Mit XSLT von XML zu LaTeX – Ein Erfahrungsbericht

Mit einer Beschreibung von tbook 1.4

Torsten Bronger*

30. September 2002

Zusammenfassung

XML gilt als die Sprache der Zukunft für logische Auszeichnung von Dokumenten aller Art. Das logische Markup ist aber auch eine hervorstechende Eigenschaft von LaTeX. Grund genug für einen LaTeXer wie mich, die Brauchbarkeit von XML zu untersuchen. XML soll die Texte frei halten von allem, was nicht zum Inhalt gehört, inklusive der teils aufwendigen Präambeln. Vor allem aber verspricht es Abhilfe beim vielleicht größten Manko von LaTeX: der Umwandlung in andere beliebte Formate wie HTML oder RTF.

Inhaltsverzeichnis

1 Warum XML? 4
1.1 Textdatei . . . 4
1.2 Trennung von Layout und Inhalt . . . 5

2 XML-Anwendungen 5
2.1 DocBook und TEI . . . 5
2.1.1 Vergleich mit LaTeX . . . 6
2.2 GELLMU . . . 7
2.3 Da war doch noch was . . . `qwertz`? . . . 8
2.4 LaTeX2HTML und TeX4ht . . . 8
2.5 Fazit: eine eigene DTD . . . 8

3 Das ganze zum Leben erwecken: XSL 9
3.1 Erst die Transformation: XSLT . . . 9
3.2 Dann die Formatierung: FO bzw. CSS . . . 9
3.3 Schicker Ausdruck: Mit XSLT direkt zu LaTeX . . . 10
3.4 Von Ordnung zum Chaos . . . 10

4 Okay, was ist also zu tun? 12

5 Die Document Type Definition DTD 12
5.1 DTD-Design-Strategie . . . 12
5.2 XML Schema . . . 13

*`mailto:bronger@physik.rwth-aachen.de`

6 Das Selbstgebackene **13**
6.1 tbook 13
6.1.1 Anmerkungen zum Projekt tbook 14
6.1.2 Homepage von tbook und die Dateien 15
6.2 Typischer Entwicklungszyklus 16
6.3 Die Umwandlungsskripte 16
6.4 Die Batchdateien 17
6.4.1 Gegenseitige Abhängigkeiten der Batchdateien 18
6.5 Bereitstellung von Grafiken 19
6.5.1 Overlay-Grafiken 19
6.6 Editor 19
6.6.1 Erfahrung mit dem Emacs 20
6.6.2 Emacs mit UTF-8 20
6.7 Besonderheiten von tbook-Dateien 20
6.8 XHTML und die Browser 21
6.8.1 Das Problem mit älteren Browsern 21
6.8.2 Der richtige Content-Type 22
6.9 Gutes altes HTML 4 22
6.9.1 HTML-4-Dateien erzeugen 22
6.9.2 Gleichungen als Bitmaps 23
6.10 Was ist mit Altlasten-Dokumenten? 24
6.11 Individuelle Konfigurierung 24
6.11.1 Cascading Style Sheets (CSS2) 24
6.11.2 LaTeX-cfg-Datei 24
6.11.3 LaTeX-sty-Datei 24
6.11.4 XSLT-Parameter 25
6.11.5 Totale Freiheit 25
6.12 Farbe 27

7 Überblick über Elemente und Attribute **28**
7.1 Elemente auf obersten Ebenen 28
7.2 Teile, Kapitel und Abschnitte 30
7.3 Absätze 32
7.4 Font-Formatierung 33
7.5 Querverweise 33
7.6 Einfache Mathematik 35
7.7 Sonstiges 37
7.8 Zitiertes und Verbatimes 39
7.9 Listen 40
7.10 Gleitumgebungen und Grafiken 41
7.11 Tabellen 44
7.12 Elemente der Frontmatter 46
7.13 Literaturverzeichnis 49
7.14 Stichwortverzeichnis 49
7.15 Brief-Elemente 50

8 Interna von tbook **52**
8.1 Dokumentation des Codes 52
8.1.1 DTX-Dateien 52
8.1.2 CWEB 52

8.2 Installation von CWEB . . . 52
8.3 Installation von tbook . . . 52
8.4 Installation von jpeg2ps . . . 52
8.5 Installation vom Saxon . . . 53
8.6 Installation von xîndy unter Linux . . . 53
8.7 Zum Stichwortprozessor xîndy . . . 54
8.8 tbook und Windows . . . 55
8.9 XSLT-Transformation . . . 55
8.10 PDF-Erzeugung . . . 56
8.11 Umwandlung nach DocBook . . . 56
8.11.1 RTF-Erzeugung . . . 57
8.12 Ein Glossar . . . 58
8.13 BIBTEX – für LATEX und HTML . . . 58
8.13.1 Möglichkeit Eins: XML . . . 58
8.13.2 Möglichkeit Zwei: BIBTEX . . . 58
8.14 Die LATEX-Installation . . . 59
8.15 Generierte Dateien . . . 59

9 Spezielle Probleme mit XML und XSLT 59
9.1 Das Problem des Unicodes in LATEX . . . 59
9.1.1 Mögliche Auswege . . . 61
9.1.2 Ein zwischengeschalteter Unicode-Filter . . . 61
9.1.3 Unicode-Filter `tbrplent` . . . 62
9.2 Die Behandlung von Whitespace . . . 63
9.2.1 Leerzeilen im Fließtext . . . 63
9.3 XML-Syntax vs. Text-Syntax . . . 64
9.3.1 Das `<multipar>`-Element . . . 64
9.3.2 `<m>`, `<dm>`, `<ch>` und `<unit>` . . . 64
9.4 Schwächen von XSLT . . . 66
9.5 Aufteilung auf Unterverzeichnisse: XML Catalog . . . 67

10 Das Problem der Sprachen 68
10.1 Zur Implementation . . . 68
10.1.1 LATEX . . . 68
10.1.2 Andere Ausgaben . . . 69
10.1.3 Sprachabhängiger LATEX-Code . . . 69
10.2 Dezimaltrennzeichen . . . 69
10.3 Das Stichwortverzeichnis . . . 69

11 Das Modul für Adressen 70
11.1 Bestehende Adreß-DTDs . . . 70
11.2 Fazit: eine eigene DTD . . . 70

12 MathML 71
12.1 Meine Implementierung . . . 72
12.1.1 `underbrace` . . . 72
12.1.2 `<mtext>` . . . 72
12.1.3 Gleichungs-Array . . . 73
12.1.4 tbook und MathML mischen . . . 73
12.1.5 Gleichungen durchnumerieren/Labels setzen . . . 74
12.1.6 Wie gut werden Gleichungen? . . . 75

12.2 Einbettung in die DTD 75
12.3 Schwächen von MathML 75

13 Ausblick **77**
13.1 Für meine Anwendung 77
13.2 Zukunft von XML – meine Sicht 78
13.3 LyX 79

14 Hat sich's gelohnt? **80**

15 Glossar **80**

1 Warum XML?

Tja, warum. Ich gehe in diesem Erfahrungsbericht von Grundwissen über XML aus, daher wiederhole ich hier nicht den *ganzen* Sermon für oder gegen XML. Ein Teil davon wird aber unumgänglich sein. Die offizielle Spezifikation findet man in XML-W3C [2000] oder in der deutschen Übersetzung unter XML-Deutsch [2000]. H. Partl, der allen LaTeXern ein Begriff sein dürfte, hat eine schöne Kurzeinführung geschrieben [Partl, 2000].

In wenigen Worten erwarte ich mir von XML folgendes:

- Einfache Eingabe in Form einer übersichtlichen Textdatei.
- Völlige Trennung von Inhalt und Layout.
- Vollautomatische Umwandlung in alle wichtigen Dokumentformate (RTF, HTML, etc), ohne nennenswerte Qualitätseinbußen.
- Ausdruck in einer Qualität, die ich vom „direkten" LaTeX gewohnt bin.
- Bestmögliche Darstellung meiner XML-Dateien in Web-Browsern.
- Das alles muß auch klappen für Tabellen, Bilder und Formeln. Außerdem möchte ich möglichst in allen Formaten Inhaltsverzeichnis, Index und Literaturverzeichnis haben.

1.1 Textdatei

Ich muß LaTeXern nicht groß und breit erzählen, warum das ein Vorteil ist. Man kann solche Dateien halt mit praktisch jedem Editor unter jedem Betriebssystem editieren, und gerade für XML gibt es viele Editoren oder Editoren-Plug-Ins, die die Arbeit noch zusätzlich erleichtern. Die weitere Verarbeitung solcher Dateien (z. B. Unicodes ersetzen o. ä.) ist recht simpel.

Es ist allerdings eine interessante Frage, inwieweit XML überhaupt dazu gedacht ist, von einem Menschen direkt eingegeben zu werden, ob es also nicht stets von einem Programm erzeugt werden sollte. Es würde ja auch niemand auf die Idee kommen, Postscript-Dateien direkt einzugeben. Insbesondere jedoch, wenn man eine eigene XML-Variante benutzt, ist man zur direkten Eingabe gezwungen.

Der Aufwand, eine XML-Datei selbst zu schreiben, liegt in der Größenordnung von LaTeX. Einiges ist umständlicher und einiges ist leichter. Der größte Batzen ist sowieso der Fließtext, und er ist in beiden Fällen identisch. In diesem Zusammenhang: Der XML-Modus des GNU Emacs unterstützt den Autor hervorragend bei der Arbeit, siehe Abschnitt 6.6.1 auf Seite 20. Siehe dazu auch den Kommentar zu LyX, Abschnitt 13.3 auf Seite 79.

1.2 Trennung von Layout und Inhalt

Wenn man von LaTeX kommt, hat man ohnehin eine etwas entspanntere Beziehung zu Layoutfragen, da sich das Programm mit seinen Standard-Einstellungen um das meiste kümmert.

Ich gehe mit meiner Forderung aber noch einen Schritt weiter. Ich will mich so gut wie überhaupt nicht mehr mit Äußerlichkeiten beschäftigen. Ich habe in vielen Jahren mein persönliches Layout gefunden und bin müde geworden, daran immer wieder herumzuspielen. Andere mögen von Anfang an bei ihren Texten sich nicht groß um das Layout gesorgt haben, aber wie dem auch sei:

Mit XML gibt man nur noch den *Inhalt* und die *logische Struktur* ein, und überläßt das Layout vollständig dem Postprozessor. Das macht die Sache wesentlich übersichtlicher und flexibler. Man wird nicht abgelenkt, kann weniger typographischen Unsinn anstellen und ist in der Lage, völlig unterschiedliches Layout für z. B. unterschiedliche Ausgabemedien anzuwenden.

(Logisches Markup bedeutet nicht, daß die persönliche Note des Layouts verlorengeht. Sie wird nur anders definiert.)

Wenn man aufpaßt, hat man auch mit LaTeX eine recht gute Trennung von Inhalt und Layout. Meine Erfahrung ist aber, daß insbesondere bei größeren Projekten und gegen Ende der Arbeit diese Trennung mehr und mehr aufweicht. Außerdem kann bereits jedes `\newcommand` für eventuelle Konvertierer das Todesurteil sein. Was bei LaTeX guter Stil ist, ist bei XML Zwang.

Bevor die Ziegel fliegen . . . erinnere ich nochmal daran, daß das hier ein *Erfahrungsbericht* ist. Ich bin ein XML-Neuling, der sich umgeschaut hat und es höchstens in der Benutzung von XSLT zu einer bescheidenen Meisterschaft gebracht hat. Was hier also folgt, sind meine Eindrücke. Wenn ich irgendwo Falsches erzähle oder offensichtlich unfair bin, mag man mich korrigieren, dann werde ich den Text korrigieren.[1]

Wenn ich aber mit einigen Aspekten von XML hart ins Gericht gehe, ist das beabsichtigt. Ich hoffe, dabei nicht überheblich zu wirken; allerdings bin ich lange und intensiv genug dabei, um zu wissen, was ein Autor braucht. Und das fordere ich rücksichtslos auch von der XML-Welt.

2 XML-Anwendungen

2.1 DocBook und TEI

Es gibt zwei große Allzweck-XML-Anwendungen: DocBook [DocBook, 2002] und TEI. TEI habe ich nicht aktiv benutzt; für TEI bzw. TEI Lite [TEI Lite, 2001] wird jedoch ähnliches wie für DocBook gelten.

DocBook ist gigantisch groß (knapp 400 Elemente). Es deckt LaTeXs Dokumentklassen außer `letter` ganz gut ab. Die Größe von DocBook ist für den Autor zwar ärgerlich, aber allein kein großes Problem, da er ja nicht alle Möglichkeiten nutzen muß,[2] sie erschwert jedoch erheblich die weitere Verarbeitung von DocBook-Dokumenten, insbesondere die Umwandlung nach LaTeX. Es gibt zwar ein System, das DocBook nach high-level LaTeX umwandelt [DB2LaTeX, 2002] und damit dasselbe tut wie das, was ich auch angestrebt habe, allerdings ist die Qualität alles andere als ausreichend.

