"/>
der <xsl:value-of select="//meta/university"/></h3>
<h4><xsl:value-of select="//meta/department"/></h4>
<h5>Gutachter:<br/>
<xsl:for-each select="//meta/supervisors/supervisor">
<xsl:value-of select="@grade"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/first_name"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/surname"/>
<xsl:if test="position()!=last()"> <br/> </xsl:if>
```

```
</xsl:for-each>
</h5>
<h5>vorgelegt von: </h5>
<xsl:for-each select="//meta/authors/author">
<xsl:value-of select="name/first_name"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/surname"/>
<br/>
<xsl:if test="address">
<xsl:apply-templates select="address"/> <br/> </xsl:if>
<xsl:if test="matrikel_nr">
Matrikel-Nr.: <xsl:value-of select="matrikel_nr"/> <p/>
</xsl:if>
<xsl:if test="position()!=last()"> <br/> </xsl:if>
</xsl:for-each>
</div>
```

Nachdem diese Angaben ausgegeben wurden, geht es nun daran, ein Inhaltsverzeichnis für die Arbeit zu erstellen. An dieser Stelle kann der Vorgang nur ausschnittsweise wiedergegeben werden, da es sich hier um den längsten Teil des XSL Style Sheets handelt. Die vollständige Version ist im Anhang und auf der CD, die dieser Arbeit beiliegt, zu finden. Daher hier nur einige Angaben zur Vorgehensweise.

Zur besseren Formatierung in HTML wird zuerst eine Tabelle erstellt. In diese wird nun für jedes Element `chapter` mit Hilfe einer `xsl:for-each` Schleife eine Tabellenzeile `tr` eingefügt. In diese Zeile wird mit Hilfe von `xsl:number count="chapter"` die Anzahl der Kapitel in eine Tabellenzelle `td` eingetragen. Das erste Kapitel erhält die Nummer "1", das zweite Kapitel die "2" usw. Daran schliesst sich der Titel des Kapitels. Damit ist ein erstes rudimentäres Inhaltsverzeichnis für alle Elemente vom Typ `chapter` erstellt worden. Da jedes dieser Elemente weitere Unterelemente (`subchapter`) enthalten kann, die wiederum Unterelemente beinhalten können, müssen vor Schließen des Elements `xsl:for-each` weitere Schleifen eingebunden werden. Exemplarisch ist das hier für die Elemente `chapter` und `subchapter` aufgeführt. Durch Einsatz von `span` können die einzelnen Hierarchie-Ebenen mit Hilfe eines Cascading Style Sheets separat formatiert werden.

Bei der Nummerierung der Unterkapitel ist zu beachten, dass die Nummern der oberen Hierarchie-Elemente mitgeführt werden (durch "." abgetrennt). Das Ergebnis ist dann ein Inhaltsverzeichnis, wie man es von jeder Textverarbeitung her kennt: Unter Kapitel 1 folgt Unterkapitel 1.1, dann Unterkapitel 1.2 und irgendwann Kapitel 2.

```
<table border="0" cellpadding="2" cellspacing="2">
<xsl:for-each select="chapter">
```

```
<tr><td>
<span class="level1">
<xsl:number count="chapter"/>
<xsl:text> </xsl:text>
<xsl:value-of select="title"/>
</span>
</td></tr>
<xsl:for-each select="subchapter">
<tr><td>
<span class="level2">
<xsl:number count="chapter"/>.<xsl:number count="subchapter"/>
<xsl:text> </xsl:text>
<xsl:value-of select="title"/>
</span>
</td></tr>
</xsl:for-each>
</xsl:for-each>
</table>
```

Noch bietet dieses Inhaltsverzeichnis keine weiteren Möglichkeiten bietet, als die einzelnen Titel der Verzeichnisse anzuzeigen. Selbst ein gedrucktes Inhaltsverzeichnis zeigt immerhin zusätzlich die Seitenzahl an, auf der der entsprechende Abschnitt der Arbeit zu finden ist. Nun besitzt HTML allerdings keinerlei Seiten, als Ausgleich aber Hyperlink-Funktionalität. Was liegt also näher, als die einzelnen Kapitelüberschriften im Inhaltsverzeichnis mit den entsprechenden Überschriften im Fließtext zu verlinken? Da sämtliche strukturierenden Elemente mit einer eindeutigen `id` ausgezeichnet sind, ist es kein Problem, diese als Linkziel zu verwerten. Eine modifizierte Version des Inhaltsverzeichnisses sieht dann so aus:

```
<table border="0" cellpadding="2" cellspacing="2">
<xsl:for-each select="chapter">
<tr><td>
<span class="level1">
<xsl:number count="chapter"/>
</span>
<xsl:text> </xsl:text>
<a class="level1">
<xsl:attribute name="href">
<xsl:text>#</xsl:text>
<xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:value-of select="title"/>
</a>
```

```
</td></tr>
<xsl:for-each select="subchapter">
<tr><td>
<span class="level2">
<xsl:number count="chapter"/>.<xsl:number count="subchapter"/>
</span>
<xsl:text> </xsl:text>
<a class="level2">
<xsl:attribute name="href">
<xsl:text>#</xsl:text> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
</xsl:for-each>
</xsl:for-each>
</table>
...
```

Das a Element hat nun ebenfalls als Attribut eine CSS-Klasse, damit Zahl und Eintrag im Inhaltsverzeichnis gleich dargestellt werden. Das hier gezeigte Inhaltsverzeichnis hat allerdings einen Nachteil. Zum einen werden nur die Kapitel, Unterkapitel, Abschnitte usw. erfasst, die im `body` der Magisterarbeit liegen. Weder die Kapitel, die im Anhang auftauchen können (`apps` enthält einzelne `chapter`), noch Abkürzungs- oder Literaturverzeichnis tauchen im Inhaltsverzeichnis auf. Das ist aber durchaus gewollt: Kapitel im Anhang werden üblicherweise nicht mit Zahlen nummeriert, sondern mit Buchstaben, gefolgt von Zahlen. Indize wie Abkürzungs- oder Literaturverzeichnis erhalten keine Nummerierung. Dem wird insofern Rechnung getragen, dass das schließende `/table` Tag erst erfolgt, nachdem die einzelnen genannten Elemente abgearbeitet worden sind. Besonders viel Aufwand erfordert das Element `apps`, da es hier theoretisch ebenso viele Hierarchie-Ebenen geben kann wie im `body`. Der Aufwand lohnt sich aber, da dadurch ein vollständiges, Standard konformes Inhaltsverzeichnis erstellt wird.

Die Nummerierung mit Buchstaben an Stelle von Zahlen lässt sich durch ein zusätzliches Attribut `format="A"` im Element `xsl:number` erreichen.

```
<xsl:for-each select="//appendix/*/chapter">
<tr><td>
<span class="level1">
<xsl:number count="chapter" format="A"/>
</span>
<xsl:text> </xsl:text>
```

```
<a class="level1">
<xsl:attribute name="href">
<xsl:text>#</xsl:text>
<xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
```

Damit nach dem Inhaltsverzeichnis auch weitere Templates ausgeführt werden, gehört in das oben definierte `body` Template als Anweisung natürlich nach dem Schließen der Inhaltsverzeichnis-Tabelle ein `xsl:apply-templates/` und ein schließendes HTML `/body` Tag. Vor dem Schließen des HTML `body` Elements bietet es sich an, noch die zu Endnoten gewandelten Inhalte des Elements `reftext` einzubinden. Das wird durch `xsl:apply-templates select="//*/reftext"/` sichergestellt. Aus ästhetischen Gründen bietet es sich an, eine Linie zwischen Fließtext und Endnoten zu ziehen. Der Ende des `body` Templates sieht also wie folgt aus:

```
...
</table>
<xsl:apply-templates/>
<hr/>
<xsl:apply-templates select="//*/reftext"/>
</body>
</xsl:template>
```

Ein Inhaltsverzeichnis - und sei es noch so funktional - nützt natürlich wenig, wenn kein Inhalt vorhanden ist. Zuerst werden also die Templates definiert, um die einzelnen strukturellen Elemente auszugeben. Den Anfang macht `chapter` als größtes Teilelement. Der Titel wird als Überschrift zweiter Ordnung ausgegeben, die Überschrift erster Ordnung bleibt dem Titel der Magisterarbeit vorbehalten. Damit die Links des Inhaltsverzeichnisses ein Ziel haben, ist es nötig, dass die Überschriften als Linkziel definiert werden. Dazu gibt es in HTML zwei Möglichkeiten: das `a` Element mit dem Attribut `name` oder das Attribut `id`, dass an andere HTML-Elemente angefügt werden kann. Als Wert bietet sich der Wert des Attributs `id` an, das sämtliche Strukturelemente haben. Dadurch vererbt man die `id`. Vor dem Titel sollte natürlich auch die Kapitelnummer stehen, was wieder mit `xsl:number` passiert. Anschließend wird der einleitende Absatz ausgegeben. Das geschieht in einem weiteren Template.

```
<xsl:template match="chapter">
<h2>
<xsl:attribute name="id">
<xsl:value-of select="@id"/>
</xsl:attribute>
```

```
<xsl:number count="chapter"/>
<xsl:text> </xsl:text>
<xsl:apply-templates select="title"/>
</h2>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subchapter"/>
</xsl:template>
```

Die hier gezeigte Methode wird für sämtliche Struktur tragenden Elemente bis zur `subsubsection` durchgeführt. Und anschließend noch einmal für das Element `apps`, das ja ebenfalls `chapter` enthalten kann. Das ist notwendig, um mit Hilfe des Attributs `format="A"` im `xsl:number` Element die korrekte Nummerierung zu erreichen. Theoretisch ist das auch erreichbar, in dem bei jedem dieser Elemente per XPath-Funktion `ancestor` kontrolliert wird, ob das Vater-, Großvater- usw. Element vom Typ `body` oder `apps` ist. Darauf wurde hier allerdings verzichtet.

Das Ausschöpfen sämtlicher in HTML zur Verfügung stehender Hierarchie-Ebenen macht die Verwendung eines externen CSS zwingend erforderlich, da eine Überschrift sechster Ordnung in HTML normalerweise kleiner als der anschließende Fließtext dargestellt wird.

Der Anhang `appendix` gestaltet sich etwas anders als der `body` der Magisterarbeit. Um beide Elemente sichtbar von einander zu trennen, bietet es sich auch hier an, mit Hilfe des Elements `hr` eine Linie zu ziehen.

Im Anhang werden dann die Templates für die allgemeinen Anhänge `apps` und das Abkürzungsverzeichnis `abbrevations` aufgerufen.

```
<xsl:template match="appendix">
<hr/><xsl:apply-templates select="apps"/>
<hr/><xsl:apply-templates select="abbrevations"/>
</xsl:template>
```

Da die größeren Teile der Magisterarbeit schon hinreichend "erschlagen" sind, wird es Zeit sich dem wichtigsten Element aus inhaltlicher Sicht zu zuwenden: dem Absatz `paragraph`. Da HTML ebenfalls Absätze in Form des Elements `p` kennt, bietet es sich an diese zu nutzen. Innerhalb eines solchen Absatzes muss dann nur dafür gesorgt werden, dass die darin auftauchenden Elemente (alle, die in der Entität `%text;` erfasst sind) ebenfalls bearbeitet werden, was wiederum mit `xsl:apply-templates/` geschieht.

```
<xsl:template match="paragraph">
<p><xsl:apply-templates/></p>
</xsl:template>
```

Damit kommen wir schon zu Elementen, die im Absatz vorkommen können. Den Anfang macht `link`.

Theoretisch lässt sich `link` 1:1 in einen HTML Link umwandeln. Allerdings stört das Attribut `range`, das es in HTML nicht gibt, da HTML keinerlei Unterscheidung zwischen internen und externen Links macht. Das ist allerdings mit Hilfe von CSS durch eine andere Darstellung im Text möglich. Damit wird dem Benutzer schnell deutlich, ob ein Link den Aufbau einer Internetverbindung erfordert oder nicht.

```
<xsl:template match="link">
<xsl:choose>
<xsl:when test="@range='extern'">
<a class="extern">
<xsl:attribute name="href">
<xsl:value-of select="@xlink:href"/>
</xsl:attribute>
<xsl:attribute name="title">
<xsl:value-of select="@xlink:title"/>
</xsl:attribute>
<xsl:value-of select="."/>
</a>
</xsl:when>
<xsl:otherwise>
<a class="intern">
<xsl:attribute name="href">
<xsl:value-of select="@xlink:href"/>
</xsl:attribute>
<xsl:attribute name="title">
<xsl:value-of select="@xlink:title"/>
</xsl:attribute>
<xsl:value-of select="."/>
</a>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
```

Auch beim Element `asset` zur Einbindung von Multimedia-Elementen ist eine Unterscheidung nötig, je nachdem ob es sich um eine Grafik, ein Video oder eine Audio-Datei handelt.

Die Einbindung von Grafiken in HTML durch das `img` Element ist recht einfach. Für die Einbindung von Audio- und Video-Daten gibt es verschiedene Wege. Sowohl das W3C als auch die Browserhersteller haben lange Zeit in dieser Hinsicht ihr eigenes Süppchen gekocht. Der Weg, der hier gewählt wurde, ist die Einbindung durch das `object` Element, wie es das W3C empfiehlt *(http://www.w3.org/TR/html4/struct/objects.html#h-13.3)* . Dessen `data` At-

tribut ist analog zum `src` Attribut des `img` Tags.

```
<xsl:template match="asset">
<xsl:choose>
<xsl:when test="@type='image'">
<img>
<xsl:attribute name="src">
<xsl:value-of select="@source"/>
</xsl:attribute>
<xsl:attribute name="width">
<xsl:value-of select="@width"/>
</xsl:attribute>
<xsl:attribute name="height">
<xsl:value-of select="@height"/>
</xsl:attribute>
<xsl:attribute name="alt">
<xsl:value-of select="@alt"/>
</xsl:attribute>
<xsl:value-of select="."/>
</img>
</xsl:when>
<xsl:otherwise>
<object>
<xsl:attribute name="data">
<xsl:value-of select="@source"/>
</xsl:attribute>
<xsl:attribute name="alt">
<xsl:value-of select="@alt"/>
</xsl:attribute>
<xsl:value-of select="."/>
</object>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
```

Auf die Elemente `emph`, `strong` und `quotation` soll an dieser Stelle nicht weiter eingegangen werden, da sie alle Entsprechungen in HTML haben (`em`, `strong` und `cite`). Für die Formatierung von `code` gibt es ebenfalls ein HTML-Äquivalent `code`, das dafür sorgt, dass der Elementinhalt mit einer äquidistanten Schrift gesetzt wird.

Interessanter ist die Realisierung des Elements `abbrevation`. Wie schon angesprochen sollen Abkürzungen mit einem Tooltip versehen werden, in dem die ausgeschriebene Form angezeigt wird. Ein solcher Tooltip wird durch Setzen des Attributs `title` im `a` Element er-

reicht, der den Wert des Elements erhält, auf das mit der Referenz `long_form` verwiesen wird. Das Element wird mit Hilfe der Funktion `id()` ermittelt, die einen Knoten im Dokumentbaum anhand seiner `id` auswählt. Den Inhalt des Knotens erhält man durch einen einfachen XPath-Ausdruck.

```
<xsl:template match="abbrevation">
<a class="abb">
<xsl:attribute name="title">
<xsl:value-of select="id(@long_form)/."/>
</xsl:attribute>
<xsl:apply-templates/>
</a>
</xsl:template>
```

Das Element `codeline`, das Quelltextzeilen enthält, kann auf zwei Arten nach HTML überführt werden. Die einfachere ist, einfach das HTML-Element `code` zu benutzen und ein `br/` für einen Zeilenumbruch anzuhängen. Allerdings lässt sich Quelltext in geballter Form leichter lesen, wenn er deutlich vom Rest des Textes abgesetzt wird. Das kann durch Einrücken oder durch Hinterlegen mit einer Hintergrundfarbe geschehen. Beides ist in HTML mit Hilfe von CSS möglich, indem das `div` Element verwendet wird, das auch gleich dafür sorgt, dass ein Zeilenumbruch nach einem `codeline` Element erfolgt.

```
<xsl:template match="codeline">
<div class="code"><xsl:apply-templates/></div>
</xsl:template>
```

Die Verwendung von Endnoten macht eine Zweiteilung der Fußnotenverwaltung nötig: das Element `reference` sorgt dafür, dass am Ort der Referenz ein Verweis auf die Endnote erfolgt, `reftext` dagegen enthält den Inhalt. Den Aufruf des Templates, das `reftext` verarbeitet, haben wir schon am Ende des `body` Templates getätigt. Er sorgt dafür, dass die Endnoten am Ende des Fließtextes des `body` der Arbeit stehen, aber noch vor dem Anhang.

Um die Fußnoten HTML-konform aufzubereiten, bietet es sich an, sie als Link zu realisieren. Dazu wird beim Element `reference` ein Link der Form `"[ ]"` eingefügt, der als Ziel und Inhalt die Anzahl der Referenzen hat. Somit erhält man eine automatisch generierte Fußnote, die sich entsprechend anpasst, wenn eine weitere Referenz in das Dokument eingebunden wird (und die Ausgabedatei neu erzeugt wird). Formatiert wird die Fußnote als `sup`, also hochgestellt. Eine entsprechende CSS-Klasse `"footnote"` sorgt dafür, dass die generierte Zahl eine geringere Textgröße als der sie umgebende Fließtext besitzt.

Darüber hinaus ist es natürlich angenehm, wenn man vom Endnotentext wieder zurück in den Fließtext springen kann, möglichst da, wo die Referenzstelle zu finden ist. Das wird dadurch erreicht, das der Link an der Stelle der Referenz nicht nur den Endnotentext als Linkziel

hat, sondern ebenfalls Linkziel ist. Dazu wird wieder die Anzahl der Fußnoten verwendet, allerdings mit einem vorangestellten "r".

```
<xsl:template match="reference">
<sup><a class="footnote">
<xsl:attribute name="name">
<xsl:text>r</xsl:text>
<xsl:number count="reference" level="any"/>
</xsl:attribute>
<xsl:attribute name="href">
<xsl:text>#</xsl:text>
<xsl:number count="reference" level="any"/>
</xsl:attribute>
<xsl:attribute name="title">
<xsl:text>Zur Literaturstelle</xsl:text>
</xsl:attribute>
<xsl:text>[</xsl:text
> <xsl:number count="reference" level="any"/>
<xsl:text>]</xsl:text>
</a></sup>
</xsl:template>
```

Beim `reftext` bleibt nur noch dafür zu sorgen, dass er zum einen Ziel der Fußnote wird, was ebenfalls wieder einfach über die Anzahl der Referenzen im Text geschieht. Zum anderen muss er natürlich die Referenz als Linkziel haben, damit ein Sprung zurück möglich ist. Für diesen Sprung zurück wird nicht der gesamte Endnotentext verwendet, sondern dem Text wird ein Zeichen in Form seiner Unicode-Nummer vorangestellt, das einen nach oben gerichteten Pfeil darstellt.

Während der Arbeit am Endnotentext kam zusätzlich noch die Idee auf, dass es auch nützlich wäre, mehr über die Literaturstelle zu erfahren, auf die verwiesen wird. Da bei Referenzen in dieser Arbeit nur jeweils der Bezeichner, der jedem Literatureintrag vorangestellt ist, genannt wird, ergab sich die Möglichkeit, diesen Bezeichner ebenfalls als Link zu nutzen, der zum entsprechenden Eintrag im Literaturverzeichnis führt. Das wird erreicht, indem man den Wert des Attributs `refered_to` benutzt, das ja auf den Eintrag im Literaturverzeichnis verweist.

```
<xsl:template match="reftext">
<span class="footnotetext">
<a class="footback">
<xsl:attribute name="name">
<xsl:number count="reftext" level="any"/>
</xsl:attribute>
```

```
<xsl:attribute name="href">
<xsl:text>#r</xsl:text><xsl:number count="reftext" level="any"/>
</xsl:attribute>
<xsl:attribute name="title">
<xsl:text>zurück zum Text</xsl:text>
</xsl:attribute>
<xsl:text>&#8593;</xsl:text>
</a>
<xsl:text> </xsl:text>
<a>
<xsl:attribute name="href">
<xsl:text>#</xsl:text>
<xsl:value-of select="../@refered_to"/>
</xsl:attribute>
<xsl:attribute name="title">
<xsl:text>ausführlichere Informationen zur Literatur</xsl:text>
</xsl:attribute>
<xsl:value-of select="../@refered_to"/>
</a>
<xsl:text> </xsl:text>
<xsl:apply-templates/>
</span><br/>
</xsl:template>
```

Auf die Darstellung der Transformation der Elemente `list`, `item`, `date` und `apps` wird aus Platzgründen verzichtet, die jeweiligen Templates finden sich im Anhang. Das gleiche gilt für das Abkürzungsverzeichnis `abbrevations` und das Literaturverzeichnis `bibliography`, die beide nicht automatisch generiert werden.

Bei letzterem gibt es allerdings zwei Besonderheiten, die eine Erwähnung verdienen. Die eine ist die - durch das Element `reftext` bedingte - und schon angesprochene Auszeichnung der einzelnen Bezeichner als Linkziel und die vorherige Sortierung der Literatureinträge anhand desselben mit Hilfe von `xsl:sort`.

Die zweite ist der Versuch, in HTML eine Art XLink zu kreieren. In XLink ist es möglich, mehrere Ziele für einen Link anzugeben. HTML unterstützt dies nicht. Für den Benuzter wäre es aber angenehm, wenn er nach Verfolgen einer Endnote bis ins Literaturverzeichnis wieder zurück zum Text gelangen könnte. Da auf eine Quelle mehrmals verwiesen werden kann, ist der Ursprung nicht feststellbar. Daher wird für den Bezeichner eines Literatureintrags nicht nur ein Linkziel, sondern auch ein Link mit dem Ziel `javascript:history.go(-2)` eingetragen. Dieser JavaScript-Befehl veranlasst den Browser zwei Seiten in der History zurück zu blättern. Falls der Benutzer wirklich von einer Endnote aus ins Literaturverzeichnis gelangt

ist, kommt er damit wieder zurück zum Text.

Die Templates zu den Elementen `address` und `authors` sind ähnlich aufgebaut. Beide verlangen nach einer Abfrage, ob fakultative Elemente vorhanden sind. Darüber hinaus sind aber beide recht einfach aufgebaut und werden daher auch nicht ausführlicher an dieser Stelle gewürdigt.

Einzig `comment` ist noch eine Erwähnung wert. Da die Anmerkungen, die bei der Erstellung der Arbeiten gemacht wurden, nicht in den fertigen Text mit einfließen sollen, ist es nötig, dass im Template für `comment` eine Ausgabe unterbleibt, was durch Definition eines leeren Templates passiert.

```
<xsl:template match="comment">
</xsl:template>
```

Damit wäre eine der beiden Aufgaben aus dem Pflichtenheft für Single-Source-Publishing erfüllt. Das im Rahmen dieser Arbeit entwickelte XSLT Style Sheet ist sicherlich nicht die einzige Möglichkeit, Instanzen der dieser DTD nach HTML zu überführen. Es ist auch möglich, dass trotz intensiven Testens noch Fehler darin enthalten sind. Einige Kritikpunkte wurden schon in diesem Abschnitt angesprochen. Zum Beispiel ist der Weg zur Generierung des Inhaltsverzeichnisses noch recht ineffizient. Das gleiche gilt für die Elemente `chapter`, `subchapter`, `section`, `subsection` und `subsubsection`, für die jeweils separate Templates erstellt wurden, abhängig davon, ob sie im `body` oder im Anhang der Arbeit stehen.

Sicherlich gibt es auch noch genügend Ansatzpunkte, um die Überführung besser zu modularisieren. So wäre ein eigenes Template für das Element `name` denkbar, was die Ausgabe der Titelseit verkürzen würde.

Daher ist dieses Style Sheet nur als eine von vielen Möglichkeiten und als Ausgangspunkt für weitere Überlegungen in diese Richtung zu verstehen. Ähnliches gilt auch für das folgende Style Sheet, das die Überführung nach XSL-FO und die anschließende Druckausgabe realisiert.

7.2 Überführung nach XSL-FO

XSL-FO ist das Ausgangsformat für die Printausgabe der Magisterarbeit. XSL-FO-Dokumente können von Hand erstellt werden, da sie nichts anderes sind als XML-Dokumente, die einer bestimmten DTD folgen - der von XSL-FO. Allerdings ist der Nutzen mit dem vergleichbar, den das Erstellen einer separaten HTML-Datei als Online-Version der Magisterarbeit hätte: beide Dokumente hätten nichts miteinander gemein. Um XSL-FO als Ausgabeformat für ein Single-Source-Publishing zu erzielen, ist wieder der Einsatz eines XSLT Style Sheets erforderlich.

Im Gegensatz zu HTML werden die Formatting Objects allerdings noch von keiner An-

wendung nativ unterstützt. In naher Zukunft wird es für Browser vermutlich selbstverständlich sein, XML-Dokumente auch in Verbindung mit einem XSLT Style Sheet als XSL-FO anzuzeigen, ähnlich wie schon heute eine Verarbeitung von XSLT Style Sheets zur Ausgabe nach HTML auf Client-Seite mit einigen Browsern möglich ist. Auf lange Sicht ist davon auszugehen, dass XSL-FO einen ähnlichen Stellenwert erlangen wird wie CSS für HTML schon heute. Bis es soweit ist, muss man sich allerdings anderer Möglichkeiten bedienen, um XSL-FO zur Druckausgabe eines XML-Dokuments nutzen zu können. Die momentan am weitesten fortgeschrittene Methode ist die Überführung in ein anderes, bekannteres Format. Neben RTF als ein mögliches Zielformat, wie es auch von DSSSL unterstützt wird, hat sich hier PDF als Mittel zum Zweck erwiesen. Mit Hilfe eines XSL-FO Prozessors wie FOP kann aus XSL-FO-Dateien PDF erzeugt werden. Dazu mehr im entsprechenden Teil dieser Arbeit. Zuerst sollen allgemeine Vorüberlegungen die Printausgabe betreffend angestellt werden.

7.2.1 Vorüberlegungen

Bei den Überlegungen zum Printdesign dieser Arbeit spielten mehrere Aspekte eine Rolle. Die einzelnen Elemente der XML-Datei müssen auf XSL-FO Elemente abgebildet werden. Dabei gibt es Elemente, die im Printbereich anders auftauchen müssen als bei der HTML-Ausgabe, das Element `link` zum Beispiel. Bei der HTML-Version sollte ein Element `link` in einen HTML-Link überführt werden, das heißt, das eigentlche Linkziel (die URL) muss für den Benutzer nicht sichtbar sein, sondern nur als Wert des `href`-Attributes des `a`-Tags vorliegen. Es wird mehr Wert darauf gelegt, dass der Inhalt des `a`-Elements und ein eventuell vorhandener `title` angezeigt werden und der Link als solcher sichtbar ist. Bei der Printausgabe gibt es für den Benutzer keinerlei Möglichkeiten, einen im Dokument verborgenen Link zu verfolgen. Auch eine übertriebene Hervorhebung, wie sie üblicherweise im Browser durch Unterstreichen und die Farbe Blau praktiziert wird, ist in einer Papierversion nicht nur unnötig, sie wird auch durchaus als Störung des Redeflusses empfunden. Unerlässlich ist dagegen, dass das Linkziel in irgendeiner Form, sei es als Fußnote oder direkt im Anschluss an den betreffenden Text, genannt wird. So kann der Benutzer durch Eingeben der URL in seinen Browser die Angaben im Text überprüfen oder sich weiterführendes Material ansehen. Als Darstellung von Links im Text wurde ein direkter Anschluss an den betreffenden Ausdruck gewählt. Die URL steht in Klammern hinter dem ursprünglich verlinktem Wort und ist kursiv gesetzt.

7.2.2 Das XSLT Style Sheet für die Überführung nach XSL-FO

Am Anfang steht bei einer Überführung nach XSL-FO natürlich die Definition eines Layout-Master-Sets mit anschließender Erstellung von Layoutvorlagen für den Inhalt. Eine Magisterarbeit sollte mindestens zwei verschiedene `fo:simple-page-master` haben: ein Template für die Titelseite und eines für den restlichen Inhalt. Je nach Design-Vorlieben kann es auch nötig sein, für das Inhaltsverzeichnis oder den Anhang weitere Vorlagen zu kreieren.

Das war in dieser Arbeit allerdings unnötig.

Zuerst wird ein `fo:simple-page-master` für die Titelseite definiert. Dazu wird eine Seite im Format A4 genutzt, bei der neben `fo:region-body` auch die Regionen `fo:region-before` und `fo:region-after` erstellt werden. In der Kopfzeilenregion werden auf der Titelseite die Angaben zu Universität, Fakultät, Fachbereich und Gutachter stehen, im `fo:region-body` Titel, Untertitel, eine zusätzliche Aussage über die Art der Arbeit und die Nennung des oder der Autoren, und im Fußzeilenbereich nähere Angaben zum Autor wie Adresse und Matrikelnummer. Um die Ränder und Maße für die einzelnen Regionen aufeinander abzustimmen, bleibt nichts anderes übrig, als mehrere Probeläufe mit Übersetzung nach PDF und anschließender Kontrolle im Adobe Acrobat Reader *(http://www.adobe.com/products/acrobat/readstep.html)*.

```
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:fo="http://www.w3.org/1999/XSL/Format" >
<xsl:output indent="yes"/>
<xsl:template match="magisterarbeit">
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
<fo:layout-master-set>
<fo:simple-page-master >
master-name="title"
page-height="297mm"
page-width="210mm"
margin-bottom="15mm"
margin-left="20mm"
margin-right="25mm"
margin-top="20mm">
<fo:region-before extent="40mm"/>
<fo:region-body margin-top="40mm" margin-bottom="32mm"/>
<fo:region-after extent="32mm"/>
</fo:simple-page-master>
```

Als zweiter `fo:simple-page-master` wird eine Seite für den Rest des Dokuments erstellt. Die Ränder entsprechen den Standardwerten abgesehen vom etwas größeren Rand auf der linken Seite, um trotz Bindung die Lesbarkeit zu bewahren. Die Kopf- und Fußzeilenregionen sind kleiner als bei der Titelseite, da hier weniger Informationen aufgenommen werden müssen. Allenfalls bei einer großen Anzahl von Fußnoten auf einer Seite kann es passieren, dass der Größe der Region nicht ausreicht.