Dieses Qualitätsproblem gilt übrigens ganz generell. Mit XML erzeugte Texte sehen häufig wie dezente Staubsauger-Manuals mit Ligaturen aus. Es fehlt einfach jeglicher Pfiff. Die individuelle Anpassung ist für einige triviale Dinge wie Seitenränder nicht schwer, aber viel Macht hat

[1] Wobei ich sicherlich nicht vorhabe, diesen Erfahrungsbericht nun über Monate auf dem neuesten Stand zu halten.

[2] obwohl DocBook häufig zu unnötig tiefen Verschachtelungen zwingt

man nicht. Insbesondere wird jegliche Formatierung über Stylesheets abgewickelt, und Stylesheet-Sprachen haben nicht die Flexibilität und Präzision einer LaTeX-Stil-Datei. Man kann ein paar *Werte* (Dimensionen, Fontnamen) verändern; die *Strukturen* sind jedoch vorgegeben.[3]

Stichwortverzeichnis oder Literaturverzeichnis sind, obgleich prinzipiell möglich, gewisse Abenteuer. Die Art, wie bisher Formeln in XML gehandhabt werden, ist kaum praktikabel: Entweder man benutzt LaTeX-Syntax, die natürlich die Umwandlung nach HTML oder RTF nahezu unmöglich macht, oder man benutzt MathML und schreibt sich Schwielen.

Woran liegt das? Diejenigen, die heute schon als Autoren SGML oder XML einsetzen, also quasi als Ersatz für LaTeX oder Word, sind noch recht wenige Exoten. Diese Gruppe besteht zum größten Teil aus Leuten, die das logische Markup benötigen, z. B. um eine HTML- und eine Plain-Text-Fassung zu erzeugen. DocBook- oder TEI-Dokumente sind in gewissem Sinne Datenbanken, und für die meisten Zwecke, zu denen sie eingesetzt werden, ist das auch hilfreich und/oder notwenig.

Dennoch ist sind DocBook DTD und die assoziierten Tools sehr ausgereift und stabil. Es stecken viele Jahre Erfahrung durch tausende von Dokumenten in dem Ding. Die Unterstützung mit verschiedenen freien Tools ist zwar nicht weltbewegend, aber beachtlich. Nicht weltbewegend, weil SGML/XML für Bücher u. ä. immer noch ein Schattendasein führt. Aber innerhalb der SGML/XML-Welt dominiert DocBook die Dokumenten-Erzeugung.

Fazit: *DocBook ist hervorragend geeignet für beliebig große Projekte, die im Web und auf dem Drucker verfügbar sein sollen, keine Ansprüche an die Ästhetik stellen und im weitesten Sinne aus dem Bereich der technischen Dokumentation stammen. Andere Textsorten sind kaum ohne Mißbrauch von DocBook und Frickeln möglich.*

2.1.1 Vergleich mit LaTeX

Ich habe bislang Briefe, Seminararbeiten, etliche Versuchsprotokolle, eine Diplomarbeit, ein bißchen Fiktionales und viele unkategorisierbare Kleinst-Texte verfaßt. Auf anderen Festplatten mag es ähnlich aussehen. Welchen Wert hat dann der Datenbank-artige Ansatz von DocBook? Zwar kann man jeden Text als Datenbank im weitesten Sinne auffassen, aber

1. schießt man damit mit Kanonen auf Spatzen und
2. kann man sowieso nie allen Textsorten gerecht werden.

Der Ansatz von LaTeX ist, daß Textelemente, die gleich formatiert werden, auch gleich benannt werden. Während DocBook sein logisches Markup streng auf allen Ebenen durchsetzt, weicht LaTeX es im Kleinen, d. h. auf Absatz-Ebene, auf und erlaubt Sonderwünsche zur Auszeichnung. (Damit meine ich übrigens nicht `\selectfont`-Befehle o. ä.)

Ich halte das für überlegen für den universellen Einsatz. Wie schon erwähnt, hat DocBook knapp 400 Elemente, ich glaube aber, daß ein XML-Format à la LaTeX mit einem Fünftel davon auch auskommt.

Grundsätzlich schließt also XML Komfort, Funktionalität und Typographie nicht aus, nur muß man sich in der gegenwärtigen Situation überlegen, was man persönlich braucht, und bestehende Systeme daran anpassen oder ein eigenes entwickeln. Solche Robinson-Crusoe-Lösungen sind zwar schade, aber der große strukturelle Informationsgehalt, den ein XML-Dokument aufweist, macht verlustarme Trafos in andere Formate, z. B. DocBook, möglich. Diese Trafos sind auch im allgemeinen sehr leicht und rasch zu schreiben. Man ist also sozusagen ein Robinson mit Yacht in der Lagune. `;-)`

[3] Es sei denn, natürlich, man programmiert die Stylesheets um.

2.2 GELLMU

Dies ist in meinen Augen der beste Ansatz, den es zur Zeit in der XML-Welt gibt. Bei [GELLMU, 2002] handelt es sich um ein Paket von Umwandlungsprogrammen (Perl und Emacs Lisp soweit ich es sehe), das ein spezielles Dateiformat, glm, umwandelt in HTML, SGML, XML und LaTeX. Das Format `glm` ist dabei fast LaTeX.

Noch mal das Problem: Normalerweise kann man sich ja in LaTeX beliebig daneben benehmen, d. h. man kann beliebig visuelles und logisches Markup vermischen, interne Befehle umdefinieren, `\specials` absetzen und Catcodes ändern. Am Ende muß das, was herauskommt, mit unser aller Vorstellung von LaTeX nichts mehr zu tun haben. Ein köstliches Beispiel dafür ist von David Carlisle kodiert worden:[4]

```
\let~\catcode~‘76~‘A13~‘F1~‘j00~‘P2jdefA71F~‘7113jdefPALLF
PA’’FwPA;;FPAZZFLaLPA//71F71iPAHHFLPAzzFenPASSFthP;A$$FevP
A@@FfPARR717273F737271P;ADDFRgniPAWW71FPATTFvePA**FstRsamP
AGGFRruoPAqq71.72.F717271PAYY7172F727171PA??Fi*LmPA&&71jfi
Fjfi71PAVVFjbigskipRPWGAUU71727374 75,76Fjpar71727375Djifx
:76jelse&U76jfiPLAKK7172F71l7271PAXX71FVLnOSeL71SLRyadR@oL
RrhC?yLRurtKFeLPFovPgaTLtReRomL;PABB71 72,73:Fjif.73.jelse
B73:jfiXF71PU71 72,73:PWs;AMM71F71diPAJJFRdriPAQQFRsreLPAI
I71Fo71dPA!!FRgiePBt’el@ lTLqdrYmu.Q.,Ke;vz vzLqpip.Q.,tz;
;Lql.IrsZ.eap,qn.i. i.eLlMaesLdRcna,;!;h htLqm.MRasZ.ilk,%
s$;z zLqs’.ansZ.Ymi,/sx ;LYegseZRyal,@i;@ TLRlogdLrDsW,@;G
LcYlaDLbJsW,SWXJW ree @rzchLhzsW,;WERcesInW qt.’oL.Rtrul;e
doTsW,Wk;Rri@stW aHAHHFndZPpqar.tridgeLinZpe.LtYer.W,:jbye
```

Nein, nein, das ist kein Hex-Dump, das ist ein gültiges TeX-Dokument, das man so, wie es da steht, TeX übergeben kann. Was man dann erhält, verrate ich hier nicht – diejenigen, die es noch nicht kennen, würden es mir sowieso nicht glauben. In LaTeX ist im Prinzip dieselbe Vergewaltigung möglich. Auch wenn das keiner voll ausnutzt, man kann sich jetzt wohl vorstellen, vor welchem Problem ein Umwandler LaTeX → Irgendwas steht.

GELLMU beschränkt die Syntax auf eine Untermenge von LaTeX. Man wird also zu einer gewissen Reinlichkeit und Disziplin gezwungen. Dafür wird man belohnt mit garantiert funktionierender Umwandlung in obige Formate. Diese Hybrid-Dateien sehen zwar etwas gewöhnungsbedürftig aus, aber der Großteil von LaTeXs Eingabesprache hat überlebt.

Letztenendes ist GELLMU also ein XML-Generierer mit LaTeX look-and-feel. Eine schöne Idee. Laut seines Autors soll GELLMU zu einem LaTeX-Dialekt avancieren, der

- transformierbar und
- reichhaltig ist, und
- von allen benutzt wird.

Bestehende LaTeX-Dokumente muß man anpassen; der Aufwand dafür hängt zwar vom Einzelfall ab, aber wenn man Glück hat, ist er recht gering.

Mich hat das dennoch nicht restlos überzeugt. Zunächst gilt auch hier wieder die alte Leier, daß es in der heutigen Form noch weit von der Praxistauglichkeit entfernt ist (Bilder, Tabellen, Formeln, etc). Vor allem aber muß sich GELLMU die Frage gefallen lassen, warum nicht direkt XML? Gut, XML fordert mehr Tipparbeit, aber für typische Dokumente ist die nicht groß genug, um als Argument zu dienen. Auch der Lernaufwand ist ungefähr derselbe.

XML ist ein internationaler Standard aus Beton mit einer strengen, sehr simplen Syntax und mit schon jetzt hunderten von Tools. GELLMU hat keine dieser Eigenschaften, die Eingabe ist

[4] Zu beziehen auf dem CTAN unter `/macros/plain/contrib/xii.tex`.

kürzer, sieht aber letztlich keinen Deut besser aus. Spezialeditoren, von denen XML enorm profitiert, sind für GELLMU kaum zu machen. Und die Gefahr, aufs falsche Pferd zu setzen, ist bei GELLMU viel folgenschwerer als bei irgendeiner[5] XML-Anwendung. Irgendwie ist GELLMU für mich das Ergebnis von Angst vor dem letzten, konsequenten Schritt.

Ich mag es also aus prinzipiellen Gründen nicht, die wahrscheinlich sehr persönlich sind. Für andere könnte es wohlmöglich eine echte Alternative für einen Zugang zu XML sein. Man sollte ein Auge darauf haben, vielleicht erreicht es ja irgendwann mal die „kritische Masse“.

2.3 Da war doch noch was … `qwertz`?

Ja, Tom Gordons `qwertz`-Projekt [qwertz, 1997]. Der Versuch, den Funktionsumfang von LaTeX mit SGML nachzubilden. Es war einige Zeit sehr beliebt, vor allem, weil damit das LinuxDoc-Projekt gearbeitet hat, das nun schon seit ein paar Jahren auf DocBook umgestiegen ist.

Schade eigentlich. Es sah ganz nett aus. Man sollte sich ruhig die Beispieldokumente, die auf der Homepage angegeben sind, anschauen. Sie zeigen, wie einfach SGML gegenüber XML war. In meinen Augen hat man bei XML zu viel der SGML-Flexibilität aufgegeben. Zum Beispiel ist es eine Wohltat, nicht jedes Tag schließen zu müssen, oder Attribute ohne Inhalt angeben zu können.

`qwertz` hat nie den Schritt in die XML-Welt vollzogen und konnte daher auch nie von den modernen Werkzeugen, wie z. B. XSLT-Prozessoren, profitieren. Außerdem sollen wohl Bilder und Formeln recht umständlich gewesen sein, weil sie extern verarbeitet wurden. Das Projekt ist seit fünf Jahren tot.

2.4 LaTeX2HTML und TeX4ht

Ich wäre nicht ehrlich, wenn ich diese beiden hervorragenden Systeme nicht erwähnte. Während LaTeX2HTML sich auf die Umwandlung nach HTML konzentriert, kann man mit TeX4ht auch in andere XML-Formate umwandeln. Ich werde jedoch hier nicht ins Detail gehen, man kann das alles am besten in [Goossens und Rahtz, 1999, Kap. 3 und 4] nachschlagen.

Denn hier soll es um XML gehen, und hinter diesen zwei Ansätzen steckt eine ganz andere Philosophie: Man benutzt weiterhin LaTeX als „Heimat“ und betrachtet alles andere, insbesondere die HTML-Umwandlung, als „Anhängsel“. Diese Sichtweise hat absolut ihre Daseinsberechtigung, und sie mag momentan auch noch die pragmatischste sein.

Ich jedoch will weg von LaTeXs Eingabesprache, die außer LaTeX keiner verstehen kann. Man muß sich immer im klaren darüber sein, daß, wenn man LaTeX wie auch immer in etwas anderes als PDF/PS umwandeln will, man entweder von Anfang an eine gewisse Disziplin einhält oder später richtig rödeln darf. Enthält das Dokument nicht-triviale Elemente (Formeln, Index etc), wird aus dem „oder“ ein „und“.