```
<fo:simple-page-master master-name="a4"
```

```
page-height="297mm"
page-width="210mm"
margin-bottom="7mm"
margin-left="25mm"
margin-right="25mm"
margin-top="15mm">
<fo:region-before extent="15mm"/>
<fo:region-body margin-top="15mm" margin-bottom="28mm"/>
<fo:region-after extent="16mm"/>
</fo:simple-page-master>
</fo:layout-master-set>
```

Nicht unwichtig ist die Diskrepanz zwischen der Größe des unteren Randes des `fo:region-body` (28mm) und die eigentliche Größe der `fo:region-after` (16mm). Da FOP den zur Verfügung gestellten Platz mitunter etwas merkwürdig nutzt, kann es ohne diesen Sicherheitsabstand vorkommen, dass die hochgestellte Fußnotennummer noch in die letzte Zeile des Fließtextes hereinragt. Aber auch hier gilt, dass das optimale Maß nur nach mehreren Versuchen zu finden ist. Diese und andere Probleme werden auch im entspechenden Kapitel dieser Arbeit noch ausführlicher gewürdigt.

```
<fo:page-sequence master-name="title">
<fo:static-content flow-name="xsl-region-before">
<fo:block font-size="12pt"
font-family="serif"
line-height="18pt"
text-align="start">
<xsl:value-of select="//meta/university"/>
</fo:block>
<fo:block font-size="12pt"
font-family="serif"
line-height="18pt"
text-align="start">
<xsl:value-of select="//meta/faculty"/>
</fo:block>
<fo:block font-size="12pt"
font-family="serif"
line-height="18pt"
text-align="start">
<xsl:value-of select="//meta/department"/>
</fo:block>
<fo:block font-size="12pt"
font-family="serif"
```

```
line-height="18pt"
text-align="start">
Gutachter:
<xsl:for-each select="//meta/supervisors/supervisor">
<xsl:value-of select="@grade"/>
<xsl:text>
<xsl:value-of select="name/first_name"/>
<xsl:text>
<xsl:value-of select="name/surname"/>
<xsl:if test="position()!=last()">
<xsl:text>, </xsl:text>
</xsl:if>
</xsl:for-each>
</fo:block>
</fo:static-content>
<fo:static-content flow-name="xsl-region-after">
<fo:block font-size="12pt"
font-family="serif"
line-height="18pt"
text-align="start">
<xsl:for-each select="//meta/authors/author">
<fo:inline>
<xsl:value-of select="name/first_name"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/surname"/>
</fo:inline>
<fo:block font-size="12pt"
font-family="serif"
line-height="18pt"
text-align="start">
<xsl:if test="address">
<xsl:apply-templates select="address"/>
</xsl:if>
</fo:block>
<fo:block font-size="12pt"
font-family="serif"
line-height="18pt"
text-align="start">
<xsl:if test="matrikel_nr">
<xsl:text>Matrikel-Nr.: </xsl:text>
<xsl:value-of select="matrikel_nr"/>
```

```
</xsl:if>
</fo:block>
</xsl:for-each>
</fo:block>
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
<fo:block font-size="24pt"
font-family="serif"
line-height="32pt"
font-weight="bold"
text-align="center"
space-before="50mm">
<xsl:value-of select="//meta/title"/>
</fo:block>
<fo:block font-size="16pt"
font-family="serif"
line-height="24pt"
text-align="center"
space-before="5pt">
<xsl:value-of select="//meta/subtitle"/>
</fo:block>
<fo:block font-size="14pt"
font-family="serif"
line-height="21pt"
text-align="center">
<xsl:text>- Magisterarbeit -</xsl:text>
</fo:block>
<fo:block font-size="12pt"
font-family="serif"
line-height="18pt"
text-align="center"
space-before="6pt">
vorgelegt von:
</fo:block>
<fo:block font-size="12pt"
font-family="serif"
line-height="18pt"
text-align="center">
<xsl:for-each select="//meta/authors/author">
<fo:inline>
<xsl:value-of select="name/first_name"/>
```

```
<xsl:text> </xsl:text>
<xsl:value-of select="name/surname"/>
</fo:inline>
</xsl:for-each>
</fo:block>
</fo:flow>
</fo:page-sequence>
```

Anschließend wird das Inhaltsverzeichnis generiert. Das geschieht ähnlich wie bei der HTML-Version. Auch in XSL-FO wird dazu eine Tabelle benutzt, hier mit drei Spalten. Allerdings werden die Überschriften nicht als Link ausgezeichnet, obwohl sowohl XSL-FO als auch PDF Linking unterstützen. Sinnvoller für die Druckversion ist es, wenn statt eines Links die Seitenzahl des entsprechenden Kapitels im Inhaltsverzeichnis zu finden ist. XSL-FO bietet hierzu das Element `fo:page-number-citation` an, das von FOP - wenn auch noch in experimenteller Weise - unterstützt wird. `fo:page-number-citation` liefert über das Attribut `ref-id` die Seitenzahl des Elements mit der betreffenden `id`, ähnlich wie `\pageref` bei LaTeX.[95] Auch hier kann also von der in allen Struktur tragenden Elementen als Attributwert enthaltenen `id` profitiert werden.

In der Kopfzeile der Seite soll als statischer Inhalt der Titel der Arbeit in kursiver Schrift erscheinen, darunter eine ein Punkt schmale, durchgezogene Linie. In der Fußzeile steht die Seitenzahl. Das wird durch das XSL-FO Element `fo:page-number` erreicht. Damit die Seitenzahlen als römische Zahlen ausgegeben werden, wird als Attribut der `fo:page-sequence format="I"` mitgegeben. Ein `initial-page-number="1"` verhindert, dass die Titelseite mitgezählt wird.

```
<fo:page-sequence master-name="a4" initial-page-number="1" for-
mat="I">
<fo:static-content flow-name="xsl-region-before">
<fo:block font-size="12pt"
text-align="start"
font-style="italic">
<xsl:value-of select="meta/title"/>
<fo:block>
<fo:leader leader-pattern="rule"
leader-length="155mm"
rule-thickness="1pt"
color="black"/>
</fo:block>
</fo:block>
</fo:static-content>
```

[95] LATEX S. 104

```
<fo:static-content flow-name="xsl-region-after">
<fo:block font-size="12pt" text-align="end">
<fo:page-number/>
</fo:block>
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
<fo:block font-size="18pt"
font-family="serif"
line-height="27pt"
break-before="page"
text-align="justify">
Inhaltsverzeichnis
</fo:block>
<fo:table>
<fo:table-column column-width="25mm"/>
<fo:table-column column-width="120mm"/>
<fo:table-column column-width="8mm"/>
<fo:table-body font-family="serif">
<xsl:for-each select="body/chapter">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start"
font-size="14pt"
line-height="21pt"
font-weight="bold">
<xsl:number count="chapter"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start"
font-size="14pt"
line-height="21pt"
font-weight="bold">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end"
font-size="14pt"
line-height="21pt"
font-weight="bold">
```

```
<fo:page-number-citation>
<xsl:attribute name="ref-id">
<xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
...
```

Bei der Ausgabe des Inhalts ist dagegen bei der Nummerierung der Seiten darauf zu achten, dass die Seitenzahlen wieder bei "1" beginnen und in arabischen Zahlen ausgegeben werden. Im `fo:region-body` werden die entsprechenden XSL Templates ausgeführt.

```
<fo:page-sequence master-name="a4" initial-page-number="1" lan-
guage="de">
<fo:static-content flow-name="xsl-region-before">
<fo:block font-size="12pt"
text-align="start"
font-style="italic">
<xsl:value-of select="meta/title"/>
<fo:block>
<fo:leader leader-pattern="rule"
leader-length="155mm"
rule-thickness="1pt"
color="black"/>
</fo:block>
</fo:block>
</fo:static-content>
<fo:static-content flow-name="xsl-region-after">
<fo:block font-size="12pt" text-align="end">
<fo:page-number/>
</fo:block>
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
<xsl:apply-templates select="body"/>
</fo:flow>
</fo:page-sequence>
```

Analog dazu ist auch die Ausgabe von Anhang und Literaturverzeichnis realisiert. Vor Ausgabe des Literaturverzeichnisses wird ein Abbildungsverzeichnis generiert, das es in der HTML-Version nicht gibt.

In der folgenden Darstellung wurden die schon bekannten Teile des Elements `fo:page-sequence` wie Kopfzeile und Seitennummer zur Förderung der Übersichtlichkeit ausgelassen.

Zuerst wird die `id` des Elements `images` auf das XSL-FO Element vererbt und der Titel des Abbildungsverzeichnisses ausgegeben. Anschließend wird in einer dreispaltigen Tabelle für jedes Element vom Typ `asset` eine Tabellenzeile angelegt. In die erste Spalte wird das Wort "Abbildung" gefolgt von der fortlaufenden Nummer der Abbildungen im Text und einem Doppelpunkt ausgegeben. In die nächste Spalte kommt der Inhalt des Attributs `alt`, das eine Bildbeschreibung enthält. In die letzte Spalte kommt schließlich die Seitenzahl, auf der das Bild zu finden ist. Hier kommt wieder `fo:page-number-citation` zum Einsatz.

```
<fo:page-sequence master-name="a4">
<fo:static-content flow-name="xsl-region-before">
...
</fo:static-content>
<fo:static-content flow-name="xsl-region-after">
...
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
<fo:block font-size="20pt"
font-family="serif"
line-height="30pt"
font-weight="bold"
text-align="justify">
<xsl:attribute name="id">
<xsl:value-of select="appendix/images/@id"/>
</xsl:attribute>
<xsl:value-of select="appendix/images/title"/>
</fo:block>
<fo:table>
<fo:table-column column-width="30mm"/>
<fo:table-column column-width="90mm"/>
<fo:table-column column-width="40mm"/>
<fo:table-body font-size="12pt" font-family="serif">
<xsl:for-each select="//*/*/*/*/asset">
<fo:table-row line-height="16pt">
<fo:table-cell>
<fo:block text-align="start">
<xsl:text>Abbildung </xsl:text>
<xsl:number count="asset" level="any"/>
<xsl:text>:</xsl:text>
```

```
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start">
<xsl:value-of select="@alt"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end">
<fo:page-number-citation>
<xsl:attribute name="ref-id">
<xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
</xsl:for-each>
</fo:table-body>
</fo:table>
</fo:flow>
</fo:page-sequence>
```

Die Ausgabe der Elemente `body` und `appendix` der Magisterarbeit ist recht unspektakulär - es werden nur die jeweiligen Templates zu `chapter` bzw. `apps` und `abbrevations` aufgerufen. Daher wird hier auf eine ausführliche Darstellung verzichtet und auf den Anhang verwiesen.

Die Templates zur Transformation der Struktur tragenden Elemente sind alle ähnlich aufgebaut. Zuerst wird ein `fo:block` Element zur Ausgabe der Überschrift ausgegeben. Zu dieser Ausgabe gilt es allerdings abzuwägen. Wie schon in den Vorüberlegungen angeklungen, wird bei Links die URL als Text an den Link angehängt. In Überschriften wäre ein solcher Effekt allerdings höchst unerwünscht, was mit der Ausgabe des Titels durch `xsl:value-of select="title"` unterdrückt werden könnte. Andererseits ist es angenehm, wenn Quellcode-Ausdrücke entsprechend formatiert werden.

Auch die Überlegung, dass Element `title` aus der Entität `%bibtext;` zu entfernen, ist nicht sinnvoll, da bei Internetquellen im Literaturverzeichnis der Verweis auf die URL fehlen würde.

Da in dieser Arbeit mehrmals Elemente vom Typ `code` in Überschriften auftauchen, `link` Elemente aber nicht, wurde entschieden `xsl:apply-templates` zu benutzen. Als allgemein gültig und optimal kann diese Regelung allerdings nicht bezeichnet werden.

Exemplarisch ist hier das Template für das Element `chapter` aufgeführt. Nach der Ausgabe des Titels werden die Templates für Absätze und - in diesem Fall - für das Element `subchapter` angewendet. Damit die Funktionalität des Inhaltsverzeichnisses gewährleistet ist, ist es nötig, den Wert des XML-Attributs `id` als Wert eines neu generierten XSL-FO Attributs `id` zu vererben. Die Überschriften sind für die hierarchischen Elemente in unterschiedlichen Größen formatiert. Mit 20 Punkt Schriftgröße und einem anderthalbfachen Zeilenabstand (`line-height="30pt"`) ist der `title` des Elements `chapter` am größten. Die Schriftgröße des Titels der anderen Elemente ist jeweils um zwei Punkte kleiner als die des in der Hierarchie übergeordneten Elements. Die Zeilenhöhe beträgt immer das 1,5fache davon (wie es bei Magisterarbeiten üblich ist).

```
<xsl:template match="chapter">
<fo:block font-size="20pt" font-family="serif" line-
height="30pt" break-before="page" font-weight="bold" keep-
with-next="always">
<xsl:attribute name="id"><xsl:value-of
select="@id"/></xsl:attribute>
<xsl:number count="chapter"/><xsl:text>
</xsl:text><xsl:apply-templates select="title"/>
</fo:block>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subchapter"/>
</xsl:template>
```

Ein wichtiges Element für die Druckausgabe ist `paragraph`, da es den Container für den Fließtext darstellt. Attribute wie Sprache des Textes und Silbentrennung werden hier angegeben, ebenso eine um fünf Millimeter eingerückte erste Textzeile.

```
<xsl:template match="paragraph">
<fo:block font-size="12pt"
font-family="serif"
line-height="18pt"
language="de"
text-align="justify"
text-align-last="start"
space-before="0mm"
space-after.optimum="2mm"
space-after.minimum="0mm"
text-indent="5mm"
hyphenate="true"
hyphenation-character="-"
keep-with-next="always">
```

```
<xsl:apply-templates/>
</fo:block>
</xsl:template>
```

Die Attribute `space-before`, `space-after.optimum` und `space-after.minimum` bestimmen den Abstand den der XSL-FO Prozessor vor und nach einem Absatz im besten Fall (`space-after.minimum`) oder im ungünstigsten Fall (`space-after.minimum`) lassen soll, bevor das nächste Formatting Objekt ausgegeben wird.

Die Transformation des Mulitmedia-Elements `asset` ist in der Druckversion denkbar einfach: da weder Audio- noch Video-Objekte in das Medium Papier eingebunden werden können, wird an dessen Stelle eine entsprechende Platzhaltergrafik ausgegeben. Das Einfügen von Bildern wird mit Hilfe des Elements `fo:external-graphic` vorgenommen. Da FOP sämtliche Grafiken so lange skalierte bis Breiten- und Höhenangaben in Millimetern vorlagen, werden hier die entsprechenden Werte aus den Attributen `width_in_mm` und `height_in_mm` übergeben und "mm" angefügt.

Sämtliche Grafiken werden rechtsbündig ausgegeben. Unter dem Bild wird ebenfalls rechtsbündig ein `fo:block` Element ausgegeben, das als Text das Wort "Abbildung", eine durch `xsl:number` ermittelte Anzahl von `asset` Elementen, einen Doppelpunkt und den Inhalt des `alt` Attributs des `asset` Elements enthält. Der gleiche Inhalt wird auch im Abbildungsverzeichnis verwendet. Damit die Angabe der Seitenzahl im Abbildungsverzeichnis korrekt ist, wird auch bei `fo:external-graphic` die `id` vererbt.

```
<xsl:template match="asset">
<xsl:choose>
<xsl:when test="@type='image'">
<fo:block text-align="end">
<fo:external-graphic keep-with-next="always">
<xsl:attribute name="id">
<xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:attribute name="src">
<xsl:value-of select="@source"/>
</xsl:attribute>
<xsl:attribute name="width">
<xsl:value-of select="@width_in_mm"/>
<xsl:text>mm</xsl:text>
</xsl:attribute>
<xsl:attribute name="height">
<xsl:value-of select="@height_in_mm"/>
<xsl:text>mm</xsl:text>
</xsl:attribute>
```

```
</fo:external-graphic>
<fo:block font-size="10pt"
font-style="italic"
text-align="end"
keep-with-previous="always">
<fo:inline font-weight="bold">
<xsl:text>Abbildung </xsl:text>
<xsl:number count="asset" level="any"/>
<xsl:text>: </xsl:text>
</fo:inline>
<fo:inline><xsl:value-of select="@alt"/></fo:inline>
</fo:block>
</fo:block>
</xsl:when>
<xsl:when test="@type='audio'">
<fo:external-graphic src="images/audio.gif"/>
</xsl:when>
<xsl:when test="@type='video'">
<fo:external-graphic src="images/video.gif"/>
</xsl:when>
</xsl:choose>
</xsl:template>
```

Die Ausgabe des Literaturverzeichnisses entspricht dem Vorgehen bei der HTML-Ausgabe, auch im Layout. Daher wird an dieser Stelle nicht gesondert darauf eingegangen. Auch das Template zur Transformation des Abkürzungsverzeichnisses enthält keinerlei Besonderheiten. Hier wird wie beim Inhaltsverzeichnis eine Tabelle als Layout-Hilfsmittel genutzt.

Da es in XSL-FO keine Unterscheidung zwischen geordneten und ungeordneten Listen gibt, muss bei der Transformation des Elements `list` entsprechend als `xsl:number` als Inhalt von `fo:list-item-label` zum Einsatz kommen. Zu Beginn dieses Templates steht also eine Unterscheidung, ob die Liste geordnet oder ungeordnet ist. Falls es einen Listentitel gibt, wird er in einem Block Element ausgegeben. Bei den Listenelementen kommt bei geordneten Listen die Anzahl der `fo:list-item` Elemente, gefolgt von einem Punkt, bei ungeordneten Listen ein Bullet-Punkt (`&#8226;`) zum Einsatz.

```
<xsl:template match="list">
<fo:block font-size="12pt" font-family="serif" line-
height="18pt">
<xsl:choose>
<xsl:when test="@type='ordered'">
<xsl:for-each select="title">
```

```
<fo:block>
<xsl:apply-templates/>
</fo:block>
</xsl:for-each>
<fo:list-block>
<xsl:for-each select="item">
<fo:list-item>
<fo:list-item-label end-indent="label-end()">
<fo:block><xsl:number count="item"/>
<xsl:text>.</xsl:text>
</fo:block>
</fo:list-item-label>
<fo:list-item-body start-indent="body-start()">
<fo:block font-size="12pt"
font-family="serif"
line-height="18pt">
<xsl:apply-templates/>
</fo:block>
</fo:list-item-body>
</fo:list-item>
</xsl:for-each>
</fo:list-block>
</xsl:when>
<xsl:otherwise>
<xsl:for-each select="title">
<fo:block>
<xsl:apply-templates/>
</fo:block>
</xsl:for-each>
<fo:list-block>
<xsl:for-each select="item">
<fo:list-item>
<fo:list-item-label end-indent="label-end()">
<fo:block>&#8226;</fo:block>
</fo:list-item-label>
<fo:list-item-body start-indent="body-start()">
<fo:block>
<xsl:apply-templates/>
</fo:block>
</fo:list-item-body>
</fo:list-item>
```

```
</xsl:for-each>
</fo:list-block>
</xsl:otherwise>
</xsl:choose>
</fo:block>
</xsl:template>
```

Ein interessantes Problem tritt beim Template für `reference` auf. Der verwendete XSL-FO Prozessor FOP gibt `reference` Elemente, die in `item` Elementen, also innerhalb einer Liste stehen, zwar inhaltlich korrekt aus, stellt sie allerdings eingerückt dar. Die Einrückung am linken Rand entspricht genau der Einrückung, die FOP benutzt, um Listenpunkte einzurücken (`start-indent="body-start()"`). Hierzu war allerdings ein Blick in das FO-Ausgabedokument nötig. Da der `footnote-body` innerhalb des `fo:list-item-body` steht, wird er ebenfalls eingerückt. Die Lösung ist einfach: dem `fo:block` Element innerhalb des `fo:footnote-body` wird als weiteres Attribut `start-indent="0mm"` hinzugefügt.

```
<xsl:template match="reference">
<fo:footnote>
<fo:inline font-size="8pt" vertical-align="super">
<xsl:number count="reference" level="any"/>
</fo:inline>
<fo:footnote-body>
<fo:block line-height="13pt" start-indent="0mm">
>
<fo:inline vertical-align="super" font-size="8pt">
<xsl:number count="reference" level="any"/>
</fo:inline>
<fo:inline font-size="10pt">
<xsl:value-of select="@refered_to"/>
</fo:inline>
<fo:inline font-size="10pt">
<xsl:apply-templates select="reftext"/>
</fo:inline>
</fo:block>
</fo:footnote-body>
</fo:footnote>
</xsl:template>
```

Wie schon mehrfach angesprochen, werden `link` Elemente in der Art transformiert, dass die URL in Klammern an den Inhalt des Elements angehängt wird.

```
<xsl:template match="link">
<fo:inline> <xsl:value-of select="."/> <fo:inline>
```

```
<fo:inline font-style="italic">
<xsl:text>(</xsl:text>
<xsl:value-of select="@xlink:href"/>
<xsl:text>)</xsl:text>
</fo:inline>
</xsl:template>
```

Auf eine Darstellung der Templates für die Elemente `code`, `address`, `date`, `authors`, `emph`, `strong`, `quotation` und `comment` wurde aus Platzgründen verzichtet. Eine Besonderheit stellt allerdings die Transformation von `codeline` dar. Stehen Quelltextzeilen im Fließtext werden sie um je fünf Millimeter links und rechts eingerückt und mit einem grauen Hintergrund hinterlegt - wie auch bei der HTML-Ausgabe. Da im Anhang dieser Arbeit allerdings der Quelltext in recht geballter Form steht, ist sowohl eine Einrückung als auch der graue Hintergrund nicht nur aus ästhetischen Gründen unsinnig - sondern auch unökonomisch. Aus diesem Gund ist für das Element `codeline` im Anhang der Arbeit ein separates Template erstellt worden.

```
<xsl:template match="codeline">
<fo:block font-family="monospace"
start-indent="5mm" end-indent="5mm"
background-color="#cccccc"
text-align="start"
font-size="11pt">
<xsl:value-of select="."/>
</fo:block>
</xsl:template>
```

```
<xsl:template match="//appendix/*/*/*/*/*/codeline">
<fo:block font-family="monospace"
text-align="start"
font-size="11pt">
<xsl:value-of select="."/>
</fo:block>
</xsl:template>
```

Mit dem hier in Auszügen vorgestellten XSLT Style Sheet wird die Transformation nach XSL-FO realisiert. Für die Druckausgabe ist allerdings noch die anschließende Überführung nach PDF mit Hilfe von FOP nötig.

8 Printausgabe mit FOP und Acrobat Reader

Die HTML-Ausgabe kann nach der Generierung sofort im Browser angesehen werden. Für die XSL-FO-Ausgabe trifft das nicht zu, da noch keine Software XSL-FO nativ unterstützt. Daher wird die schon angesprochene Überführung in ein druckfähiges Format wie PDF nötig, das mit einem Programm wie Acrobat Reader ausgedruckt werden kann.

PDF ist ein von Adobe entwickeltes Nachfolgeformat zu PostScript. Während PostScript, das von vielen Laserdruckern nativ unterstützt wird, vor allem zur Druckausgabe erschaffen wurde, kombiniert PDF dessen Druckmöglichkeiten mit Linking- und Sicherheitsfunktionen, sowie einer Kompression zur Speicherung der Daten. Vor allem durch die Sicherheitsfunktionen hat sich PDF als wichtiges Austauschformat von Dateien, aber auch zur Druckvorstufe etabliert. Der kostenlos für alle wichtigen Betriebssysteme (inkl. Linux, PalmOS und Windows CE) downloadbare Acrobat Reader ermöglicht es, PDF-Dateien darzustellen und auszudrucken (solange nicht die Druckfunktion vom Autor gesperrt wurde).

Zur Erstellung von PDF-Dateien gibt es mehrere Möglichkeiten: die komfortabelste ist die Nutzung von Adobe Acrobat, einem kostenpflichtigen Programm, das es erlaubt, aus jeder beliebigen Anwendung heraus PDF zu erzeugen. Ähnliche Möglichkeiten, wenn auch um einiges komplizierter, bietet das kostenlos erhältliche Ghostview. Eine weitere Möglichkeit ist LaTeX, das mit Hilfe von PDF-TeX native LaTeX-Dokumente nach PDF wandelt.

Die für diese Arbeit entscheidende Möglichkeit ist die Benutzung eines XSL-FO Prozessors, der XSL-FO nach PDF transformiert. Momentan sind zwei Prozessoren bekannt: XEP *(http://www.renderx.com/FO2PDF.html)* von RenderX *(http://www.renderx.com)* und FOP vom XML Apache Project.

XEP ist ein kommerzielles Programm, das unter anderem vom W3C eingesetzt wird, um die W3C Candidate Recommendations nach PDF zu transformieren. In der Nutzung eingeschränkte Trial-Versionen der Software sind auf den Webseiten des Herstellers bereit gestellt. FOP dagegen ist ein frei verfügbarer XSL-FO Prozessor, der zwar den Fähigkeiten von XEP etwas nachsteht, sich dafür in den letzten Monaten aber rasant entwickelt hat.

Nach dem Download der Binärversion von FOP und Entpacken in ein beliebiges Verzeichnis kann FOP mit Hilfe der Batch-Datei fop.bat gefolgt von der Eingabedatei und der Ausgabedatei gestartet werden. Das nachstehende Beispiel erzeugt aus der XSL-FO-Datei `Magisterarbeit.fo` eine PDF-Datei `Magisterarbeit.pdf`.