Zu TeX4ht siehe auch Abschnitt 6.10 auf Seite 24.

2.5 Fazit: eine eigene DTD

Für meine Zwecke ist es also am besten, eine eigene DTD, sprich eine eigene XML-Anwendung, zu basteln und sie mit den entsprechenden Transformationen und Formatierungen auszustatten. Die Arbeit ist nur ein wenig mehr, als jetzt in DocBook die nötigen umfangreichen Erweiterungen hinein zu wurschteln, außerdem habe ich so die volle Kontrolle.

Ich habe in meinen sieben Jahren mit LaTeX lediglich die Klassen Buch, Artikel und Brief benutzt. Meine DTD sollte also diese drei Dokumenttypen abdecken. Was die Funktionalität angeht,

[5] eventuell auch selbstgestrickten

ist das Ziel, die Möglichkeiten von LaTeX in XML nachzubilden. Mehrere hundert Tags wie in DocBook sind also unnötig.

Achtung: *Im folgenden verquicke ich zwei Dinge auf unschönste Weise: eine Beschreibung und Bewertung von XML, seinen Möglichkeiten und seinen Tools, und die Entwicklung einer eigenen XML-Anwendung. Ich wollte aber diese Verquickung nicht aufheben, um deutlich zu machen, daß meine XML-Anwendung in diesem Zusammenhang lediglich als Illustration dient.*

3 Das ganze zum Leben erwecken: XSL

Was nützen einem ein Haufen XML-Dateien auf der Festplatte? So erst mal nichts. XML wird erst durch Konvertierung in manierliche Dateiformate interessant. XSL ist die eleganteste Möglichkeit, XML auf diese Weise zum Leben zu erwecken. Dabei besteht XSL aus zwei Standards:

- *Transformation* hin zu einer anderen Dokumentstruktur mittels XSLT und dann
- *Formatierung* mittels sogenannter Formatierungsobjekte (FOs, XSL-W3C [2001]).

Damit die Namen klar sind: Die *Dateien*, um die es hier vor allem geht, haben alle die Endung `.xsl`, obwohl es XSLT-Dateien sind. *Zusätzlich können* sie auch Tags anderer XML-Sprachen enthalten, also z. B. (X)HTML, MathML, DocBook oder FO. Aus denen wird dann das Ergebnis der Transformation gebastelt.

3.1 Erst die Transformation: XSLT

XSLT (Spezifikation unter XSLT-W3C, 1999, kleiner Überblick unter Clark, 1999, gutes Buch ist Fitzgerald, 2001) ist grundsätzlich dafür gedacht, eine XML-Datei in eine andere XML-Datei umzustricken.

Sogenannte XSLT-Prozessoren lesen die XSLT-Datei und die Quell-XML-Datei und wandeln das in (X)HTML, XSL:FO oder DocBook um. Es gibt auch einen Plain-Text-Ausgabemodus von XSLT, der dann z. B. für LaTeX benutzt werden kann.

3.2 Dann die Formatierung: FO bzw. CSS

Die Konvertierung mittels XSLT in eine XSL:FO-Datei ist ein recht eleganter Ansatz, aus einer XML-Datei was vorzeigbares zu machen. FO-Prozessoren, z. B. [FOP, 2002] oder [PassiveTeX, 2002], wandeln die XSL:FO-Datei in PDF um. Im Prinzip erlaubt diese Vorgehensweise die automatische Konvertierung in fast alle Dokumentformate dieses Planeten, inklusive der Darstellung in Browsern. Leider ist das typographisch sehr dürftig, und Formeln müssen gesondert behandelt werden.

Die Unterstützung der FOs ist eher schlecht. Die Browser werden wahrscheinlich FOs nie interpretieren. Grund: Es ist recht schwer, FOs in die bestehenden Browser-Programme einzuweben. Außerdem scheint da eine Art Glaubenskrieg ausgebrochen zu sein, siehe dazu Lie [1999].

Für Browser muß man daher eine XSLT-Trafo nach HTML schreiben, und die Formatierung, wie bei HTML üblich, mit den alten Cascading Style Sheets CSS [CSS2-W3C, 1998; CSS3-W3C, 2002] bewerkstelligen.

Fazit: *XSLT ist für die Verarbeitung von XML-Dokumenten essentiell. Wenn man seine XML-Dokumente in Web-Browsern bewundern möchte, muß man eine XSLT-Trafo nach HTML+CSS2 schreiben.*

3.3 Schicker Ausdruck: Mit XSLT direkt zu LaTeX

Es ist sicher verführerisch, für seine XML-Anwendung Style Sheets (CSS, FO) oder eine Umwandlung nach HTML oder DocBook bereitzustellen.

Zusätzlich will ich jedoch auch eine direkte Transformation zu high-level LaTeX haben. In meinen Augen ist das die einzige Möglichkeit, mit XML auf dem Papier oder als PDF eine ausgezeichnete Qualität zu haben.

Außerdem brauche ich wenigstens eine Fassung meines Dokuments mit voll funktionstüchtigem Inhalts-, Literatur- und Stichwortverzeichnis. Dabei möchte ich, wie früher, die Leistungsfähigkeit von xindy (oder MakeIndex) und BIBTeX genießen können.

XSLT leistet dabei großartige Dienste, denn es kann statt XML- auch Plain-Text erzeugen. Man kann damit das XML-Dokument in ein LaTeX-Dokument umwandeln, das sogar einigermaßen lesbar bleibt. Natürlich sollte man das LaTeX-Dokument nicht mehr editieren, aber hat man es mal, kann man all die alten Tools zur Weiterverarbeitung bis zum Endprodukt (ps oder pdf) nutzen.

3.4 Von Ordnung zum Chaos

Die Abbildung 1 auf der nächsten Seite zeigt so eine Art Hierarchie der Dokument-Formate. Man nagle mich bitte nicht auf dem Ding fest, es ist mal als spontane Idee entstanden. Zumindest habe ich mir so selber klargemacht, was möglich ist, was schwer oder gar nicht möglich ist, welche Formate „viel wert sind" und welche Formate „wenig wert sind".

Die senkrechte Achse stellt die Entropie dar. Wahrscheinlich weniger das, was sich Informatiker darunter vorstellen, denn ich habe Entropie nur im thermodynamischen Sinn kennengelernt. Oben wenig, unten viel Entropie.

Wenig Entropie steht für Ordnung, viel Information über die Struktur, im übertragenden Sinn auch für geringes Alter. Viel Entropie bedeutet Chaos, wenig Struktur-Information und Annäherung an das Ende der Lebensdauer.

Das wichtigste ist: „Entropie" bedeutet „eine Richtung". Die Natur erlaubt nur die Richtung hin zu *größerer* Entropie.[6] Der umgekehrte Weg hin zu größerer Struktur wird von der Natur nur bei menschlichem Eingreifen geduldet. In der Abbildung funktionieren daher nur diejenigen Umwandlungen, die *nach unten* weisen, automatisch. Umwandlungen *nach oben* (TeX4ht habe ich beispielhaft dafür eingezeichnet) sind ohne menschliches Zutun nicht zu machen.

In diesem Sinne sind also die ganzen XML-Formate die edelsten aller Dokument-Formate, PDF und Postscript sind am wenigsten wert. Meine Lieblingsfirma Adobe möge mir verzeihen, aber es ist doch auch so: Mit einem Postscript kann man bloß noch eines machen – ausdrucken. LaTeX liegt offenbar gut im Rennen, aber nur, wenn man alles Feintuning und alle Frickeleien, insbesondere fast die ganze Präambel, außer acht läßt. Ich habe übrigens keine Ahnung, wie das FrameMaker-Format aussieht, daher habe ich es einfach mal bei RTF einsortiert.

Die Abbildung ist bitte nicht zu wörtlich zu nehmen, ich bin immer vom günstigsten/typischsten Fall ausgegangen. Es ist kein Problem, eines der edleren Dateiformate so zu mißhandeln, daß es nicht mehr mehr wert ist als ein Postscript. Besonders RTF und HTML sind sehr weite Begriffe. Die Suchmaschine Google[7] beispielsweise konvertiert PDF nach HTML, ein Unding gemäß meiner Abbildung. Aber wenn man sich die Quellcodes dieser HTMLs anschaut, erkennt man, daß hier HTML als Seitenbeschreibungssprache herhalten muß. Umgekehrt kann man niedere Dateiformate mit Tricks (z. B. Postscript mit Kommentaren) aufpeppen.

Die Abbildung gibt eine weitere schöne Motivation für die Benutzung von XML: TEI, DocBook und das eventuelle Eigengewächs thronen über allen anderen Formaten. Beim automati-

[6] Deshalb werden Schreibtische, wenn man die Dinge schleifen läßt, auch immer unordentlicher. Nein echt! Das ist der Grund!

[7] `http://www.google.com`

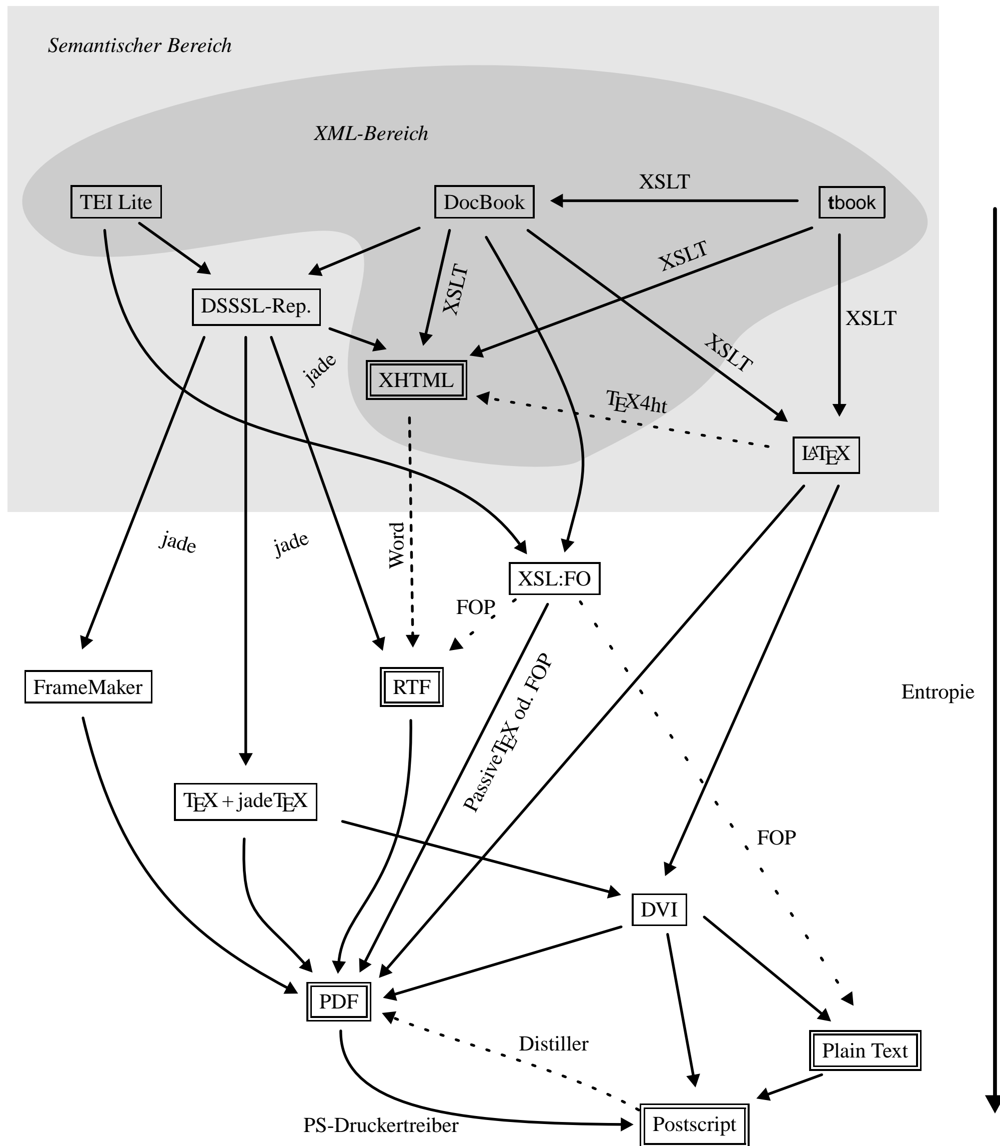


Abbildung 1: Überblick über die verschiedenen Textformate und die Möglichkeiten der Konvertierung untereinander. Die doppelt umrahmten Formate sind die typischerweise gewünschten Endformate. Einiges ist gestrichelt: Zwei der FOP-Wege sind noch nicht implementiert. Word kann mehr schlecht als recht HTML importieren. Und der Distiller verringert natürlich *nicht* die Entropie; er erzeugt entweder miesere PDFs oder geht von besseren Postscripts aus, genau weiß ich das nicht. Weitere Erläuterungen stehen auf der vorherigen Seite.

schen Herunterrutschen kommt man an so ziemlich allen Ausgabeformaten vorbei, die man sich wünscht.

Noch etwas anderes ist gut zu erkennen: XSLT ist *die* Möglichkeit schlechthin, um innerhalb der XML-Welt zu navigieren.

4 Okay, was ist also zu tun?

Folgendes muß ich also irgendwie verdrahten/programmieren, um eine eigene XML-Anwendung zum Laufen zu bekommen:

- Eine **DTD**-Datei, die die Syntax meines XML-Formats definiert. Diese DTD muß auch MathML einbinden (für die Formeln), außerdem die ganzen HTML-Entitäten (z. B. `…` für »...«).
- Einen Mechanismus, um Unicodes für LaTeX verständlich zu machen. Da gibt es einige Möglichkeiten, siehe Abschnitt 9.1 auf Seite 59.
- Eine **XSLT**-Datei für die Umwandlung XML → high-level LaTeX.
- Eine **XSLT**-Datei für die Umwandlung XML → HTML+CSS.

Damit wäre ich schon glücklich; gewissermaßen als Sahnehäubchen wäre aber noch das folgende interessant:

- Eine **XSLT**-Datei für die Umwandlung meine XML → DocBook, um das Gewissen zu beruhigen, weil man so den Anschluß an die große DocBook-Welt gesichert hat. (Die Yacht in der Lagune, ihr wißt schon.) Beispielsweise winken hier fix und fertige Umwandlungen nach RTF und FrameMaker.
- Ein auf TeX4ht basierendes System, um zumindest die harmloseren alten LaTeX-Dokumente in mein XML-Format umzuwandeln, und die anderen zum größten Teil umzuwandeln.
- Eine **XSL**-Datei für die Beschreibung des Aussehens meiner XML-Dokumente mittels FOs zur Umwandlung nach RTF, FrameMaker, Plain Text, etc.
- Ein XML-Schema, das die eigene DTD unterstützen kann.

5 Die Document Type Definition DTD

Ausgangspunkt für meine Bemühungen ist die miniLaTeX-DTD aus „Mit LaTeX ins Web“ von Goossens/Rahtz [Goossens und Rahtz, 1999, Anh. B.4.5].

Eine DTD beschreibt die Syntax einer bestimmten XML-Anwendung, wie z. B. DocBook oder TEI. Daher kann man eine DTD zur Validierung einsetzen, d. h. man kann mit ihrer Hilfe herausfinden, ob eine XML-Datei auch den Regeln von, sagen wir DocBook entspricht. So richtig was anstellen kann man allein mit einer DTD-Datei nicht.[8]

5.1 DTD-Design-Strategie

Das folgende erhebt keinen Anspruch auf Originalität, aber das sind so die wichtigsten Grundsätze, an die ich mich gehalten habe, als ich die DTD erstellte:

- Sich von LaTeX-Befehlen inspirieren lassen und, wo's geht, die XML-Elemente nach den entsprechenden LaTeX-Befehlen benennen.
- So viele Elemente und Attribute wie nötig, so wenige wie möglich.
- Den Autor nicht zu allzu tiefen Schachtelungen zwingen.

[8] Eine interessante Frage für mich ist, ob man den XSLT-Standard nicht so hätte definieren können, daß man mit XSLT-Dateien auch validieren kann. Dann müßte man nämlich nicht mehr ständig zwei Dateien konsistent halten, was sehr umständlich ist. Aber ich bin kein Informatiker.

- Alles, was formatiert werden könnte, als Element einbauen, alles andere nur als Attribut.
- Möglichst wenige Attribute auf Standardwerte setzen, lieber mit `#IMPLIED` arbeiten. Nur dort Defaults, wo der Kontext nie eine Bedeutung hat.
- Nicht-XML-Syntax in Betracht ziehen (siehe Abschnitt 9.3 auf Seite 64).
- Die verschiedenen Gliederungsebenen (Kapitel, Abschnitt, ...) sollten sich nicht aus dem Kontext ergeben wie in einem System, wo es ein Element `<section>` gibt, und dessen Schachtelung die Hierarchie ergibt. Der Autor muß immer ausdrücklich wissen, was für eine *Art* von Abschnitt er gerade eröffnet.
- Im Zweifelsfall die Freiheiten des Autors einschränken, d. h. unübliche oder gar zweifelhafte Konstrukte verbieten. Das verbessert auch die Kompatibilität mit zukünftigen Versionen, wo man dann eventuell einige dieser Dinge freigibt, statt weitere einzuschränken.
- Möglichst viele Elemente mit Attributen für Referenzierung (Querverweise) und Landes-Sprache versehen.
- Für Module (Adreßbuch, Formeln, Bibliographie) nach bereits existierenden Standards suchen und eventuell diese verwenden.

5.2 XML Schema

Ein XML Schema ist ein junges Dateiformat, das irgendwann einmal die DTD in einiger Hinsicht ablösen soll [Schema-W3C, 2001]. Der große Vorteil ist, daß man mit Schema wesentlich feiner angeben kann, wie die Syntax aussehen soll. Leider ist ein Schema ebenfalls eine XML-Datei, was eine echte Quälerei ist (siehe auch Abschnitt 9.3 auf Seite 64). Glücklicherweise gibt es Programme, die eine DTD in XML Schema umwandeln können [dtd2xs, 2002]. Danach kann man dann feintunen und vergißt die Datei am besten ganz schnell.

Momentan können erst einige Programme schon was mit Schema-Dateien anfangen. Xerxes [Xerxes, 2002], Apaches XML-Parser und mit Xalan [Xalan, 2001] zusammen zu haben, kann damit schon validieren, aber Erfahrungsberichten zufolge ist die Unterstützung des Standards noch mangelhaft. Außerdem ist noch nicht ganz abzusehen, in welche Richtung diese Schema-Sprachen gehen werden. Es gibt da nämlich noch ernstzunehmende Alternativen (z. B. RELAX), die vermutlich in die nächsten Fassungen von XML Schema einfließen werden und u. U. tiefgreifende Änderungen mit sich bringen werden. XML Schema hat sich also noch alles andere als gesetzt.

Wohlgemerkt: Es geht hier nur um's *validieren*, also prüfen, ob die Syntax stimmt. Die Umwandlung nach irgendwas ist ein ganz anderer Vorgang, dabei kann, muß aber nicht validiert werden. (Allerdings ist zweifelhaft, ob eine Datei mit falscher Syntax korrekt umgewandelt wird.)

6 Das Selbstgebackene

Ein Beispiel sagt mehr als tausend Worte. Meine ursprüngliche Absicht war es, ein neues Heim für meine anstehenden textuellen Abenteuer zu finden. Das ist dann ein klitzekleinwenig aus den Fugen geraten. Aber was soll's, es war ganz gute Abend/Nachtunterhaltung. Tüftler bleibt eben Tüftler.

6.1 tbook

Ich habe meine eigene XML-Anwendung vorerst **tbook** genannt, wohlahnend, daß dieser Name wahrscheinlich mit drei Dutzend anderen Paketen kollidiert. Der SourceForge-Name, und da-

mit wenn man so will der „vollständige“ Name, lautet **tbookdtd**. Vorab als grober Überblick, was **tbook** bislang leistet, was also offenbar mit XML möglich ist:

- Ein XML Dateiformat, das man mit einem guten Allzweck-Editor eingegeben kann, ohne alt oder aggressiv zu werden (oder beides), und was nach ein bißchen Gewöhnung am Bildschirm so gut lesbar wie LaTeX ist.
- Vollautomatische Umwandlung in XHTML+MathML, DocBook, PDF und Postscript. Die Endprodukte sind technisch und typographisch weitestgehend so gut, wie ich es auch „von Hand“ erzeugt hätte.
- Ein HTML-4-Kompatibilitätsmodus, inklusive Bitmap-Versionen aller Gleichungen.
- Halbautomatische Erzeugung eines Literatur- und Stichwortverzeichnisses; eine BIBTeX-Datei anlegen, bzw. Index-Einträge absetzen, das muß man in guter LaTeX-Manier noch selber machen. Funktioniert auch für HTML.
- Unterstützung von Bildern, Tabellen und Gleichungen, auch numeriert.
- Unterstützung für Deutsch[9], Englisch[10], Französisch, Italienisch, Spanisch und Katalanisch.
- Umfangreiche Möglichkeiten der individuellen Anpassung.
- Bequeme Grafikeinbindung, benötigte Bitmap-/Vektorformate werden automatisch erzeugt. Psfrag funktioniert für alle drei Ausgabeformate.
- Bequeme Gesamtbedienung; alles läuft über Shell-Skripte, die z. T. ständig mit dem Dokument konsistent gehalten werden.

Ich werde weiter unten noch meinen Wunschzettel angeben, aber die zwei größten Probleme kann ich hier schon nennen:

1. Das ganze System ist wohl recht fragil. Es ist beispielsweise mein erstes Makefile, und daß es bei mir funktioniert, wird gar nichts heißen. Ich bin abhängig von vielen Programmen, die installiert sein müssen, und benötige neueste Browser.
2. Ich habe mich daran orientiert, was ich in der Vergangenheit benötigt habe. XML ist sehr viel restriktiver als LaTeX, einfach mal’n Kommando in der Präambel definieren, das läuft hier nicht. Zwar kann man über LaTeX-Pakete, CSS-Stylesheets etc. mit recht wenig Aufwand großen Einfluß auf das Layout nehmen, aber wenn jemand total andere Vorstellungen vom Aufbau eines Buches hat als ich, wird er wohl ein Problem haben.

   Insbesondere *zusätzliche* Strukturen sind ein Problem.

6.1.1 Anmerkungen zum Projekt tbook

Die DTD ist zunächst einmal auf mich zugeschnitten. Ich habe aber auch versucht, zu erraten, was andere brauchen. Ziel war es, mit XML meine bereits in LaTeX verfaßte Diplomarbeit eingeben zu können, mit dem Hintergedanken, daß diese strukturell sehr komplex war und daher gut als Test dient.

Wenn jemand interessiert daran ist, Dokumente mit XML zu erstellen, und wie ich mit DocBook und TEI nicht viel anfangen kann, steht es ihm frei, beliebige Teile meines Ansatzes zu benutzen, oder ihn als Startpunkt für eigene Unternehmungen zu benutzen. Insbesondere die

[9] GER/AUT und neue/traditionelle Rechtschreibung
[10] GB/US

MathML-Routinen sollten sich leicht aus ihrem Kontext reißen lassen. Ich habe versucht, alles zu kommentieren und zu dokumentieren (was natürlich immer zu spärlich ist).

Was mich betrifft, ist das Projekt tbook – abgesehen von Bugfixes – abgeschlossen. Es kann, was ich brauche. Wenn jedoch jemand das Projekt weiterführen möchte, soll er sich bei mir melden.

6.1.2 Homepage von tbook und die Dateien

Unter der Adresse `http://sourceforge.net/projects/tbookdtd` kann man die Dateien herunterladen. Die Projekt-Homepage ist unter `http://tbookdtd.sourceforge.net` zu finden, dort gibt es auch ein Beispieldokument in den drei Endformaten PS, PDF und HTML, sowie den XML-Quellcode, auch im Unicode-Format.