```
fop Magisterarbeit.fo Magisterarbeit.pdf
```

Eine vollständigere Anleitung bieten sowohl Knobloch und Kopp[96] als auch Herold[97].

[96] S. 206 f.
[97] Abschnitt "Formatting Objects and Their Properties", "Using FOP"

Wie an der Druckversion dieser Arbeit zu sehen ist, lassen sich mit der PDF-Ausgabe von FOP schon brauchbare Ergebnisse erzielen. Man darf gespannt sein, mit welcher Geschwindigkeit die FOP-Entwickler in der Lage sein werden, den vollständigen XSL-FO Umfang zu implementieren.

9 Theorie und Praxis - die Probleme bei der Benutzung neuer Technologien

Wie vielleicht schon in vorhergehenden Abschnitten dieser Arbeit anklungen ist, hat das Arbeiten mit relativ jungen Technologien leider nicht nur Vorteile. Travis und Waldt haben über ihre Erfahrungen bei der Erstellung von SGML-Anwendungen ein ganzes Kapitel geschrieben.[98] Darin geben die Autoren eine Reihe von Tipps, die bei Beachtung ein aufwändiges Redesign ersparen. Auf Grund dieser Aussagen konnten einige Fehler bei der Erstellung der DTD vermieden werden. Die Empfehlung zur Benutzung von Parameter Entitäten[99] ist bei der Definition der einzelnen Elemente befolgt worden. Auch die Hinweise zur Vermeidung doppelt definierter Elemente bei Verwendung einer fragmentierten DTD, wie es auch in dieser Arbeit der Fall ist, sind vor allem bei noch größeren Projekten hilfreich.[100]

Besondere Beachtung sollte man sicherlich der Vermeidung von Ambiguitäten im Inhaltsmodell widmen. Sehr hilfreich ist hier XMetaL, der ausdrücklich vor solchen warnt, falls innerhalb einer DTD Ambiguitäten vorliegen. Es gibt in SGML und XML eigentlich immer eine weitere Lösung, um ein gewünschtes Informationsmodell zu realisieren. Sicherlich sind solche Lösungen meist etwas unbequemer in der Definition, aber dadurch ist man vor unvorhersehbaren Reaktionen des Parsers sicher.[101]

Ist man solchen Problemen aus dem Weg gegangen, warten allerdings weitere Probleme bei der praktischen Arbeit mit XML-Dokumenten. Auch wenn man bei XML im Gegensatz zu SGML schon jetzt eine vergleichsweise große Auswahl an Software zur Verfügung hat, hinkt die Umsetzung einiger Teile des Standards den real nutzbaren Programme hinterher.

Sinnvoll wäre es zum Beispiel, aus einer XML-Datei mit Hilfe von XSLT-Style Sheets mehrere HTML-Ausgabedateien zu erzeugen. Vor allem einzelne Seiten für die jeweiligen Kapitel wären wünschenswert. Allerdings unterstützt der XSLT-Standard in der Version 1.0 *(http://www.w3.org/TR/2001/REC-xsl-20011015)* eine solche Transformation nicht. Die kommende Version XSLT 1.1 *(http://www.w3.org/TR/xslt11/)* sieht mit dem Element `xsl:document` eine Lösung vor. Damit ist es möglich, mehrere Ausgabedateien zu produzieren. SAXON unterstützt schon seit einiger Zeit `xsl:document` (wenn auch zuerst als `saxon:output`). Dennoch wurde in dieser Arbeit auf einen Einsatz verzichtet, um vollständige Kompatibilität mit dem derzeitigem Standard zu wahren.

[98] SGMLMIG S. 301 ff.
[99] SGMLMIG S. 302 ff.
[100] SGMLMIG S. 308
[101] SGMLMIG S. 315

Der Großteil der während der Entstehung dieser Arbeit aufgetretenen Probleme betrifft die Druckausgabe. Das war allerdings vorauszusehen, da XSL-FO noch ein sehr junger Teil des XSL Standards ist, vor allem, was die Unterstützung durch Software angeht. Der hier zum Einsatz kommende XSL-FO Prozessor FOP in der Version 0.20.1 machte trotz der geringen Versionsnummer seine Sache erstaunlich gut. Dennoch ließen sich Schwierigkeiten nicht vermeiden. Größtes Problem war die Platzierung von Bildern. Da FOP das `fo:float` Element noch nicht unterstützt, wurden die Bilder genau an der Stelle im Text eingefügt, an der das Element `asset` im Originaldokument stand. Durch den beschränkten Platz, der auf einer Druckseite im Gegensatz zu einem HTML-Dokument zur Verfügung steht, war es unvermeidlich, dass Bilder allein auf der Seite platziert wurden, wobei vor allem der Zusammenhang zwischen Bild und ursprünglich umgebenden Text litt. Um die Screenshots von den Programmen XMetaL und Emacs nicht bis zur Unkenntlichkeit verkleinern zu müssen, wurde die Anzahl der Bilder verringert und teilweise auf Bildausschnitte zurück gegriffen.

Ein ganz anderes, unerwartetes Problem gab bei den Fußnoten. Dieses Problem und die Lösung desselben wurden allerdings schon im entsprechenden Abschnitt zum Style Sheet ausreichend gewürdigt.

Bei der Umsetzung von Links zeigte sich bei der Druckausgabe, dass viele sehr lange und von FOP nicht trennbare URLs den Blocksatz nachhaltig störten. Aus diesen Gründen wurde im XML-Dokument jeweils nur das erste Vorkommen eines Links als solcher gekennzeichnet, alle weiteren wurden nicht entsprechend ausgezeichnet.

Insgesamt lässt sich feststellen, dass erwartungsgemäß Probleme bei der Realisierung dieser Magisterarbeit aufgetreten sind. Keiner der hier skizzierten Nachteile hat allerdings die Fertigstellung des Projektes verhindert oder ernsthaft verzögert. Mit voran schreitender Unterstützung und Verbreitung von XML und der Weiterentwicklung der in dieser Arbeit eingesetzten Software werden die Möglichkeiten, die dem Nutzer von XML und XSL zur Verfügung stehen, dagegen noch stark erweitert.

10 Schluss

Anspruch dieser Arbeit war und ist es zu zeigen, dass mit XML und XSL Single-Source-Publishing schon heute möglich ist. Das macht die eXtensible Markup Language nicht nur für Unternehmen interessant, die ja schon verstärkt XML zu diesem Zweck einsetzen[102], sondern auch für "Normalsterbliche". Das Beispiel dieser Magisterarbeit hat gezeigt, dass die genannten Softwareprodukte zusammen eingesetzt schon nahezu die gleiche Funktionalität erreichen wie traditionelle Werkzeuge. In manchen Teilen geht die hier vorgestellte Entwicklungsumgebung sogar über die Fähigkeiten von Word und Co. weit hinaus und ermöglicht echtes Single-Source-Publishing, nicht einfach nur eine Umsetzung bzw. Speicherung in einem anderen Format. Als Beispiel sei das unterschiedliche Verhalten bei Links genannt.

Die hier vorgestellte Lösung ist in vielerlei Hinsicht vom Optimum noch weit entfernt. Das beginnt bei den eigenen Teilen der Arbeit wie der DTD und den XSLT Style Sheets, die sicherlich noch an manchen Stellen verbesserungswürdig sind, und hört bei der verwendeten Software, kommerzielle oder freie, nicht auf. Noch erkauft man sich den Zusatznutzen, den die hier vorgestellte Lösung bietet, durch einen erhöhten Aufwand bei der Erstellung der XML-Texte. Obwohl hier vor allem XMetaL schon recht gute Unterstützung bietet, von einigen Nachteilen, die ja im entsprechenden Kapitel angesprochen wurden, mal abgesehen.

Sinnvollerweise sollte an dieser Stelle auch nicht der Ausblick in die Zukunft fehlen. Die Arbeit an diesem Projekt hat noch einige verbesserungswürdige Stellen offenbart. Und das, obwohl sowohl die DTD als auch die zur Transformation eingesetzten Skripte im Laufe des Entstehungsprozesses immer wieder angepasst wurden, um auf sinnvolle Wünsche zu reagieren oder Missstände, die sich erst bei der praktischen Erprobung zeigten, abzustellen.

In einer verbesserten Version der DTD sollte dem Umstand Rechnung getragen werden, dass die Verwendung von `link` im `title` Element zumindest bei dem in der Printversion gewählten Umgang mit Links problematisch ist. Für Überschriften in Kapiteln und anderen Abschnitten der Arbeit könnte ein weiteres Element definiert werden, dass keine Links enthalten darf, auch wenn ein solches Vorgehen die Mehrfachnutzung von Elementen etwas untergräbt.

Sicherlich ist auch der Gebrauch des Elements `reference` noch verbesserungswürdig. Denkbar wäre eine zweiteilige Referenz, bestehend aus einem Element, das wirklich nur eine Referenz (als Ziel und als Ausgangspunkt) anzeigt und einem optionalen Text. Eine solche Referenz wäre dann auch bei Querverweisen im gleichen Dokument einfacher zu nutzen. In der aktuellen Version wurde darauf verzichtet, da `fo:page-number-citation` von der FOP Entwicklergemeinde noch als experimentell bezeichnet wird und nur innerhalb von Verzeichnissen einwandfrei funktioniert. Störend hat sich bei der Arbeit mit der DTD das Fehlen

[102] XML ist Zugpferd in Microsofts .NET Vision. Sun wird ab der kommenden Version 6.0 von StarOffice XML als universelles Dateiformat benutzen.

einer allgemeinen Referenzmöglichkeit auf Quellen im Literaturverzeichnis erwiesen. Zwar ist es möglich, eine leere Referenz zu erzeugen, allerdings wird diese immer als Fußnote bzw. Endnote umgesetzt. Eine Referenz im Fließtext wäre hier wünschenswert, ist allerdings aus zeitlichen Gründen nicht mehr implementiert worden.

Darüber hinaus wäre eine Funktion, die bei aufeinander folgenden Fußnoten aus der gleichen Quelle eine Ersetzung des Bezeichners in "ibid" oder "ebd" vornimmt eine sinnvolle Erweiterung. Mit einer solchen Funktion würde die hier skizzierte Entwicklungsumgebung auch im rein textlichen Bereich weitaus mehr bieten als Textverarbeitungsprogramme und einen semantischen Aspekt berücksichtigen.

Bei einer späteren Version der Style Sheets würde mit Sicherheit die Funktion `generate-id()` öfter in Anspruch genommen werden. Sie bietet sich für alle Elemente an, auf die nicht per `IDREF` Attribut schon im XML-Dokument zugegriffen wird. Vor allem bei größeren Arbeiten ist es ein erhöhter Aufwand, jedem Element eine `id` zu geben, die dann im Style Sheet zur Transformation in das jeweilige Element des Ausgabeformats vererbt werden muss.

Davon abgesehen ist auch die doppelte Behandlung von Elementen auf Grund ihres Vorkommens im `body` oder `appendix` der Arbeit zumindest aus softwareökonomischen Gesichtspunkten etwas unglücklich.

Was die Umsetzung in die Ausgabeformate angeht, gibt es auch hier noch Verbesserungen. Worte, die gleichzeitig als Abkürzungen oder als Link getaggt werden könnten, wie zum Beispiel W3C können zwar im XML-Teil der Arbeit sowohl als auch sein. Bei der Überführung nach HTML geht hier allerdings eine Information verloren. In dieser Arbeit wurde daher Abkürzungen vorrangig behandelt, da im Abkürzungsverzeichnis Links stehen können.

Die hier genannten Punkte machen zweierlei deutlich: der Weg zu einer in jederlei Hinsicht perfekten Entwicklungsumgebung ist noch lang und steinig. Allerdings gibt es viel verspechende Ansätze, die es als mehr als lohnend erscheinen lassen, die Entwicklung in diesem Bereich weiter zu verfolgen. Schließlich winkt am Ende nicht mehr als *eine Integration von Text, Grafik und Animationen, Klängen und Sprache, die bisher nicht möglich war: die Integration auf der Ebene der Erzeugung dieser Formate* und damit eine Revolution in der Pflege und Weiterentwicklung von Inhalten.[103]

[103] WEBXML S. 207

A Anhang

In dem Anhang findet sich die DTD, mit der diese Arbeit geschrieben wurde, und die Style Sheets zur Überführung nach HTML und XSL-FO.

A.1 Die DTD

Nähere Erläuterungen der einzelnen Elemente, Attribute und Entitäten finden sich im entsprechenden Abschnitt dieser Arbeit.

A.1.1 magisterarbeit.dtd

```
<!ENTITY % text "#PCDATA | link | asset | emph | strong | quotation
| abbrevation | code | codeline | reference | list | comment ">
<!ENTITY % bibtext "#PCDATA | link | quotation | abbrevation |
code">
<!ENTITY % bib SYSTEM "bibliography.ent">
<!ENTITY % gloss SYSTEM "glossary.ent">
<!ENTITY % abb SYSTEM "abbrevations.ent">

<!NOTATION jpg SYSTEM "iexplore.exe">
<!NOTATION gif SYSTEM "iexplore.exe">
<!NOTATION rm SYSTEM "iexplore.exe">
<!NOTATION ram SYSTEM "iexplore.exe">
<!NOTATION avi SYSTEM "iexplore.exe">
<!NOTATION mov SYSTEM "iexplore.exe">
<!NOTATION wav SYSTEM "iexplore.exe">
<!NOTATION mp3 SYSTEM "iexplore.exe">

<!ELEMENT magisterarbeit (meta, body, appendix, bibliography)>

<!ELEMENT meta (title, subtitle?, authors, revision?)>
<!ELEMENT title (%bibtext;)*>
<!ELEMENT subtitle (%bibtext;)*>
<!ELEMENT authors (author+)>
<!ELEMENT author ((name, address*, phone*, e-mail*, matrikel_nr?) |
organization)>
<!ELEMENT name ((first_name+, surname) | (surname, first_name+))>
<!ELEMENT surname (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT organization (#PCDATA | link)*>
<!ELEMENT address ((street, nr)*, postal_code*, town*, country*)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT nr (#PCDATA)>
<!ELEMENT postal_code (#PCDATA)>
```

```
<!ELEMENT town (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT e-mail (#PCDATA)>
<!ELEMENT matrikel_nr (#PCDATA)>
<!ELEMENT date (day?, month?, year)>
<!ELEMENT day (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT year (#PCDATA)>

<!ELEMENT link ANY>
<!ATTLIST link
xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
xlink:type (simple) #FIXED "simple"
xlink:href CDATA #REQUIRED
xlink:title CDATA #IMPLIED
xlink:show (new) #FIXED "new"
xlink:actuate (onRequest) #FIXED "onRequest"
range (intern | extern) #REQUIRED>

<!ELEMENT body (chapter+)>
<!ELEMENT chapter (title, paragraph+, subchapter*)>
<!ATTLIST chapter id ID #REQUIRED>
<!ELEMENT subchapter (title, paragraph+, section*)>
<!ATTLIST subchapter id ID #REQUIRED>
<!ELEMENT section (title, paragraph+, subsection*)>
<!ATTLIST section id ID #REQUIRED>
<!ELEMENT subsection (title, paragraph*, subsubsection*)>
<!ATTLIST subsection id ID #REQUIRED>
<!ELEMENT subsubsection (title, paragraph+)>
<!ATTLIST subsubsection id ID #REQUIRED>
<!ELEMENT paragraph (%text;)*>

%gloss;
%abb;
%bib;
<!ELEMENT appendix (apps*, glossary?, abbrevations?, images?)>
<!ELEMENT apps (chapter+)>
<!ELEMENT images (title)>
<!ATTLIST images id ID #REQUIRED>
<!ELEMENT asset ANY>
<!ATTLIST asset
id ID #REQUIRED
```

```
source CDATA #REQUIRED
type (video | audio | image) #REQUIRED
format NOTATION (gif|jpg|rm|ram|avi|mov|wav|mp3) "gif"
width CDATA #IMPLIED
height CDATA #IMPLIED
width_in_mm CDATA #IMPLIED
height_in_mm CDATA #IMPLIED
alt CDATA #REQUIRED
>
<!ELEMENT emph (%text;)*>
<!ELEMENT strong (%text;)*>
<!ELEMENT quotation (%text;)*>
<!ELEMENT abbrevation (%text;)*>
<!ATTLIST abbrevation
long_form IDREF #REQUIRED
>
<!ELEMENT term (%bibtext;)*>
<!ATTLIST term definition IDREF #REQUIRED>
<!ELEMENT list (title*, item)+>
<!ATTLIST list type (ordered | unordered) #REQUIRED>
<!ELEMENT item (%text;)*>
<!ELEMENT code (#PCDATA)>
<!ELEMENT codeline (#PCDATA)>
<!ELEMENT reference (reftext)>
<!ATTLIST reference refered_to IDREF #IMPLIED>
<!ELEMENT reftext (%text;)*>
<!ELEMENT comment (%text;)*>
```

A.1.2 glossary.ent

```
<!ELEMENT glossary (title, gloss_item+)>
<!ATTLIST glossary id ID #REQUIRED>
<!ELEMENT gloss_item (term, definition)>
<!ATTLIST gloss_item def ID #REQUIRED>
<!ELEMENT definition (%text;)*>
```

A.1.3 abbrevations.ent

```
<!ELEMENT abbrevations (title, abb_item+)>
<!ATTLIST abbrevations id ID #REQUIRED>
<!ELEMENT abb_item (abbrevation, longform)>
<!ELEMENT longform (%bibtext;)*>
<!ATTLIST longform id ID #REQUIRED>
```

A.1.4 bibliography.ent

```
<!ELEMENT bibliography (title, biblio_item+)>
<!ATTLIST bibliography id ID #REQUIRED>
<!ELEMENT biblio_item (authors, title, subtitle*, source*, publis-
her*, address*, date)>
<!ATTLIST biblio_item
designator ID #REQUIRED
type (Book | Inbook | Journalarticle | Proceedings | Inproceedings |
Phdthesis | Mastersthesis | Techreport | Manual | Newspaperarticle |
AV | Internet | Unpublished) #REQUIRED>
<!ELEMENT source (%bibtext;)*>
<!ELEMENT publisher (%bibtext;)*>
```

A.2 Die XSLT Style Sheets

Mit den folgenden Style Sheets wurde die Überführung der Magisterarbeit nach HTML und nach XSL-FO realisiert. Anmerkungen und Erläuterungen finden sich in den entsprechenden Abschnitten der Arbeit.

A.2.1 Überführung nach HTML

```
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xlink="http://www.w3.org/1999/xlink" >
<xsl:output method="html"
doctype-system="http://www.w3.org/TR/html4/loose.dtd"
doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"
encoding="ISO-8859-1"
indent="yes"/>

<xsl:template match="/">
<html>
<xsl:apply-templates/>
</html>
</xsl:template>

<xsl:template match="meta">
<head>
<meta name="author">
<xsl:attribute name="content">
<xsl:for-each select="authors/author">
<xsl:value-of select="name/first_name"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/surname"/>
< xsl:if test="position()!=last()"> <xsl:text>, </xsl:text>
</xsl:if>
</xsl:for-each>
</xsl:attribute>
</meta>
<link rel="stylesheet" type="text/css" href="maggi.css"/>
<title>
<xsl:value-of select="title"/>
</title>
</head>
</xsl:template>
```