Wenn die DTX-Dateien ausgepackt sind, umfaßt das Paket im großen und ganzen das folgende:

Datei	Beschreibung
`tblatex.xsl`	XSLT Stylesheet für die Trafo tbook → LaTeX
`tbhtml.xsl`	Dasselbe für tbook → HTML
`tbcommon.xsl`	Gemeinsamer Code von den beiden obigen Dateien
`tbook.dtd`	DTD von tbook
`tbbib.dtd`	DTD der BibTeXML-Variante für tbook (siehe Abschnitt 8.13.1 auf Seite 58)
`tbadrdb.dtd`	DTD der Adreß-Datenbank (siehe Abschnitt 11 auf Seite 70)
`hmml2dst.dtd`	leicht modifizierte MathML2-DTD mit fast allen MathML- und HTML-Entitäten
`tbtoltx.xsl`	XSLT Hilfs-Stylesheet für tbook → LaTeX, um Konfigurierbarkeit herzustellen (siehe Abschnitt 6.11.5 auf Seite 25)
`tbtohtml.xsl`	Dasselbe für tbook → HTML
`tbook.sty`	LaTeX-Grundpaket für tbook
`tbook-pl.sty`	Default LaTeX-Stildatei für tbook(siehe Abschnitt 6.11.3 auf Seite 24)
`tbletter.cls`	Default LaTeX-Briefklasse (siehe Abschnitt 6.11.5 auf Seite 27)
`t1pcrv.fd`	Kleinere Courier-Variante
`ts1pcrv.fd`	Kleinere Courier-Variante
`tbookman.pdf`	Kurze englische Installation- und Bedienungsanleitung für tbook
`ltxmled.tex`	Quellcode dieses Textes
`ltxmled.bib`	Bibliographie dieses Textes
`tbook.ent`	Extra-Entitäten, die tbook definiert (siehe Abschnitt 6.7 auf Seite 20)
`tblatex.ent`	Mapping von Entitäten auf LaTeX-Macros (Abschnitt 9.1.3 auf Seite 62)
`tbrplent.w`	CWEB-Quellcode für tbrplent, dem Unicode-Filter (Abschnitt 9.1.3 auf Seite 62)
`tbcrent.w`	CWEB-Quellcode für tbcrent, zum Erzeugen einer Hilfsdatei für tbrplent und so weiter und so weiter (auch Abschnitt 9.1.3)
`Makefile`	Makefile für tbook
`*.dbj`	Quelldateien für `custom-bib` (jeweils vier mal zwei für jede Sprache und jedes Ausgabeformat, siehe auch Abschnitt 8.13.2 auf Seite 58)
`*.bst`	BibTeX-Stile, von `custom-bib` aus den dbj-Dateien erzeugt (ebenso acht Stück)
`merlinht.dif`	Patch um `merlinht.mbs` zu erzeugen, das BibTeX XML-fähig macht (auch Abschnitt 8.13.2 auf Seite 58)
`bibfix.l`	Flex-Filter, um BibTeX XML-fähig zu machen (ebenso in 8.13.2)

`tb2xindy.l`	Flex-Quellcode für tb2xindy, dem xîndy-XML-Präprozessor (das und das folgende siehe Abschnitt 8.7 auf Seite 55)
`tbhtml.xdy`	xîndy-Stildatei für HTML-Stichwortverzeichnis
`tblatex.xdy`	Ditto, nur für LaTeX
`tbook.xdy`	xîndy-Stildatei mit gemeinsamen Code für LaTeX und HTML

Für das Ausprobieren benötigt man ein modernes LaTeX, pdfLaTeX, Make, Flex, Sed, C++-Compiler, ps2pdf und einen modernen Browser. Emacs 21.1 und was zum validieren (beispielsweise nsgmls) wären schön. Außerdem bekommt man einige Schmankerl nur mit den Umwandlungstools für PNM-Bitmaps, genau gesagt werden die Programme `ppmtojpeg`, `pnmtopng` und `pnmfile` benötigt.

Das alles ist bei einem neueren Linux bereits dabei. Aber für Saxon, CWEB und xîndy muß man wohl selber sorgen. Zur Installation sage ich etwas ab Seite 52. Ferner *kann* man sich die MathML-DTD und die BIBTEXML-Tools herunterladen. Wenn man die mitgelieferten Linux-Binaries verwendet, kann man auf Flex, CWEB und C-Compiler verzichten.

6.2 Typischer Entwicklungszyklus

Man beginnt ein neues Projekt mit **tbook**, indem man ein Verzeichnis dafür anlegt, hineingeht und dort

```
tbprepare buch
```

aufruft. Dadurch wird eine Datei „`buch.xml`“ angelegt, die als Ausgangspunkt für das eigene Dokument dient. Außerdem werden Prototypen der Batchdateien mit den richtigen Privilegien ins aktuelle Verzeichnis kopiert.

Der typische Entwicklungszyklus ist dann: XML-Datei editieren/speichern – Umwandlungs-Skript aufrufen – eventuell Batchdateien aufrufen – LaTeX oder pdfLaTeX aufrufen. Für HTML-Dateien wird letzteres ersetzt durch das Öffnen im Browser. Wenn also mal eine Konfiguration steht, ist es sehr simpel.

6.3 Die Umwandlungsskripte

Es gibt drei Skripte, mit denen man **tbook**-XML-Dateien in brauchbares umwandeln kann.

```
tbtolatex buch
```

liest die **tbook**-Datei `buch.xml` und schreibt `buch.tex`, das sowohl für LaTeX, als auch für pdfLaTeX benutzt werden kann.

```
tbtohtml buch
```

liest die **tbook**-Datei `buch.xml` und schreibt `buch.xhtml`, was eine gültige “strict XHTML 1.1+ MathML 2.0”-Datei ist. Der Aufruf

```
tbtohtml -t buch
```

(`-t` $\hat{=}$ „traditionell“) erzeugt hingegen `buch.html`, was eine gültige “strict HTML 4.01”-Datei darstellt. Damit man damit Freude hat, muß man aber für dieses Dokument auch `tbtolatex` mit `-t` aufrufen, siehe Abschnitt 6.9 auf Seite 22.

```
tbtodocbk buch
```

liest die **tbook**-Datei `buch.xml` und schreibt `buch-db.xml`, was eine gültige DocBook XML 4.2-Datei ist.

Man kann auch Parameter übergeben, siehe dazu Abschnitt 6.11.4 auf Seite 25.

6.4 Die Batchdateien

Benutzt man den Saxon (aber in naher Zukunft wird das mit jedem modernen XSLT-Prozessor funktionieren), werden nebenbei folgende Batchdateien im aktuellen Verzeichnis erzeugt bzw. ständig mit dem Dokument konsistent gehalten:

- `makeidx` (durch `tbtolatex` und `tbtohtml`)
- `makebib` (durch `tbtolatex`, `tbtohtml` und `tbtodocbk`)
- `makepdfs` (durch `tbtolatex`)
- `makewebs` (durch `tbtolatex`)
- `makeeqns` (durch `tbtolatex`, nicht immer)
- `makeepss` (durch `tbtolatex`)

Diese Dateien werden im allgemeinen nur bei der LaTeX-Umwandlung mit `tbtolatex` generiert, `makeidx` und `makebib` jedoch auch (und mit anderem Inhalt) von `tbtohtml`. Die Datei `makeeqns` wird von `tbtolatex` nur dann erzeugt, wenn der XSLT-Parameter `html-equations` auf `'bitmaps'` steht. Aber dazu später mehr.

Damit diese Batchdateien funktionieren, müssen sie unter Linux das Ausführbar-Attribut haben. Dafür sollte aber schon `tbprepare` gesorgt haben. Unter Windows brauchen sie die Endung `.bat`, die sie auch bekommen, wenn man Saxon zusätzlich `assume-os='windows'` übergibt.

Die Batchdateien erzeugen für ihre Arbeit ein paar Hilfsdateien mit langen, exotischen Namen, die höchstwahrscheinlich nichts überschreiben. Man wird aber dennoch hineinschauen, um sich zu vergewissern.[11] Eine Liste dieser Dateien bietet Abschnitt 8.15 auf Seite 59.

`makeepss`, `makepdfs`, `makewebs` und `makeeqns` akzeptieren ein optionales Argument. Wenn es vorhanden ist, wird es als Name einer Grafik interpretiert, und nur diese wird dann prozessiert. Das ist bequem wenn man nur eine Grafik geändert hat und nicht deswegen alles neu machen will.

`makeidx` `makeidx` hat je nachdem, ob zuletzt nach LaTeX oder nach HTML umgewandelt wurde, verschiedenen Inhalt. Es ruft xindy auf und erzeugt ein Stichwortverzeichnis für das jeweilig letzte Ausgabeformat.

Vorausgesetzte Programme: `xindy`.

`makebib` `makebib` hat je nachdem, ob zuletzt nach LaTeX oder nach HTML umgewandelt wurde, verschiedenen Inhalt. Es ruft BIBTeX auf und erzeugt ein Literaturverzeichnis für das jeweilig letzte Ausgabeformat.

Vorausgesetzte Programme: `bibtex`.

`makeepss` Ruft man diese Datei auf, werden aus allen JPEGs, die für's LaTeXen benötigt werden, eps-Dateien, da dvips mit JPEGs ja nichts anfangen kann. Die beiden Batch-Dateien `makepdfs` und `makewebs` sind i. a. davon abhängig, daß vorher einmal `makeepss` aufgerufen worden ist.

Vorausgesetzte Programme: Nur `jpeg2ps`.

`makepdfs` Mit dem Aufruf dieser Batchdatei werden automatisch aus allen eps-Dateien pdf-Dateien erzeugt, für pdfLaTeX. Psfrag-Elemente werden mitverwurschtelt.

Vorausgesetze Programme: `latex`, `dvips` und `ps2pdf`.

[11] Außerdem sollte man niemals blind irgendwelche generierten Batchdateien fremder Leute ausführen! ;-)

makewebs Diese Batchdatei erzeugt die Grafikdateien (PNG und JPEG) für die HTML-Fassung. Die Dateien bekommen die Namens-Erweiterung „-web". Also wird aus `super.jpg` die Datei `super-web.jpg` und aus `geil.eps` die Datei `geil-web.png`.

Vorausgesetzte Programme: `latex`, `dvips`, `gs` (mit den Treibern `ppm` und `pnm`) sowie `ppmtojpeg` und `pnmtopng`. Falls man mit `anti-aliasing = false` arbeitet, werden statt dessen die Ghostscript-Treiber `jpeggray` und `pnggray` benutzt; die `p?mto*`-Umwandler werden dann nicht benötigt.

Hat man auch noch `include-image-dimensions` auf `'true'` (standardmäßig der Fall), wird auch noch das Programm `pnmfile` aufgerufen.

makeeqns Diese Batchdatei wird nur dann generiert, wenn `html-equations` auf `'bitmaps'` (und nicht auf `'mathml'`) steht. Sie bewirkt, daß von *allen* Mathematik-Elementen des Dokuments meist sehr kleine Grafiken erzeugt werden, die dann an der entsprechenden Stelle im HTML-Dokument eingefügt werden.

Man kann ihr auch eine Formel-Nummer in der Form `eqn-123` übergeben, dann wird nur diese eine Formel neu berechnet. Ein Segen, wenn man nur eine Formel geändert hat, denn der komplette Prozeß kann gut und gerne ein Kaffeepäuschen lang werden. Pech hat man natürlich, wenn man eine Formel einfügt oder löscht. Dann verschieben sich alle Nummern.

Vorausgesetzte Programme: Siehe `makewebs`.

makepage Während die PDF- und die Postscript-Fassung des Dokuments aus jeweils nur einer einzigen Datei bestehen, verteilt sich die HTML-Fassung auf zumindest eine HTML-Datei und u. U. viele Grafiken. Mit dieser Batchdatei wird ein Unterverzeichnis `public-`*`Dokumentname`* angelegt, und in dieses Verzeichnis alle zur Web-Seite gehörigen Dateien kopiert.

makeclean macht sauber, d. h. es löscht alle generierten Dateien, mit Ausnahme von denen, die LaTeX erzeugt hat (`aux`-Datei, `toc`-Datei etc). Aus Sicherheitsgründen kommen in dieser Batchdatei – wie in allen anderen auch – keine Wildcards zum Einsatz.

Summa summarum sollte die Aufrufsequenz

```
tbtolatex buch
makeepss
makepdfs
makewebs
latex buch
pdflatex buch
tbtohtml buch
```

eine Postscript-, PDF- und HTML-Fassung des Dokumentes erzeugen, alles mit korrekten Grafiken und möglichst platzsparend. Für ein Stichwortverzeichnis müßte man noch `makeidx`- und `(pdf)latex`-Aufrufe nachschalten, für ein Literaturverzeichnis `makebib`.

6.4.1 Gegenseitige Abhängigkeiten der Batchdateien

Die gegenseitigen Abhängigkeiten dieser ganzen Skripte und Batchdateien sind etwas trickreich. Grundsätzlich gilt jedoch, daß man nach signifikanten Änderungen im Dokument (z. B. neue Grafik) `tbtolatex` aufrufen sollte, damit alle Batchdateien aktualisiert werden.

Danach sollte man, wenn es ein JPEG war, `makeepss` aufrufen, zumal das ein billiger Aufruf ist, und dann `makepdfs` und `makewebs` nach Bedarf und mit dem Grafik-Namen als Argument.

Schließlich kann man nach (X)HTML oder DocBook umwandeln.

Literaturverzeichnis und Index verlangen, daß man nochmal `tbtohtml` oder LaTeX aufruft, weil sonst selbstredend die Änderungen nicht eingebunden werden.

Schief- oder kaputtgehen kann nichts; ein Aufruf zuviel ist im schlimmsten Falle Zeitverschwendung.