```
<xsl:template match="body">
<body>
<div align="center">
<h1><xsl:value-of select="//meta/title"/></h1>
<h2><xsl:value-of select="//meta/subtitle"/></h2>
<h3>Magisterarbeit der <xsl:value-of select="//meta/faculty"/> der
<xsl:value-of select="//meta/university"/></h3>
<h4><xsl:value-of select="//meta/department"/></h4>
<h5>Gutachter:<br/>
<xsl:for-each select="//meta/supervisors/supervisor">
<xsl:value-of select="@grade"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/first_name"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/surname"/>
<xsl:if test="position()!=last()"> <br/> </xsl:if>
</xsl:for-each>
</h5>
<h5>vorgelegt von: </h5>
<xsl:for-each select="//meta/authors/author">
<xsl:value-of select="name/first_name"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/surname"/>
<br/>
<xsl:if test="address"><xsl:apply-templates select="address"/> <br/>
</xsl:if>
<xsl:if test="matrikel_nr">Matrikel-Nr.: <xsl:value-of se-
lect="matrikel_nr"/> <p/>
</xsl:if>
<xsl:if test="position()!=last()"> <br/> </xsl:if>
</xsl:for-each>
</div>
<table border="0" cellpadding="2" cellspacing="2">
<xsl:for-each select="chapter">
<tr><td>
<span class="level1"> <xsl:number count="chapter"/> </span>
<xsl:text> </xsl:text>
<a class="level1">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
```

```
</a>
</td></tr>
<xsl:for-each select="subchapter">
<tr><td>
<span class="level2"> <xsl:number count="chapter"/>.<xsl:number
count="subchapter"/> </span>
<xsl:text> </xsl:text>
<a class="level2">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
<xsl:for-each select="section">
<tr><td>
<span class="level3"> <xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/> </span>
<xsl:text> </xsl:text>
<a class="level3">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
<xsl:for-each select="subsection">
<tr><td>
<span class="level4"> <xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/> </span>
<xsl:text> </xsl:text>
<a class="level4">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
<xsl:for-each select="subsubsection">
<tr><td>
<span class="level5"> <xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/>.<xsl:number count="subsubsection"/> </span>
```

```
<xsl:text> </xsl:text>
<a class="level5">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
<xsl:for-each select="//appendix/*/chapter">
<tr><td>
<span class="level1"> <xsl:number count="chapter" format="A"/>
</span>
<xsl:text> </xsl:text>
<a class="level1">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
<xsl:for-each select="subchapter"
<tr><td>
<span class="level2"> <xsl:number count="chapter"
format="A"/>.<xsl:number count="subchapter"/></span>
<xsl:text> </xsl:text>
<a class="level2">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
<xsl:for-each select="section">
<tr><td>
<span class="level3"> <xsl:number count="chapter"
format="A"/>.<xsl:number count="subchapter"/>.<xsl:number
count="section"/> </span>
<xsl:text> </xsl:text>
<a class="level3">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
```

```
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
<xsl:for-each select="subsection">
<tr><td>
<span class="level4"> <xsl:number count="chapter"
format="A"/>.<xsl:number count="subchapter"/>.<xsl:number
count="section"/>.<xsl:number count="subsection"/> </span>
<xsl:text> </xsl:text>
<a class="level4">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
<xsl:for-each select="subsubsection">
<tr><td>
<span class="level5"> <xsl:number count="chapter"
format="A"/>.<xsl:number count="subchapter"/>.<xsl:number
count="section"/>.<xsl:number count="subsection"/>.<xsl:number
count="subsubsection"/> </span>
<xsl:text> </xsl:text>
<a class="level5">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
<xsl:for-each select="//appendix/abbrevations">
<tr><td>
<a class="level1">
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
</xsl:for-each>
<xsl:for-each select="//bibliography">
<tr><td>
<a class="level1">
```

```
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="@id"/> </xsl:attribute>
<xsl:value-of select="title"/>
</a>
</td></tr>
</xsl:for-each>
</table>
<xsl:apply-templates/>
<hr/>
<xsl:apply-templates select="//*/reftext"/>
</body>
</xsl:template>

<xsl:template match="chapter">
<h2>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/> <xsl:text> </xsl:text>
<xsl:apply-templates select="title"/>
</h2>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subchapter"/>
</xsl:template>

<xsl:template match="apps/chapter">
<h2>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter" format="A"/><xsl:text> </xsl:text>
<xsl:apply-templates select="title"/>
</h2>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subchapter"/>
</xsl:template>

<xsl:template match="subchapter">
<h3>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/>.<xsl:number count="subchapter"/>
<xsl:text> </xsl:text><xsl:apply-templates select="title"/>
</h3>
<xsl:apply-templates select="paragraph"/>
```

```
<xsl:apply-templates select="section"/>
</xsl:template>

<xsl:template match="apps/*/subchapter">
<h3>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/> <xsl:text> </xsl:text><xsl:apply-templates se-
lect="title"/>
</h3>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="section"/>
</xsl:template>

<xsl:template match="section">
<h4>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/> <xsl:text>
</xsl:text> <xsl:apply-templates select="title"/>
</h4>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subsection"/>
</xsl:template>

<xsl:template match="apps/*/*/section">
<h4>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/> <xsl:text>
</xsl:text> <xsl:apply-templates select="title"/>
</h4>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subsection"/>
</xsl:template>

<xsl:template match="subsection">
<h5>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/>.<xsl:number
```

```
xsl:number count="section"/>.<xsl:number count="subsection"/>
<xsl:text> </xsl:text> <xsl:apply-templates select="title"/>
</h5>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subsubsection"/>
</xsl:template>

<xsl:template match="apps/*/*/*/subsection">
<h5>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/> <xsl:text> </xsl:text> <xsl:apply-templates se-
lect="title"/>
</h5>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subsubsection"/>
</xsl:template>

<xsl:template match="subsubsection">
<h6>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/> <xsl:text> </xsl:text> <xsl:apply-templates se-
lect="title"/>
</h6>
<xsl:apply-templates select="paragraph"/>
</xsl:template>

<xsl:template match="apps/*/*/*/*/subsubsection">
<h6>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/> <xsl:text> </xsl:text> <xsl:apply-templates se-
lect="title"/>
</h6>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subsubsection"/>
```

```
</xsl:template>

<xsl:template match="appendix">
<hr/><xsl:apply-templates select="apps"/>
<hr/><xsl:apply-templates select="abbrevations"/>
</xsl:template>

<xsl:template match="paragraph">
<p><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="link">
<xsl:choose>
<xsl:when test="@range='extern'">
<a class="extern">
<xsl:attribute name="href"> <xsl:value-of select="@xlink:href"/>
</xsl:attribute>
<xsl:attribute name="title"> <xsl:value-of select="@xlink:title"/>
</xsl:attribute>
<xsl:value-of select="."/>
</a>
</xsl:when>
<xsl:otherwise>
<a class="intern">
<xsl:attribute name="href"> <xsl:value-of select="@xlink:href"/>
</xsl:attribute>
<xsl:attribute name="title"> <xsl:value-of select="@xlink:title"/>
</xsl:attribute>
<xsl:value-of select="."/>
</a>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

<xsl:template match="asset">
<xsl:choose>
<xsl:when test="@type='image'">
<img>
<xsl:attribute name="src"> <xsl:value-of select="@source"/>
</xsl:attribute>
<xsl:attribute name="width"> <xsl:value-of select="@width"/>
</xsl:attribute>
<xsl:attribute name="height"> <xsl:value-of select="@height"/>
</xsl:attribute>
```

```
<xsl:attribute name="alt"> <xsl:value-of select="@alt"/>
</xsl:attribute>
<xsl:value-of select="."/>
</img>
</xsl:when>
<xsl:otherwise>
<object>
<xsl:attribute name="data"> <xsl:value-of select="@source"/>
</xsl:attribute>
<xsl:attribute name="alt"> <xsl:value-of select="@alt"/>
</xsl:attribute>
<xsl:value-of select="."/>
</object>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

<xsl:template match="emph">
<em><xsl:apply-templates/></em>
</xsl:template>

<xsl:template match="strong">
<strong><xsl:apply-templates/></strong>
</xsl:template>

<xsl:template match="quotation">
<cite><xsl:apply-templates/></cite>
</xsl:template>

<xsl:template match="abbrevation">
<a class="abb">
<xsl:attribute name="title"> <xsl:value-of se-
lect="id(@long_form)/."/> </xsl:attribute>
<xsl:apply-templates/>
</a>
</xsl:template>

<xsl:template match="code">
<code><xsl:apply-templates/></code>
</xsl:template>
<xsl:template match="codeline">
<div class="code"><xsl:apply-templates/></div>
</xsl:template>

<xsl:template match="reference">
```

```
<sup><a class="footnote">
<xsl:attribute name="name"> <xsl:text>r</xsl:text> <xsl:number
count="reference" level="any"/> </xsl:attribute>
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:number
count="reference" level="any"/> </xsl:attribute>
<xsl:attribute name="title"> <xsl:text>Zur
Literaturstelle</xsl:text> </xsl:attribute>
<xsl:text>[</xsl:text> <xsl:number count="reference" level="any"/>
<xsl:text>]</xsl:text>
</a></sup>
</xsl:template>

<xsl:template match="reftext">
<span class="footnotetext">
<a class="footback">
<xsl:attribute name="name"> <xsl:number count="reftext"
level="any"/> </xsl:attribute>
<xsl:attribute name="href"> <xsl:text>#r</xsl:text><xsl:number
count="reftext" level="any"/> </xsl:attribute>
<xsl:attribute name="title"> <xsl:text>zurück zum Text</xsl:text>
</xsl:attribute>
<xsl:text>&#8593;</xsl:text>
</a>
<xsl:text> </xsl:text>
<a>
<xsl:attribute name="href"> <xsl:text>#</xsl:text> <xsl:value-of se-
lect="../@refered_to"/> </xsl:attribute>
<xsl:attribute name="title"> <xsl:text>ausführlichere Informationen
zur Literatur</xsl:text> </xsl:attribute>
<xsl:value-of select="../@refered_to"/>
</a>
<xsl:text> </xsl:text>
<xsl:apply-templates/>
</span><br/>
</xsl:template>

<xsl:template match="list">
<xsl:choose>
<xsl:when test="@type='ordered'">
<xsl:for-each select="title">
<br/><xsl:apply-templates/>
</xsl:for-each>
<ol>
```

```
<xsl:apply-templates select="item"/>
</ol>
</xsl:when>
<xsl:otherwise>
<xsl:for-each select="title">
<br/><xsl:apply-templates/>
</xsl:for-each>
<ul>
<xsl:apply-templates select="item"/>
</ul>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

<xsl:template match="comment">
</xsl:template>

<xsl:template match="apps">
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="bibliography">
<hr/><h2>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:apply-templates select="title"/>
</h2>
<p>
<xsl:for-each select="biblio_item">
<xsl:sort select="@designator"/>
<a>
<xsl:attribute name="href">
<xsl:text>javascript:history.go(-2)</xsl:text> </xsl:attribute>
<xsl:attribute name="title"> <xsl:text>zurück zur
Textstelle</xsl:text> </xsl:attribute>
<xsl:text>&#8592; </xsl:text>
</a>
<a>
<xsl:attribute name="name"> <xsl:apply-templates se-
lect="@designator"/> </xsl:attribute>
<strong><xsl:apply-templates select="@designator"/></strong>
</a><br/>
<xsl:apply-templates select="authors"/>
```

```
<xsl:text>.</xsl:text><br/>
<strong><xsl:apply-templates select="title"/></strong>
<xsl:text>.</xsl:text>
<xsl:for-each select="publisher">
<xsl:apply-templates/>
<xsl:if test="position()=last()"><br/></xsl:if>
</xsl:for-each>
<xsl:for-each select="address">
<xsl:apply-templates select="."/>
</xsl:for-each>
<xsl:for-each select="date">
<xsl:apply-templates select="."/>
</xsl:for-each>
<br/><br/>
</xsl:for-each>
</p>
</xsl:template>

<xsl:template match="authors">
<xsl:for-each select="author">
<xsl:choose>
<xsl:when test="name">
<xsl:apply-templates select="name/surname"/>
<xsl:text>, </xsl:text>
<xsl:for-each select="name/first_name">
<xsl:apply-templates select="."/>
<xsl:if test="position()!=last()"><xsl:text> </xsl:text> </xsl:if>
</xsl:for-each>
<xsl:if test="position()!=last()">; </xsl:if>
</xsl:when>
<xsl:when test="organization">
<xsl:apply-templates select="organization"/>
</xsl:when>
</xsl:choose>
</xsl:for-each>
</xsl:template>

<xsl:template match="abbrevations">
<h2>
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:apply-templates select="title"/>
</h2>
```

```
<table width="90%">
<xsl:for-each select="abb_item">
<xsl:sort select="abbrevation/@long_form"/>
<tr>
<td width="30%"><xsl:value-of select="abbrevation"/></td>
<td><xsl:apply-templates select="longform"/></td>
</tr>
</xsl:for-each>
</table>
</xsl:template>

<xsl:template match="address">
<xsl:for-each select="street">
<xsl:apply-templates/>
<xsl:text> </xsl:text>
<xsl:value-of select="../nr"/>
<br/>
</xsl:for-each>
<xsl:for-each select="postal_code">
<xsl:value-of select="."/>
<xsl:text> </xsl:text>
</xsl:for-each>
<xsl:for-each select="town">
<xsl:apply-templates/>
<xsl:choose>
<xsl:when test="position()!=last()"> <xsl:text>, </xsl:text>
</xsl:when>
<xsl:otherwise> <br/> </xsl:otherwise>
</xsl:choose>
</xsl:for-each>
<xsl:text> </xsl:text>
<xsl:for-each select="country">
<xsl:apply-templates/>
<xsl:text>, </xsl:text>
</xsl:for-each>
</xsl:template>

<xsl:template match="date">
<xsl:for-each select="day">
<xsl:apply-templates/>
<xsl:text>.</xsl:text>
</xsl:for-each>
<xsl:for-each select="month">
```

```
<xsl:apply-templates/>
<xsl:text>.</xsl:text>
</xsl:for-each>
<xsl:value-of select="year"/>
</xsl:template>
</xsl:stylesheet>
```

A.2.2 Überführung nach XSL-FO

```
<?xml version="1.0" encoding="UTF-16"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:fo="http://www.w3.org/1999/XSL/Format" >
<xsl:output indent="yes"/>
<xsl:template match="magisterarbeit">
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
<fo:layout-master-set>
<fo:simple-page-master >
master-name="title"
page-height="297mm"
page-width="210mm"
margin-bottom="15mm"
margin-left="20mm"
margin-right="25mm"
margin-top="20mm">
<fo:region-before extent="40mm"/>
<fo:region-body margin-top="40mm" margin-bottom="32mm"/>
<fo:region-after extent="32mm"/>
</fo:simple-page-master>

<fo:simple-page-master master-name="a4"
page-height="297mm"
page-width="210mm"
margin-bottom="20mm"
margin-left="25mm"
margin-right="25mm"
margin-top="15mm">
<fo:region-before extent="15mm"/>
<fo:region-body margin-top="15mm" margin-bottom="28mm"/>
<fo:region-after extent="16mm"/>
</fo:simple-page-master>
</fo:layout-master-set>
```

```
<fo:page-sequence master-name="title">
<fo:static-content flow-name="xsl-region-before">
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
text-align="start">
<xsl:value-of select="//meta/university"/>
</fo:block>
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
text-align="start">
<xsl:value-of select="//meta/faculty"/>
</fo:block>
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
text-align="start">
<xsl:value-of select="//meta/department"/>
</fo:block>
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
text-align="start">
Gutachter:
<xsl:for-each select="//meta/supervisors/supervisor">
<xsl:value-of select="@grade"/>
<xsl:text>
<xsl:value-of select="name/first_name"/>
<xsl:text>
<xsl:value-of select="name/surname"/>
<xsl:if test="position()!=last()"> <xsl:text>, </xsl:text> </xsl:if>
</xsl:for-each>
</fo:block>
</fo:static-content>
<fo:static-content flow-name="xsl-region-after">
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
text-align="start">
<xsl:for-each select="//meta/authors/author">
<fo:inline>
<xsl:value-of select="name/first_name"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/surname"/>
</fo:inline>
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
text-align="start">
<xsl:if test="address">
<xsl:apply-templates select="address"/>
</xsl:if>
```

```
</fo:block>
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
text-align="start">
<xsl:if test="matrikel_nr">
<xsl:text>Matrikel-Nr.: </xsl:text>
<xsl:value-of select="matrikel_nr"/>
</xsl:if>
</fo:block>
</xsl:for-each>
</fo:block>
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
<fo:block font-size="24pt" font-family="serif" line-height="32pt"
font-weight="bold" text-align="center" space-before="50mm">
<xsl:value-of select="//meta/title"/>
</fo:block>
<fo:block font-size="16pt" font-family="serif" line-height="24pt"
text-align="center" space-before="5pt">
<xsl:value-of select="//meta/subtitle"/>
</fo:block>
<fo:block font-size="14pt" font-family="serif" line-height="21pt"
text-align="center">
<xsl:text>- Magisterarbeit -</xsl:text>
</fo:block>
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
text-align="center" space-before="6pt">
vorgelegt von:
</fo:block>
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
text-align="center">
<xsl:for-each select="//meta/authors/author">
<fo:inline>
<xsl:value-of select="name/first_name"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name/surname"/>
</fo:inline>
</xsl:for-each>
</fo:block>
</fo:flow>
</fo:page-sequence>

<fo:page-sequence master-name="a4" initial-page-number="1" for-
```

```
mat="I">
<fo:static-content flow-name="xsl-region-before">
<fo:block font-size="12pt" text-align="start" font-style="italic">
<xsl:value-of select="meta/title"/>
<fo:block>
<fo:leader leader-pattern="rule" leader-length="155mm" rule-
thickness="1pt" color="black"/>
</fo:block>
</fo:block>
</fo:static-content>
<fo:static-content flow-name="xsl-region-after">
<fo:block font-size="12pt" text-align="end">
<fo:page-number/>
</fo:block>
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
<fo:block font-size="18pt"
font-family="serif"
line-height="27pt"
break-before="page"
text-align="justify">
Inhaltsverzeichnis
</fo:block>
<fo:table>
<fo:table-column column-width="25mm"/>
<fo:table-column column-width="120mm"/>
<fo:table-column column-width="8mm"/>
<fo:table-body font-family="serif">
<xsl:for-each select="body/chapter">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt"
font-weight="bold">
<xsl:number count="chapter"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt"
font-weight="bold">
<xsl:value-of select="title"/>
</fo:block>
```

```
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt" font-
weight="bold">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
<xsl:for-each select="subchapter">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:number count="chapter"/>.<xsl:number count="subchapter"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
<xsl:for-each select="section">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
```

```
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
<xsl:for-each select="subsection">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
<xsl:for-each select="subsubsection">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:number count="chapter"/>.<xsl:number
```

```
xsl:number count="section"/>.<xsl:number
count="subsection"/>.<xsl:number count="subsubsection"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
<xsl:for-each select="appendix/apps/chapter">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt"
font-weight="bold">
<xsl:number count="chapter" format="A"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt"
font-weight="bold">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt" font-
weight="bold">
<fo:page-number-citation>
```

```
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
<xsl:for-each select="subchapter">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
<xsl:for-each select="section">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
```

```
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
<xsl:for-each select="subsection">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
<xsl:for-each select="subsubsection">
<fo:table-row>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/>.<xsl:number count="subsubsection"/>
</fo:block>
</fo:table-cell>
```

```
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
<xsl:for-each select="appendix/abbrevations">
<fo:table-row>
<fo:table-cell>
<fo:block/>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt"
font-weight="bold">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt" font-
weight="bold">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
</xsl:for-each>
```

```
<xsl:for-each select="appendix/images">
<fo:table-row>
<fo:table-cell>
<fo:block/>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt"
font-weight="bold">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt" font-
weight="bold">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
</xsl:for-each>
<xsl:for-each select="bibliography">
<fo:table-row>
<fo:table-cell>
<fo:block/>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start" font-size="14pt" line-height="21pt"
font-weight="bold">
<xsl:value-of select="title"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end" font-size="14pt" line-height="21pt" font-
weight="bold">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
```

```
</fo:table-cell>
</fo:table-row>
</xsl:for-each>
</fo:table-body>
</fo:table>
</fo:flow>
</fo:page-sequence>

<fo:page-sequence master-name="a4" initial-page-number="1" lan-
guage="de">
<fo:static-content flow-name="xsl-region-before">
<fo:block font-size="12pt" text-align="start" font-style="italic">
<xsl:value-of select="meta/title"/>
<fo:block>
<fo:leader leader-pattern="rule" leader-length="155mm" rule-
thickness="1pt" color="black"/>
</fo:block>
</fo:block>
</fo:static-content>
<fo:static-content flow-name="xsl-region-after">
<fo:block font-size="12pt" text-align="end">
<fo:page-number/>
</fo:block>
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
<xsl:apply-templates select="body"/>
</fo:flow>
</fo:page-sequence>

<fo:page-sequence master-name="a4" language="de">
<fo:static-content flow-name="xsl-region-before">
<fo:block font-size="12pt" text-align="start" font-style="italic">
<xsl:value-of select="meta/title"/>
<fo:block>
<fo:leader leader-pattern="rule" leader-length="155mm" rule-
thickness="1pt" color="black"/>
</fo:block>
</fo:block>
</fo:static-content>
<fo:static-content flow-name="xsl-region-after">
<fo:block font-size="12pt" text-align="end">
<fo:page-number/>
</fo:block>
```

```
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
<xsl:apply-templates select="appendix"/>
</fo:flow>
</fo:page-sequence>

<fo:page-sequence master-name="a4">
<fo:static-content flow-name="xsl-region-before">
<fo:block font-size="12pt" text-align="start" font-style="italic">
<xsl:value-of select="meta/title"/>
<fo:block>
<fo:leader leader-pattern="rule" leader-length="155mm" rule-
thickness="1pt" color="black"/>
</fo:block>
</fo:block>
</fo:static-content>
<fo:static-content flow-name="xsl-region-after">
<fo:block font-size="12pt" text-align="end">
<fo:page-number/>
</fo:block>
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
<fo:block font-size="20pt" font-family="serif" line-height="30pt"
font-weight="bold" text-align="justify">
<xsl:attribute name="id"> <xsl:value-of se-
lect="appendix/images/@id"/> </xsl:attribute>
<xsl:value-of select="appendix/images/title"/>
</fo:block>
<fo:table>
<fo:table-column column-width="30mm"/>
<fo:table-column column-width="90mm"/>
<fo:table-column column-width="40mm"/>
<fo:table-body font-size="12pt" font-family="serif">
<xsl:for-each select="//*/*/*/*/asset">
<fo:table-row line-height="16pt">
<fo:table-cell>
<fo:block text-align="start">
<xsl:text>Abbildung </xsl:text> <xsl:number count="asset" le-
vel="any"/> <xsl:text>:</xsl:text>
</fo:block>
</fo:table-cell>
<fo:table-cell>
```

```
<fo:block text-align="start">
<xsl:value-of select="@alt"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="end">
<fo:page-number-citation>
<xsl:attribute name="ref-id"> <xsl:value-of select="@id"/>
</xsl:attribute>
</fo:page-number-citation>
</fo:block>
</fo:table-cell>
</fo:table-row>
</xsl:for-each>
</fo:table-body>
</fo:table>
</fo:flow>
</fo:page-sequence>

<fo:page-sequence master-name="a4" language="de">
<fo:static-content flow-name="xsl-region-before">
<fo:block font-size="12pt" text-align="start" font-style="italic">
<xsl:value-of select="meta/title"/>
<fo:block>
<fo:leader leader-pattern="rule" leader-length="155mm" rule-
thickness="1pt" color="black"/>
</fo:block>
</fo:block>
</fo:static-content>
<fo:static-content flow-name="xsl-region-after">
<fo:block font-size="12pt" text-align="end">
<fo:page-number/>
</fo:block>
</fo:static-content>
<fo:flow flow-name="xsl-region-body">
<xsl:apply-templates select="bibliography"/>
</fo:flow>
</fo:page-sequence>
</fo:root>
</xsl:template>

<xsl:template match="body">
<xsl:apply-templates select="chapter"/>
```

```
</xsl:template>

<xsl:template match="appendix">
<xsl:apply-templates select="apps"/>
<xsl:apply-templates select="abbrevations"/>
</xsl:template>

<xsl:template match="chapter">
<fo:block font-size="20pt" font-family="serif" line-height="30pt"
break-before="page" font-weight="bold" keep-with-next="always">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/> <xsl:text> </xsl:text>
<xsl:apply-templates select="title"/>
</fo:block>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subchapter"/>
</xsl:template>

<xsl:template match="apps/chapter">
<fo:block break-after="page" keep-with-next="always">
<fo:block font-size="20pt" font-family="serif" line-height="30pt"
font-weight="bold">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter" format="A"/> <xsl:text> </xsl:text>
<xsl:apply-templates select="title"/>
</fo:block>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subchapter"/>
</fo:block>
</xsl:template>

<xsl:template match="paragraph">
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
language="de" text-align="justify" text-align-last="start" space-
before="0mm" space-after.optimum="2mm" space-after.minimum="0mm"
text-indent="5mm" hyphenate="true" hyphenation-character="-" keep-
with-next="always">
<xsl:apply-templates/>
</fo:block>
</xsl:template>

<xsl:template match="subchapter">
<fo:block keep-with-next="always">
```

```
<fo:block font-size="18pt" font-family="serif" line-height="27pt"
space-before.minimum="0pt" space-before.optimum="6pt" font-
weight="bold">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/>.<xsl:number count="subchapter"/>
<xsl:text> </xsl:text> <xsl:apply-templates select="title"/>
</fo:block>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="section"/>
</fo:block>
</xsl:template>

<xsl:template match="apps/*/subchapter">
<fo:block keep-with-next="always">
<fo:block font-size="18pt" font-family="serif" line-height="27pt"
space-before.minimum="0pt" space-before.optimum="6pt" font-
weight="bold">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/> <xsl:text> </xsl:text> <xsl:apply-templates se-
lect="title"/>
</fo:block>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="section"/>
</fo:block>
</xsl:template>

<xsl:template match="section">
<fo:block keep-with-next="always">
<fo:block font-size="16pt" font-family="serif" line-height="24pt"
space-before.minimum="0pt" space-before.optimum="4pt" font-
weight="bold">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/> <xsl:text>
</xsl:text> <xsl:apply-templates select="title"/>
</fo:block>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subsection"/>
</fo:block>
```

```
</xsl:template>

<xsl:template match="apps/*/*/section">
<fo:block keep-with-next="always">
<fo:block font-size="16pt" font-family="serif" line-height="24pt"
space-before.minimum="0pt" space-before.optimum="4pt" font-
weight="bold">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/> <xsl:text>
</xsl:text> <xsl:apply-templates select="title"/>
</fo:block>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subsection"/>
</fo:block>
</xsl:template>

<xsl:template match="subsection">
<fo:block font-size="14pt" font-family="serif" line-height="21pt"
space-before.minimum="0pt" space-before.optimum="2pt" font-
weight="bold" keep-with-next="always">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/> <xsl:text> </xsl:text> <xsl:apply-templates se-
lect="title"/>
</fo:block>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subsubsection"/>
</xsl:template>

<xsl:template match="apps/*/*/*/subsection">
<fo:block font-size="14pt" font-family="serif" line-height="21pt"
space-before.minimum="0pt" space-before.optimum="2pt" font-
weight="bold" keep-with-next="always">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/> <xsl:text> </xsl:text> <xsl:apply-templates se-
lect="title"/>
```

```
</fo:block>
<xsl:apply-templates select="paragraph"/>
<xsl:apply-templates select="subsubsection"/>
</xsl:template>

<xsl:template match="subsubsection">
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
space-before.minimum="0pt" font-weight="bold" keep-
with-next="always">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/>.<xsl:number count="subsubsection"/> <xsl:text>
</xsl:text> <xsl:apply-templates select="title"/>
</fo:block>
<xsl:apply-templates select="paragraph"/>
</xsl:template>

<xsl:template match="apps/*/*/*/*/subsubsection">
<fo:block font-size="12pt" font-family="serif" line-height="18pt"
space-before.minimum="0pt" font-weight="bold" keep-
with-next="always">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:number count="chapter" format="A"/>.<xsl:number
count="subchapter"/>.<xsl:number count="section"/>.<xsl:number
count="subsection"/>.<xsl:number count="subsubsection"/> <xsl:text>
</xsl:text> <xsl:apply-templates select="title"/>
</fo:block>
<xsl:apply-templates select="paragraph"/>
</xsl:template>

<xsl:template match="asset">
<xsl:choose>
<xsl:when test="@type='image'">
<fo:block text-align="end">
<fo:external-graphic keep-with-next="always">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:attribute name="src"> <xsl:value-of select="@source"/>
</xsl:attribute>
<xsl:attribute name="width"> <xsl:value-of select="@width_in_mm"/>
```

```
xsl:text>mm</xsl:text> </xsl:attribute>
<xsl:attribute name="height"> <xsl:value-of select="@height_in_mm"/>
<xsl:text>mm</xsl:text> </xsl:attribute>
</fo:external-graphic>
<fo:block font-size="10pt" font-style="italic" text-align="end"
keep-with-previous="always">
<fo:inline font-weight="bold">
<xsl:text>Abbildung </xsl:text> <xsl:number count="asset" le-
vel="any"/> <xsl:text>: </xsl:text></fo:inline>
<fo:inline><xsl:value-of select="@alt"/></fo:inline>
</fo:block>
</fo:block>
</xsl:when>
<xsl:when test="@type='audio'">
<fo:external-graphic src="images/audio.gif"/>
</xsl:when>
<xsl:when test="@type='video'">
<fo:external-graphic src="images/video.gif"/>
</xsl:when>
</xsl:choose>
</xsl:template>