6.5 Bereitstellung von Grafiken

Die Grafikeinbindung ist bereits mit normalem LaTeX nicht ganz einfach, wenn man zusätzlich pdfLaTeX glücklich machen will, wird's noch schwerer, aber hier bei **tbook** müssen wir auch noch die Browser bedienen. Die Batchdateien aus Abschnitt 6.4 auf Seite 17 machen das Leben ganz wesentlich einfacher, aber man muß dafür „gute" Grafikdateien als Ausgangsmaterial bereitstellen. Die Grafiken müssen in demselben Verzeichnis wie das XML-Dokument stehen. (Das könnte sicher besser sein.)

Die Batchdateien erwarten folgendes:

Bitmaps (das Attribut `kind` des `<graphics>`-Elements ist `"bitmap"`) müssen als JPEGs vorliegen, die wahre Größe wird über die Auflösung ermittelt. Man braucht also ein Grafikprogramm, das nicht nur die Einstellung der Auflösung erlaubt, sondern diese dann auch korrekt in die jpg-Datei schreibt.

Vektorgrafiken (`kind` steht auf `"vector"`) müssen als eps-Dateien vorliegen. Theoretisch kann man auch Bitmaps hierin unterbringen, aber das sollte man besser nicht tun. Außerdem profitiert die HTML-Fassung von den originalen JPEGs. Man kann nämlich dort auf die meist verkleinerten Bitmaps klicken, um in einem neuen Fenster die Original-Bitmap in Augenschein nehmen zu können.

Overlays müssen als jpeg- *und* eps-Datei vorliegen, die beide selbstredend exakt gleiche Abmessungen haben müssen. *Diagramme* als LaTeX-Fragment mit der Endung `.pic`. Diagramme bleiben aber von den Batchdateien unberührt.

Also kurz: Bitmaps als JPEG, Vektorgrafiken als EPS. Wenn man diese Voraussetzungen erfüllen kann, wird die Grafikeinbindung trivial.

6.5.1 Overlay-Grafiken

Beim den Overlays zeigt sich wahrscheinlich ganz besonders der persönliche Geschmack des Autors, also mir. Ich kann mir gut vorstellen, daß sich andere fragen, was das soll. Ein Overlay ist ein JPEG, über das ein EPS geschrieben wird, das Labels und ähnliches enthält. Vorteile: Kleineres PDF/Postscript und auflösungsunabhängige Labels. Nachteile: Mehr Arbeit und u. U. ungenaue Positionierung.

Meine Erfahrung ist, daß man Overlays auf 1 pt genau positionieren kann. Die EPS-Dateien müssen eine saubere Bounding Box haben, besonders die linke untere Ecke muß mit der (gedachten) linken unteren Ecke der Bitmap zusammenfallen. Da sind einige Grafikprogramme etwas ungenau. Ich zeichne in Corel Draw meine Grafik immer in die linke untere Ecke, dort sitzt auch die Bitmap als Hintergrund-Layer. Ein unausgefülltes, rahmenloses Rechteck, das genauso groß wie die Bitmap ist, sorgt für die richtige Bounding Box beim EPS-Export. Die EPS-Datei müßte dann die Bounding Box auf `0 0 ...` haben. Wenn sie das nicht hat, muß man geringfügige Ungenauigkeiten hinnehmen oder von Hand nachbessern.

6.6 Editor

Zur Erzeugung von XML-Dateien benötigt man einen Editor. Man sollte besser nicht daran denken, einen zu benutzen, dem die Endung XML nichts sagt. Ich fand den Emacs klasse: Einfach eine XML-Datei einlesen, sich mit `C-h m` die Hilfe zum XML-Mode holen und durchlesen,

dann ist man dabei. Mit `C-c C-v` kann man sogar validieren, also die Syntax überprüfen. Eine Funktion, die man gerne mal „zwischendurch“ aufruft.

Ich habe keine speziellen DocBook-Editoren ausprobiert. Die werden natürlich noch hilfreicher sein, aber eben nur für DocBook.

6.6.1 Erfahrung mit dem Emacs

Meine Implementierung des Emacs 20.7, die bei S.u.S.E. Linux dabei war, hat einen speziellen XML-Modus, der den Quelltext einfärbt und automatisch einrückt.

Das hat bei mir vorzüglich funktioniert. Wenn ich ein Element einfügte, gab mir der Emacs eine Auswahl denjenigen Elemente, die laut der tbook-DTD – die der Emacs vorher geparst hatte – an der aktuellen Position erlaubt waren. Dasselbe galt für Attribute.

Wenn das Einfügen eines Elements ein anderes Element nötig machte (beispielsweise muß jedes `<figure>` ein `<graphics>` enthalten), fügte der Emacs letzteres Element automatisch schon ein. Auch notwendige Attribute wurden im Minipuffer sofort abgefragt und eingefügt.

Einziger Störfaktor war, daß der XML-Modus wohl nicht immer in den Emacs einkompiliert ist.

6.6.2 Emacs mit UTF-8

Mit dem Emacs 21 kann man UTF-8-Dateien bearbeiten. Auch das habe ich ausprobiert. Es ist zwar sehr schön, wenn man α oder ➊ direkt im Textfenster sieht, aber die Eingabe dieser Zeichen gestaltete sich recht umständlich, was mit an Sicherheit grenzender Wahrscheinlichkeit an mir lag.

Mangelnde Portabilität ist meiner Ansicht nach kein Thema. Wer mit einer XML-UTF-8-Datei nichts anfangen kann, kann das auch nicht mit einer XML-Latin-1-Datei. Alle XML-Tools *müssen* mit UTF-8 zurechtkommen. Und durch Email-Versand passiert heutzutage ja wohl nichts mehr. (Außerdem ergibt die UTF-8-Kodierung garantiert keine zusätzlichen Steuercodes.)

6.7 Besonderheiten von tbook-Dateien

Eine tbook-XML-Datei ist eine stinknormale wohlgeformte XML-Datei. Ich habe versucht, alle Eigenheiten, die LaTeX zur weiteren Verarbeitung benötigt, aus den XML-Dateien herauszuhalten. Insbesondere kann man LaTeX's aktive Zeichen benutzen, ohne sich über Backslashs Gedanken zu machen, und man kann die meisten wichtigen Unicode-Zeichen entweder über Entitäten oder die UTF-8-Kodierung eingeben.

Einige typographische Feinheiten, besonders diejenigen, die `german.sty` bietet, habe ich über Entitäten zur Verfügung gestellt. Von der folgenden Tabelle sind alle außer `&bph;` und `&nbhy;` aus HTML entnommen, auch wenn sie dort meist keine Bedeutung besitzen.

Unicode	Entität	LaTeX	Name
`&x0082;`	`&bph;`	`""`	break permitted here
`&x00AD;`	``	`"-`	soft hyphen
`&x200C;`	`‌`	`"|`	zero-width non-joiner
(`&x200D;`	`‍`	`\noboundary`	zero-width joiner)
`&x2010;`	`‐`	`"=`	hyphen
`&x2011;`	`&nbhy;`	`"~`	non-breaking hyphen
	`&sf;`	`\@`	space factor reset

„LaTeX“ bezieht sich hier natürlich teilweise auf die `german.sty`-Erweiterungen. Diese „Zero-Width-Irgendwas“ sind wohl eigentlich (laut RFC [2070], 1997, Absch. 4.2.3) für Arabisch und sowas gemacht. Der Zero-With-Non-Joiner scheint dort wie eine Art Ligatur-Aufbrechung zu die-

nen. Dann soll er das auch hier tun. Der Zero-With-Joiner macht sowas ähnliches wie „mache aus einem runden ‚s' am Wortende ein langes ‚s' ". Dürfte man wohl fast nie brauchen.

Die Entität „`&sf;`" macht nach und vor Punkten Sinn, und führt zu einer korrekten Behandlung des folgenden Leerraums. Man kennt das vom `\@`-Befehl aus LaTeX. Diese Entität ist in der Datei `tbook.ent` definiert, die automatisch von der DTD eingebunden wird. Diese Datei definiert auch die Entitäten `&LaTeX;`, `&TeX;` und `&tbook;`, die das jeweilige Logo erzeugen.

Außerdem gibt es dort etliche zwei-Buchstaben-Entitäten, die bei „ck" und Dreifach-Konsonanten in der alten Rechtschreibung für korrekte Trennung sorgen. So entspricht

```
Die Schi&ff;ahrt um die Haff&zwnj;insel ist langsam wie eine Schne&ck;e.
```

in LaTeX:

```
Die Schi"ffahrt um die Haff"|insel ist langsam wie eine Schne"cke.
```

(Mit dem kleinen Unterschied, daß die Entitäten auch bei einer anderen aktivierten Sprache als Deutsch funktionieren.) Lediglich das `‌` stört beachtlich, aber da es – im Gegensatz zu den beiden anderen Entitäten in diesem Beispiel – bloß für einen Unicode steht, kann man es in einem UTF-8-fähigen Editor auch durch *ein* Symbol darstellen.

6.8 XHTML und die Browser

Die Transformation mit `tbtohtml` erzeugt eine XHTML-1.0-Datei. XHTML ist HTML, das auf XML getrimmt wurde. Da (leider) die Browser sich über die Jahre angewöhnt haben, sämtliche Fehler/Dummheiten/schmutzigen Tricks der Web-Autoren zu verkraften, können auch alte Browser ohne Probleme mit den XHTML-Dateien umgehen.

Es gibt zwei wichtige Typen von XHTML-Dateien, "transitional" und "strict". Meine Dateien sind weitgehend "strict". Lediglich auf einige wenige, in striktem XHTML unerlaubte Attribute wollte ich aus Kompatibilitätsgründen nicht verzichten.

6.8.1 Das Problem mit älteren Browsern

Ein großes Problem ist natürlich MathML. Netscape 7 wird es sowohl in der Unix-, als auch in der Windows-Grundversion enthalten, für Macintosh ist es in Aussicht gestellt. Der Explorer wird sich dem nicht verschließen können, denn auf den Zug mit Mathematik ohne Plug-Ins werden wohl zu viele aufspringen. Zumal MathML auch außerhalb der Web-Welt Fuß faßt.

Das größte Problem bei der Anzeige von **tbook**-HTML-Dateien ist die oft mangelhafte Unterstützung des CSS-Standards. Netscape 4.77 baut da völligen Mist, der Internet Explorer 6 und besonders Netscape 6.2 machen es gut, aber nicht perfekt.

In diesem Zusammenhang: Ich verstehe den schlechten Ruf von Netscape 6 nicht; einige „Cracks" schwören darauf, NS 6 zu überspringen und weiterhin mit NS 4.77 herumzumachen, was nun wirklich *schlecht* ist. Andererseits kann ich nicht verstehen, daß immer wieder behauptet wird, der IE 6 hätte die beste CSS-Unterstützung. Das stimmt einfach nicht, er interpretiert zwar fast alles, stellt es aber häufiger als NS 6 falsch dar.[12]

Ich habe ein paar erträgliche Modifikationen an meinen CSS vorgenommen, damit die Dokumente in diesen Browsern nicht absolut grotesk dargestellt werden, aber beliebig bin ich diesen Armseligkeiten nicht entgegengekommen. Die neuesten Versionen schaffen es ja.

Mozilla 1.0 bietet bislang die beste Sicht auf die XHTML-Dateien, die **tbook** erzeugt. Da klappt alles. Mozilla bietet die Grundlage für Netscape 7. Der erstmal durchaus gerechtfertigten

[12] Lustiger Bug im IE 6: **tbook**-Dateien sind grundsätzlich exakt 2 Pixel zu breit und brauchen daher eine horizontale Scroll-Leiste, egal, wie breit das Browser-Fenster ist!

Forderung, auch für alte und sehr alte Browser zu arbeiten, bin ich durch eine HTML-4-Option begegnet, siehe Abschnitt 6.9.

tbooks XHTML-Dateien werden übrigens von dem Validator des W3C verrissen. Das liegt daran, daß der Validator zwar behauptet, XHTML-Dateien untersuchen zu können, das aber ganz schlicht und ergreifend nicht stimmt.

Wie dem auch sei: Die Standards stehen, es klappt schon jetzt, und die Zeit arbeitet für uns.

6.8.2 Der richtige Content-Type

Noch an ganz anderer Front droht Gefahr: Der Web-Server muß XHTML-Dateien, insbesondere, wenn das MathML in ihnen funktionieren soll, als `application/xhtml+xml`, oder wenigstens als `text/xml` liefern. Das scheint aber noch sehr untypisch zu sein. Für Dateien mit der Endung `.xhtml` kann beispielsweise ein `text/plain` herauskommen. Ein Setzen wie

```
<meta http-equiv="Content-Type" content="application/xhtml+xml"/>
```

in der HTML-Datei bringt nichts. Es kann eine Lösung sein, die XHTML-Datei mit der Endung `.xml` zu speichern. Ich habe das so auf der **t**book-Homepage auf SourceForge machen müssen. Das ist jedoch unsauber, und man sollte es als Bug beim Provider melden. Der Server muß lediglich anders konfiguriert werden.