<xsl:template match="bibliography">
<fo:block keep-with-next="always">
<fo:block font-size="20pt" font-family="serif" line-height="30pt"
font-weight="bold">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:value-of select="title"/>
</fo:block>
<xsl:for-each select="biblio_item">
<xsl:sort select="@designator"/>
<fo:block font-size="12pt" font-family="serif" line-height="16pt"
space-after="12pt" keep-together="always">
<fo:block font-weight="bold" space-after="2pt">
<xsl:value-of select="@designator"/>
</fo:block>
<fo:block>
<xsl:apply-templates select="authors"/>
<xsl:text>.</xsl:text>
</fo:block>
<fo:block>
```

```
<fo:inline font-weight="bold">
<xsl:apply-templates select="title"/>
</fo:inline>
<xsl:text>. </xsl:text>
<xsl:if test="subtitle">
<fo:inline font-style="italic">
<xsl:apply-templates select="subtitle"/>
<xsl:text>. </xsl:text>
</fo:inline>
</xsl:if>
</fo:block>
<fo:block>
<xsl:value-of select="publisher"/>
</fo:block>
<xsl:apply-templates select="address"/>
<xsl:apply-templates select="date"/>
</fo:block>
</xsl:for-each>
</fo:block>
</xsl:template>

<xsl:template match="abbrevations">
<fo:block keep-with-next="always">
<fo:block font-size="20pt" font-family="serif" line-height="30pt"
font-weight="bold">
<xsl:attribute name="id"> <xsl:value-of select="@id"/>
</xsl:attribute>
<xsl:value-of select="title"/>
</fo:block>
<fo:table>
<fo:table-column column-width="35mm"/>
<fo:table-column column-width="125mm"/>
<fo:table-body font-size="12pt" font-family="serif">
<xsl:for-each select="abb_item">
<xsl:sort select="abbrevation/@long_form"/>
<fo:table-row line-height="16pt">
<fo:table-cell>
<fo:block text-align="start"> <xsl:value-of select="abbrevation"/>
</fo:block>
</fo:table-cell>
<fo:table-cell>
<fo:block text-align="start">
```

```
<xsl:apply-templates select="longform"/>
</fo:block>
</fo:table-cell>
</fo:table-row>
</xsl:for-each>
</fo:table-body>
</fo:table>
</fo:block>
</xsl:template>

<xsl:template match="list">
<fo:block font-size="12pt" font-family="serif" line-height="18pt">
<xsl:choose>
<xsl:when test="@type='ordered'">
<xsl:for-each select="title">
<fo:block>
<xsl:apply-templates/>
</fo:block>
</xsl:for-each>
<fo:list-block>
<xsl:for-each select="item">
<fo:list-item>
<fo:list-item-label end-indent="label-end()">
<fo:block><xsl:number count="item"/> <xsl:text>.</xsl:text>
</fo:block>
</fo:list-item-label>
<fo:list-item-body start-indent="body-start()">
<fo:block font-size="12pt" font-family="serif" line-height="18pt">
<xsl:apply-templates/>
</fo:block>
</fo:list-item-body>
</fo:list-item>
</xsl:for-each>
</fo:list-block>
</xsl:when>
<xsl:otherwise>
<xsl:for-each select="title">
<fo:block>
<xsl:apply-templates/>
</fo:block>
</xsl:for-each>
<fo:list-block>
```

```
<xsl:for-each select="item">
<fo:list-item>
<fo:list-item-label end-indent="label-end()">
<fo:block>&#8226;</fo:block>
</fo:list-item-label>
<fo:list-item-body start-indent="body-start()">
<fo:block>
<xsl:apply-templates/>
</fo:block>
</fo:list-item-body>
</fo:list-item>
</xsl:for-each>
</fo:list-block>
</xsl:otherwise>
</xsl:choose>
</fo:block>
</xsl:template>

<xsl:template match="reference">
<fo:footnote>
<fo:inline font-size="8pt" vertical-align="super">
<xsl:number count="reference" level="any"/>
</fo:inline>
<fo:footnote-body>
<fo:block line-height="13pt" start-indent="0mm">
<fo:inline vertical-align="super" font-size="8pt">
<xsl:number count="reference" level="any"/>
</fo:inline>
<fo:inline font-size="10pt">
<xsl:value-of select="@refered_to"/>
</fo:inline>
<fo:inline font-size="10pt">
<xsl:apply-templates select="reftext"/>
</fo:inline>
</fo:block>
</fo:footnote-body>
</fo:footnote>
</xsl:template>

<xsl:template match="link">
<fo:inline> <xsl:value-of select="."/> <fo:inline>
<fo:inline font-style="italic">
<xsl:text>(</xsl:text> <xsl:value-of select="@xlink:href"/>
```

```
xsl:text>)</xsl:text>
</fo:inline>
</xsl:template>

<xsl:template match="code">
<fo:inline font-family="monospace" font-size="11pt">
<xsl:value-of select="."/>
</fo:inline>
</xsl:template>

<xsl:template match="codeline">
<fo:block font-family="monospace" start-indent="5mm" end-in-
dent="5mm" background-color="#cccccc" text-align="start" font-
size="11pt">
<xsl:value-of select="."/>
</fo:block>
</xsl:template>

<xsl:template match="//appendix/*/*/*/*/*/codeline">
<fo:block font-family="monospace" text-align="start" font-
size="11pt">
<xsl:value-of select="."/>
</fo:block>
</xsl:template>

<xsl:template match="address">
<xsl:if test="street">
<fo:block>
<xsl:value-of select="street"/>
<xsl:text> </xsl:text>
<xsl:value-of select="nr"/>
</fo:block>
</xsl:if>
<fo:block>
<xsl:for-each select="postal_code">
<xsl:value-of select="."/>
<xsl:text> </xsl:text>
</xsl:for-each>
<xsl:for-each select="town">
<xsl:value-of select="."/>
<xsl:if test="position()!=last()"> <xsl:text>, </xsl:text> </xsl:if>
</xsl:for-each>
</fo:block>
<xsl:for-each select="country">
```

```
<fo:inline><xsl:value-of select="."/> <xsl:text>, </xsl:text>
</fo:inline>
</xsl:for-each>
</xsl:template>

<xsl:template match="date">
<fo:inline>
<xsl:for-each select="day">
<xsl:value-of select="."/>
<xsl:text>.</xsl:text>
</xsl:for-each>
<xsl:for-each select="month">
<xsl:value-of select="."/>
<xsl:text>.</xsl:text>
</xsl:for-each>
<xsl:value-of select="year"/>
</fo:inline>
</xsl:template>

<xsl:template match="authors">
<xsl:for-each select="author">
<fo:inline>
<xsl:choose>
<xsl:when test="name">
<xsl:value-of select="name/surname"/>
<xsl:text>, </xsl:text>
<xsl:for-each select="name/first_name">
<xsl:value-of select="."/>
<xsl:if test="position()!=last()"> <xsl:text> </xsl:text> </xsl:if>
</xsl:for-each>
<xsl:if test="position()!=last()">; </xsl:if>
</xsl:when>
<xsl:when test="organization">
<xsl:apply-templates select="organization"/>
</xsl:when>
</xsl:choose>
</fo:inline>
</xsl:for-each>
</xsl:template>

<xsl:template match="emph">
<fo:inline font-weight="bold">
<xsl:apply-templates/>
```

```
</fo:inline>
</xsl:template>

<xsl:template match="strong">
<fo:inline font-weight="bold" font-style="italic">
<xsl:apply-templates/>
</fo:inline>
</xsl:template>

<xsl:template match="quotation">
<fo:inline font-style="italic">
<xsl:apply-templates/>
</fo:inline>
</xsl:template>

<xsl:template match="comment">
</xsl:template>

</xsl:stylesheet>
```

Abkürzungsverzeichnis

ASCII	American Standard of Code for Information Interchange *(http://czyborra.com/charsets/iso8859.html)*
B2B	Business to Business
CSS	Cascading Style Sheets *(http://www.w3.org/TR/REC-CSS1)*
DOM	Document Object Model *(http://www.w3.org/DOM)*
DSSSL	Document Style and Semantic Specification Language *(ftp://ftp.ornl.gov/pub/sgml/WG8/DSSSL/dsssl96b.pdf)*
DTD	Document Type Definition
E-Commerce	Electronic Commerce
FOSI	Formatting Output Specification Instance
GIF	Graphics Interchange Format
GML	Generalized Markup Language
GNU	GNU's not Unix *(http://www.gnu.org)*
HTML	HyperText Markup Language *(http://www.w3.org/TR/html401)*
HyTime	Hypermedia and Time-based Structuring Language *(http://www.ornl.gov/sgml/wg8/docs/n1920/html/n1920.html)*
IDE	Integrated Developer Environment
ISO	International Standardization Organization *(http://www.iso.ch/iso/en/ISOOnline.openerpage)*
JPEG	Joint Photographic Experts Group *(http://www.jpeg.org/)*
MathML	Mathematical Markup Language *(http://www.w3.org/Math)*
PDA	Personal Digital Assistant
PDF	Portable Document Format
RTF	Rich Text Format
SAX	Simple API for XML *(http://www.megginson.com/SAX/index.html)*
SDQL	Standard Document Query Language *(http://www.ornl.gov/sgml/wg8/docs/n1920/html/clause-7.11.html)*
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol *(http://www.w3.org/2000/xp/)*
SVG	Scalable Vector Graphics *(http://www.w3.org/Graphics/SVG)*
TEI	Text Encoding Initiative *(http://www.tei-c.org)*
URI	Unified Ressource Identifier *(http://www.w3.org/Addressing/)*
URL	Unified Ressource Locator *(http://www.w3.org/Addressing/)*
W3C	World Wide Web Consortium *(http://www.w3.org)*
WAP	Wireless Application Protocol *(http://www.wapforum.org/)*
WML	Wireless Markup Language
WYSIWYG	What You See Is What You Get
XHTML	eXtensible HyperText Markup Language *(http://www.w3.org/MarkUp/)*
XLink	XML Linking Language *(http://www.w3.org/XML/Linking)*
XML	eXtensible Markup Language *(http://www.w3.org/TR/REC-xml)*
XPath	XML Path Language *(http://www.w3.org/TR/xpath)*
XPointer	XML Pointer Language *(http://www.w3.org/XML/Linking)*

XSL	eXtensible Stylesheet Language *(http://www.w3.org/TR/xsl)*
XSLFO	eXtensible Stylesheet Language Formatting Objects *(http://www.w3.org/TR/xsl/slice6.html#fo-section)*
XSLT	eXtensible Stylesheet Language for Transformation *(http://www.w3.org/TR/xslt)*

Abbildungsverzeichnis

Abbildung 1:	Word-Dokument im Texteditor Emacs	5
Abbildung 2:	eine Word HTML-Datei	6
Abbildung 3:	ein LaTeX-Dokument	9
Abbildung 4:	XMetaL in der Ansicht "Tags On"	34
Abbildung 5:	Emacs im PSGML-Mode	37
Abbildung 6:	Verhalten bei einer Abkürzung	59

Literaturverzeichnis

BIBLE17
Herold, Eliotte Rusty.
The XML Bible *(http://www.ibiblio.org/xml/books/bible2/chapters/ch17.html)* *. Kapitel 17: XSL Transformations.*
16.10.2001

BIBLE18
Herold, Eliotte Rusty.
The XML Bible *(http://www.ibiblio.org/xml/books/bible2/chapters/ch18.html)* *. Kapitel 18: XSL Formatting Objects .*
16.10.2001

BOCK
Bock, Matthias.
Integration eines Strukturediteors mit einer SGML-Datenbank *(http://www.informatik.uni-bremen.de/~bock/da)* *. Diplomarbeit am Fachbereich 3 der Universität Bremen.*
1997

INFXML
Lobin, Henning.
Informationsmodellierung in XML und SGML.
Springer
Berlin, Heidelberg
Deutschland, 2000

LATEX
Detig, Christine.
Der LaTeX Wegweiser.
International Thomson Publishing GmbH
Bonn, Albany, Attenkirchen
1997

LXML
Ray, Erik T.
Learning XML.
O'Reilly & Assiciates
Sebastopol
USA, 2001

SGMLMIG
Travis, Brian E; Waldt, Dale C.

The SGML Implementation Guide. *A Blueprint for SGML Migration.*
Springer
Berlin, Heidelberg, New York, Barcelona, Budapest, Hong Kong, London, Milan, Paris, Tokyo
1996

TEIG
TEI *(http://www.tei-c.org)* .
TEI Guidelines P4 ***(http://www.tei-c.org/Guidelines2/index.html)*** .
16.10.2001

TEXWEB
Goossens, Michel; Rahtz, Sebastian.
The LaTeX Web Companion. *Integrating TeX, HTML, and XML.*
Addison-Wesley Longman
Massachusetts
1999

WEBXML
Knobloch, Manfred; Kopp, Matthias.
Web-Design mit XML. *Webseiten erstellen mit XML, XSL und Cascading Style Sheets.*
dpunkt.verlag
Heidelberg
Deutschland, 2001

XHTMLWD
W3C *(http://www.w3.org)* .
XHTML 1.0: The Extensible HyperText Markup Language (Second Edition) ***(http://www.w3.org/TR/2001/WD-xhtml1-20011004)*** . *A Reformulation of HTML 4 in XML 1.0W3C Working Draft 4 October 2001.*
22.10.2001

XMLPR
Behme, Henning; Mintert, Stefan.
XML in der Praxis ***(http://irb-www.informatik.uni-dortmund.de/~sm/aw/xml/msie)*** .
Professionelles Web-Publishing mit der Extensible Markup Language.
21.10.2001

XMLPRO
Martin, Didier; Birbeck, Mark; Kay, Michael; Loesgen, Brian; Pinnock, Jon; Livingstone, Steven; Stark, Peter; Williams, Kevin; Anderson, Richard; Mohr, Stephen; Baliles, David; Peat, Bruce; Ozu, Nikola.
Professional XML. *Covers W3C DOM, SAX, CSS, XSLT, DTDs, XML Schemas, XLink, XPointer, XPath, E-Commerce, BizTalk, B2B, SOAP, WAP, WML* .

Wrox Press Ltd.
UK, USA, 2000

XSLREC
W3C *(http://www.w3.org)* .
Extensible Stylesheet Language (XSL) Version 1.0 ***(http://www.w3.org/TR/xsl)*** . *W3C Recommendation 15 October 2001.*
22.10.2001

XSLTPR
Kay, Michael.
XSLT Programmer's Reference.
Wrox Press Ltd.
UK, USA, 2000

XSLTREC
W3C *(http://www.w3.org)* .
XSL Transformations (XSLT) Version 1.0 ***(http://www.w3.org/TR/xslt)*** . *W3C Recommendation 16 November 1999.*
22.10.2001

Erklärung

Ich versichere, dass ich diese Magisterarbeit einschließlich evtl. beigefügter Darstellungen selbstständig angefertigt und keine anderen Quellen und Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, habe ich in jedem einzelnen Fall unter genauer Angabe der Quelle als Entlehnung deutlich gemacht.

Bielefeld, 23. Oktober 2001