6.9 Gutes altes HTML 4

Noch können nur sehr wenige Browser-Installationen etwas mit MathML anfangen. Ursprünglich habe ich mir gesagt, ist egal, die Zeit arbeitet für mich, sollen sich die Leute doch Mozilla 1.0 oder Netscape 7 herunterladen. Dann fiel mir auf, daß es ganz einfach ist, aus Gleichungen Bitmaps zu machen.

6.9.1 HTML-4-Dateien erzeugen

Wenn man schon Gleichungen als Bitmaps hat, kann man auch gleich versuchen, doch noch etwas kompatibler mit alten Browsern zu sein und beispielsweise kein *X*HTML, sondern HTML zu erzeugen. Gleichzeitig kann man noch die CSS-Anforderungen in diesem Modus etwas herunterschrauben. Heraus kam die Option `-t` („traditionell“), die man `tbtohtml` und `tbtolatex`[13] übergeben kann:

```
tbtohtml -t buch
```

Die entstehende Datei hat die Endung `.html`, ist auch kein X(HT)ML mehr, sondern – mit Ausnahme weniger Attribute – gültiges striktes HTML 4.01, und sie benutzt vereinfachtes CSS und Bitmap-Gleichungen. Das wird dann sehr manierlich in IE 6 oder Netscape 6 dargestellt, in Netscape 4.77 bleibt es wenigstens lesbar, wenn man zusätzlich

```
tbtohtml -t buch shift-equations=false
```

übergibt, was allerdings sehr schade ist, weil dann die Bitmap-Gleichungen nicht mehr auf der Grundlinie aufsitzen.

Im Explorer mußte ich die Schriftgröße einmalig ändern und dann wieder zurückstellen, um Darstellungsfehler zu beheben. Eindeutiger, unumgehbarer Bug, der vielleicht nur an meiner Installation liegt. In Netscape 4.77 durfte ich die Schriftgrößte nicht zu groß wählen. Netscape 6 machte keine Probleme. Die Explorer 3, 4 und 5 standen mir nicht zur Verfügung, der Text sollte aber, wenn schon keine Augenweide, so doch zumindest lesbar sein.

[13] bei `tbtolatex` führt das nur dazu, daß Bitmap-Gleichungen generiert werden

Übrigens habe ich diese HTML-4-Dateien durch den W3C-Validator gejagt. Der hatte nur etwas an einigen Attributen auszusetzen, auf die ich jedoch nicht verzichten möchte. Ansonsten bleibt festzustellen, daß diese Validatoren ziemlicher Mist sind. Selten habe ich in einem so prominenten Programm so viele Bugs gesehen, und in der entsprechenden Mailingliste berichten täglich andere ähnliches.

Kurzum: Mit der Option `-t` kann man die Dateien besser in alten Browsern ansehen, aber dann sehen sie nicht mehr so gut aus; insbesondere ist dann kein MathML mehr möglich.

6.9.2 Gleichungen als Bitmaps

So werden aus Formeln Bitmaps: Es wird eine Gleichungen-Gallerie erzeugt, die nichts anderes als eine LaTeX-Datei ist, die nur aus den Gleichungen besteht. Ähnliches habe ich ja auch schon für die Web-Fassungen aller Grafiken gemacht. Dann werden daraus Bitmaps gezogen (mit LaTeX, dvips und Ghostscript), und schließlich anstelle der Grafiken im Browser die Grafiken eingefügt.

Der Nachteil dieser Methode ist klar: Grafiken werden nicht garantiert zum umgebenden Text passen, erst recht nicht, wenn man die Schriftgröße etwas ungewöhnlich eingestellt hat. Außerdem hat man so hunderte, bei sehr großen Dokumenten tausende von kleinen PNGs herumfliegen. Die sind zwar insgesamt nicht besonders groß, aber für jedes einzelne muß, wenn ich das Web richtig verstehe, eine HTTP-Verbindung hergestellt werden, was natürlich zeitraubend ist.

Aber an sich funktioniert das vorzüglich. Mit Kantenglättung und teilweiser Transparenz sieht das dann richtig gut aus. Der größte Leckerbissen ist meiner Ansicht nach jedoch, daß LaTeX die Tiefen der Formel-Boxen speichern kann, das Stylesheet lädt das dann wieder und kann die Formel vertikal auf die Grundlinie aufsetzen.

Ähnliche Systeme drucken nämlich Ausdrücke wie E_y zu hoch, weil sie eine Unterlänge haben, was dann wie E_y aussieht. Das ist mit meiner Methode behoben, und manchmal kann man die Grafik gar nicht mehr als solche erkennen. Das ganze basiert auf einem CSS Befehl, den auch IE 6 und Netscape 6 verstehen.

Die Textgröße der Gleichungs-Bitmaps läßt sich übrigens mit dem XSLT-Parameter `equation-resolution` feintunen. Der steht standardmäßig auf demselben Wert wie `html-resolution`, welcher wiederum eine Vorgabe von 100 dpi hat.

Eine Möglichkeit, die sich daraus ergibt, ist die folgende: Während man das Dokument erzeugt, schaut man sich Previews in Mozilla 1.0 mit MathML an. Wenn alles für die Veröffentlichung fertig ist, erzeugt man die Bitmaps für alle Formeln.

Preview-LaTeX Seit **t**book Version 1.4 ist es möglich, für die Generierung von Gleichungs-Bitmaps Preview-LaTeX [Kastrup u. a., 2002], genauer gesagt das Paket um `preview.sty` herum, das einen Teil davon darstellt, zu benutzen. Der Geschwindigkeitsgewinn ist atemberaubend. Für alle Gleichungen muß jetzt nur noch *einmal* LaTeX, dvips und Ghostscript aufgerufen werden. Lediglich die pnm → png-Umwandlung muß für jede Grafik einzeln erfolgen. Der einzige Nachteil, nämlich größerer Platzverbrauch auf der Festplatte (temporär), fällt bei den winzigen Gleichungen kaum ins Gewicht. Man muß dafür `preview.sty` installiert haben und den XSLT-Parameter `preview-latex` auf `'true'` setzen.

Man könnte Preview-LaTeX auch für die Generierung der PDFs mittels `makepdfs` oder die Generierung der Web-Grafiken mittels `makewebs` einsetzen, aber bislang sah ich noch keinen zwingenden Grund. Grafiken müssen normalerweise nicht ständig auf den neuesten stand gebracht werden, zumindest nicht alle auf einmal, so wie es bei Gleichungen häufiger nötig ist, da durch Einfügen einer neuen Gleichung sich alle Nummern verschieben.

6.10 Was ist mit Altlasten-Dokumenten?

Kann man alte LaTeX-Dateien in XML umwandeln? Naja, zumindest dürfte es nur für nahezu triviale Fälle automatisch gehen. Im Prinzip ginge es über GELLMU (Abschnitt 2.2 auf Seite 7), aber dafür müßte man das originale Dokument überarbeiten, und ich fürchte, die geringe Ausgereiftheit des GELLMU-Projekts macht einem einen Strich durch die Rechnung.

Bessere Chancen hat man mit TeX4ht (Abschnitt 2.4 auf Seite 8), allerdings ist dafür eine *umfangreiche* Anpassung von TeX4ht an die jeweilige XML-Anwendung vonnöten, so daß es sich für einzelne Dokumente nicht rechnet.

Es ist wohl am wirtschaftlichsten, das alte LaTeX-Dokument in einen guten Editor zu laden, als XML-Datei abzuspeichern, und dann von vorne bis hinten durchzugehen und in XML umzuwandeln. Wie ich in Abschnitt 3.4 auf Seite 10 dargelegt habe, ist hier eben menschliche Intelligenz nötig, und Suchen–Ersetzen macht die Sache erträglich. Ich habe es genau so mit einem guten Fünftel eines 100-Seiten-Textes von mir gemacht, mit Tabellen, Abbildungen und Formeln, und ich muß sagen – es geht, ohne daß man die Wände hochläuft.

6.11 Individuelle Konfigurierung

6.11.1 Cascading Style Sheets (CSS2)

Mit dem XSLT-Parameter `css-file` kann man den Namen einer XML-Datei angeben, die z. B. folgenden Inhalt haben könnte:

```
<style xmlns="http://www.w3.org/1998/Style/CSS2">
  h1 { color: red }
</style>
```

Damit werden alle Überschriften 1. Stufe (bei Büchern die Kapitel) in rot gedruckt. Die beiden `<style>`-Tags müssen genau so aussehen, weitere Tags sind nicht erlaubt. Zwischen ihnen können beliebige CSS-Formatierungen stehen, die garantiert als *letzte* geladen werden und so Priorität über alle Default-Werte erhalten. Damit kann man das endgültige Erscheinungsbild der HTML-Fassung in weiten Grenzen ändern.

6.11.2 LaTeX-`cfg`-Datei

Existiert im LaTeX-Suchpfad (also vorzugsweise im aktuellen Verzeichnis) eine Datei namens `tbook.cfg`, so wird sie *direkt vor* `\begin{document}` eingelesen. Die darin enthaltenen LaTeX-Macros dürfen nicht dergestalt sein, daß ohne die `cfg`-Datei der LaTeX-Durchlauf einen Fehler produzieren würde.

Ansonsten kann man in dieser Datei alles fröhlich umdefinieren. Zum Beispiel das Makro `\MakeTitlePage` aus `tbook.sty`. Man kann auch andere Schriftarten auswählen oder den Satzspiegel ändern.

6.11.3 LaTeX-`sty`-Datei

Mit dem XSLT-Parameter `sty-file` kann man die Standard-Stildatei von `tbook-pl.sty` umbiegen auf eine beliebige andere Stildatei, die im Suchpfad steht. In dieser Datei kann man dann auf etwas saubere Art und Weise, als das in einer `cfg`-Datei möglich ist, Anpassungen vornehmen. Folgendes ist aber zu beachten:

- Wenn man Veränderungen des Satzspiegels vornimmt, darf daß nur geschehen, wenn es sich nicht um eine „Gallerie" handelt. Darauf prüft man mit `\ifgallery`:

```
\ifgallery\else
\RequirePackage[...]{geometry}
\fi
```

- Man beachte, daß bei Beginn des Dokuments der Befehl `\pagestyle{fancy}` aktiviert wird. Kopf- und Fußzeilen sollte man also mit `fancyhdr`-Befehlen bewerkstelligen.
- Typischerweise wird man `\MakeTitlePage` re-definieren. Die Bedeutung der Parameter findet sich in `tbook.sty`.

Man kann im `class`-Attribut des Top-Level-Elements (`<book>` oder `<article>`) den Namen eines LaTeX-Pakets angeben, daß man gerne immer zum **tbook**-Dokument geladen haben möchte. Ist es nicht vorhanden, wird während des LaTeX-Laufs eine Warnung ausgegeben. Mehr nicht, denn es sollte sich nur das Erscheinungsbild ändern. Keinesfalls sollte man seine Dokumente von einem eigens kreierten Paket abhängig machen!

Man kann also sagen

```
<book class="mystyle.sty">
  ...
```

womit `mystyle.sty`, und *nicht* `tbook-pl.sty` eingebunden wird. Die Dateiendung ist Pflicht und muß '`.sty`' sein. Der Kommandozeilen-XSLT-Parameter `sty-file` (siehe unten) hat jedoch stets die höchste Priorität.

6.11.4 XSLT-Parameter

Wie schon erwähnt, kann man einem XSLT-Prozessor Parameter übergeben. Beim Saxon geschieht das in der Form

```
tbtolatex <Dokumentname> Parametername=NeuerWert ...
```

Ist `NeuerWert` ein String, sollte er in `'...'` eingefaßt werden. Ferner ist „wahr" `'true'` und „falsch" `'false'`. Für meine Anwendung existieren die Parameter aus Tabelle 2 auf der nächsten Seite. Das sind aber nur die wichtigsten. Eine komplette Liste findet man am Anfang des Kapitels zum "Common Code" in der Datei `tbookxsl.dtx`.

Wenn man statt XHTML- HTML-4-Dateien (mit der Option `-t`) erzeugt, ändern sich einige Default-Werte, die dann eben für HTML 4 optimiert sind. Insbesondere wird kein MathML mehr erzeugt, siehe Abschnitt 6.9 auf Seite 22.

6.11.5 Totale Freiheit

Die bisher genannten Möglichkeiten zur Konfigurierung sind nur für Anpassungen geeignet, wie man sie typischerweise mit LaTeX-Paketen erreichen kann. Alles, was zwischen `\begin{document}` und `\end{document}` steht, wird durch die XSLT-Stylesheets produziert und kann so nicht verändert werden.

Die ganze Umwandlungsarbeit übernehmen `tblatex.xsl` und `tbhtml.xsl`. Diese Dateien sollte man nicht verändern (aus ähnlichen Gründen, aus denen man auch nicht `book.cls` modifizieren sollte), aber es gibt noch zwei andere XSLT-Dateien: `tbtoltx.xsl` und `tbtohtml.xsl`. Diese beiden Dateien sind fast leer und tun erst mal nicht anderes als `tblatex.xsl` bzw. `tbhtml.xsl` einzubinden (wie LaTeX's `\input`).

Betrachten wir das Shell-Skript `tbtolatex` (für `tbtohtml` gilt dasselbe). Es ruft eigentlich gar nicht das Stylesheet `tblatex.xsl` auf, obwohl da der ganze Code drinsteht, sondern eben jenes `tbtoltx.xsl`.

Parametername	Default	Bedeutung
`adr-filename`	`'adrbook.xml'`	Dateiname der Adreßbuch-Datei
`bib-filename`	`'biblio.xml'`	Dateiname der BibTeX- oder BibTeXML-Datei
`css-file`	`''`	Dateiname einer CSS-Datei (XML-Format)
`sty-file`	`'tbook-pl'`	Dateiname einer LaTeX-`sty`-Datei, `'(none)'` für keine
`document-fontsize`	`'10pt'`	globale Schriftgröße des Dokuments. 10pt, 11pt oder 12pt.
`two-columns`	`'false'`	Bei `'true'` gibt's zweispaltigen Druck.
`graphics-fontsize`	`'default'`	Dokument-Schriftgröße, für die die Grafiken optimiert wurden. Default (selbe des Dokuments), 10pt, 11pt oder 12pt. Siehe "`basefontsize`" in Abschnitt 7.10 auf Seite 42.
`assume-os`	`'linux'`	angenommenes Betriebssystem. Mögliche Werte sind `linux`, `unix`, `windows` und `os2`.[14]
`anti-aliasing`	`'true'`	Default ist `'false'` für OS/2. Bei `'true'` werden Grafiken für HTML-Ausgabe geglättet. Benötigt aber zusätzliche Programme, siehe Abschnitt 6.4 auf Seite 18.
`include-image-dimensions`	aktueller Wert von `anti-aliasing`	`'true'` oder `'false'`. Bei `'true'` werden die Ausmaße einer Grafik in das `<img>`-Tag in die HTML-Datei geschrieben. Macht die Darstellung ein bißchen schneller, siehe auch Abschnitt 6.4 auf Seite 18. Geht nur bei Anti-Aliasing.
`html-resolution`	`100`	Auflösung in dpi für Grafiken der HTML-Fassung. Kann man zur globalen Skalierung benutzen.
`transparent-pngs`	`'false'`	Soll alles Weiß in PNGs transparent werden? (Auch in eventuellen Gleichungs-Bitmaps.)
`preview-latex`	`'false'`	`'true'` macht die Erzeugung der Bitmap-Gleichungen *viel* schneller.
`html-equations`	`'mathml'`	Mit `'bitmaps'` werden alle Formeln zu Bitmaps.
`jpeg-quality`	`75`	Qualität für JPEGs der HTML-Fassung in der üblichen Prozent-Einheit, d. h. zwischen 0 (Kubismus) und 100 (auch nicht besser als 95).
`create-image-comments`	aktueller Wert von `anti-aliasing`	Wenn `'true'`, werden in alle `*-web.{png,jpg}` Urheber-Kommentare eingefügt. In PNGs außerdem Angaben zu Erstelldatum, eventuell Beschreibung u. ä. Geht nur bei Anti-Aliasing.
`include-originals`	`'true'`	Wenn `'true'`, alle Originale der JPEGs mit in die HTML-Seite aufgenommen.
`rm`	`rm -f`	Default ist `del` für Windows & OS/2. Befehl zum Löschen einer Datei.[15]
`maximal-alt-length`	`50`	Maximale Zeichenzahl in den `alt`-Attribut Popup-Rähmchen in HTML, wenn man mit der Maus über der Grafik ist.
`desperate-measures`	`false`	Wenn `true`, dann werden die „Verzweifelten Maßnahmen“ angewendet, siehe Abschnitt 7.7 auf Seite 38.

[14]Nur `linux` ≡ `unix` und `windows` sind getestet, also wird der Rest mit Sicherheit schiefgehen. `;-)`
[15]Wenn man sowas nicht will, kann man es z. B. auf `'#'` setzen.

Tabelle 2: Mögliche Parameter für den XSLT-Prozessor. Es sind nur die wichtigen aufgeführt.

Der Witz ist, daß eine XSLT-Regel besagt, daß alles, was ein Stylesheet importiert, geringe Priorität hat. Mit anderen Worten, alles, was man in der Minidatei `tbtoltx.xsl` definiert, *überschreibt* das Standard-Verhalten. Eine typische Vorgehensweise wäre nun also, aus dem Original `tblatex.xsl` ein Template heraus zu kopieren und in `tbtoltx.xsl` hinein zu kopieren. Dort kann man es dann verändern.

Theoretisch könnte man so auch neue Elemente oder Attribute hinzufügen. Das ist selbstredend ein wenig heikler.

Andere XML-Anwendungen machen das ähnlich, genaugenommen scheint dieses kontrollierte Überschreiben die Standardmethode zur Konfigurierung zu sein. (Ist ja auch die einzige, die immer verfügbar ist.) DocBook nennt diese Mini-Front-End-XSLTs “Customisation Layers” oder „Treiberdateien“.

Brief-Stile Besonders wichtig ist diese Möglichkeit der Konfigurierung für eigene Brief-Stile. Hier ist es ein wenig schwierig (wenn ich auch zugebe, nicht unmöglich), alle sinnvollen Varianten über LaTeX-Pakete zu verwirklichen. Daher ist **t**books Standard-Verhalten ein wenig albern, denn es funktionieren erstmal nur Briefe von einem gewissen Bugs Bunny.

Ich habe mir da lange den Kopf über bessere Möglichkeiten zerbrochen, aber letztenendes kommt man um eine kleine XSLT-Konfigurierung bei Briefen nicht vorbei, ohne die schönen Vorteile von XML hier aufzugeben.

Das praktische an Briefen ist ja die Menge an impliziten Dingen. Briefkopf, Datum, Empfänger*adresse* – all das kann das Umwandlungsprogramm hinzufügen, und der Quellcode wird dadurch schlank und übersichtlich. Ich gebe zu, wenn man dann mal umzieht, ziehen alle alten Briefe mit um, nicht jedoch, wenn man sich einfach eine neue ID gibt.

Gut, nun zum Anpassen für eigene Briefe, was wirklich simpel ist: Das `"letter"`-Template wie oben gesagt aus der Originaldatei in die Mini-Datei kopieren und dort die Standard-Absender-ID durch die eigene ersetzen. Außerdem sollte man die Briefklasse ändern, voreingestellt ist `tbletter.cls`. Man kann aber auch `tbletter.cls` ändern, denn diese Datei ist zum Abschuß freigegeben.

Mit ein bißchen Mehraufwand kann man dem System auch beibringen, mehrere Absender-Identitäten parallel zu kennen und korrekt umzusetzen.

6.12 Farbe

Farbe ist ein schönes Beispiel dafür, daß es möglich und sinnvoll sein kann, CSS-Angaben auch für die LaTeX-Ausgabe zu interpretieren.

Man kann ja fast jedem **t**book-Element über das `style`-Attribut CSS-Eigenschaften geben, unter anderem auch zur Farbe. Also z. B.

```
<p style="color: red">Das ist jetzt in rot.</p>
```

Wenn das zu HTML verarbeitet wird, wird das `style`-Attribut einfach durchgeschliffen und im Browser erscheint der Text dann rot. Für LaTeX wird die CSS-Angabe durch das XSLT-Stylesheet geparst und in einen `\color{red}`-Befehl umgewandelt.

Auch die anderen Syntaxen des CSS-`color`-Befehls werden verstanden. Siehe dazu eine beliebige CSS-Erklärung.

Leider scheint pdfLaTeX Probleme mit der Einfärbung der *korrekten* Textstellen zu haben. Da kann es dazu führen, daß ein solcher Farbbefehl dafür sorgt, daß die komplette Seite *davor* rot wird. Meistens klappt es aber.

7 Überblick über Elemente und Attribute

Ein minimales tbook-Dokument sieht wie folgt aus:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE book PUBLIC "-//Torsten Bronger//DTD tbook 1.3//EN"
                      "http://tbookdtd.sourceforge.net/tbook13.dtd">
<book xml:lang="de">
  <frontmatter>
    <title>Ein kleines Buch</title>
    <author>Torsten Bronger</author>
  </frontmatter>

  <mainmatter>
    <chapter>
      <heading>Erstes Kapitel</heading>

      <p>Small is beautiful.</p>
    </chapter>
  </mainmatter>
</book>
```

Auf den folgenden Seiten werden einige Sammelbegriffe gebraucht. Die Bezeichnung „*Inline-Element*“ umfaßt folgende Elemente:

Font-Manipulation: `<em>`, `<visual>`, `<verb>`,
Mathematik: `<m>`, `<math>`, `<ch>`,
Querverweise: `<cite>`, `<pageref>`, `<ref>`, `<vref>`, `<mathref>`,
Index: `<ix>`, `<idx>`, `<indexsee>`,
Sonstiges: `<url>`, `<hspace>`, `<unit>`, `<relax>`, `<wrap>`, `<footnote>`, `<graphics>`, `<latex>`.

Die Bezeichnung „*Block-Element*“ umfaßt die folgenden Elemente:

Listen: `<description>`, `<enumerate>`, `<itemize>`,
Mathematik: `<math>`, `<dm>`, `<ch>`,
Eingeschobenes Material: `<quote>`, `<verbatim>`, `<verse>`,
Sonstiges: `<p>`, `<multipar>`, `<tabular>`, `<latex>`.

Die Bezeichnung „*Abbildung/Tabelle*“ umfaßt die Elemente `<figure>` und `<table>`.

7.1 Elemente auf obersten Ebenen

<book> – Buch (Wurzelelement)

Attribute:

`xml:lang`	Sprache (Standard: `"en"`)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `frontmatter, mainmatter, backmatter?`

Beschreibung: Dieses Element umklammert alle anderen XML-Elemente (wie das `<html>`-Tag in HTML) eines *Buches*. Das `id`-Attribut macht wahrscheinlich wenig Sinn.

`<frontmatter>` – Deckblatt-Info

Attribute: Keine.

Möglicher Inhalt: `title, author+, subtitle?, date?, keywords?, year?, city?, graphics?, typeset?, legalnotice?`

Beschreibung: Alles, was normalerweise vor dem Inhaltsverzeichnis genannt wird. Nur Titel und ein Autor sind Pflicht, der Rest nur wenn man will, aber die Reihenfolge ist zwingend. Der erstgenannte Autor steht auch hinter dem Copyright.

`<author>` muß einfach sein, d. h. es sollte keine Informationen über Institut oder Email-Adresse enthalten. Wenn doch, wird's ignoriert.

`<mainmatter>` – der Text-Inhalt

Attribute: Keine.

Möglicher Inhalt: `(part | chapter)*, appendix?`

Beschreibung: Hier steht letztlich das ganze Buch drin, d. h. alle Kapitel oder Teile.

`<backmatter>` – der Ausklang

Attribute: Keine.

Möglicher Inhalt: `references?, index?`

Beschreibung: Literaturverzeichnis und Index. Nichts von dem ist Pflicht (wie Backmatter ohnehin nicht), wohl die Reihenfolge.

`<article>` – Artikel (Wurzelelement)

Attribute:

`xml:lang`	Sprache (Standard: `"en"`)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `title, author+, date?, keywords?, year?, abstract?, (`*`Block-Element`*` | `*`Abbildung/Tabelle`*`)*, section*, references`

Beschreibung: Dieses Element umklammert alle anderen XML-Elemente (wie das `<html>`-Tag in HTML) eines *Artikels*. Das `id`-Attribut macht wahrscheinlich wenig Sinn.

Das `<author>`-Attribut kann für Artikel auch `<newline>` und `<footnote>` enthalten, um z. B. das Institut anzugeben.

<letter> – Brief (Wurzelelement)

Attribute:

from	eindeutiges Zeichen des Absenders
formal	"true" für „förmlicher Brief“ oder "false"
xml:lang	Sprache (Standard: "en")
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: city, date, to, subject, opening, *Block-Element**, closing

Beschreibung: Dieses Element umklammert alle anderen XML-Elemente (wie das <html>-Tag in HTML) eines *Briefes*.

from kann z. B. "Torsten Bronger" sein. Es sollte möglichst stets gleich bleiben und global eindeutig sein, soweit man das hinkriegen kann. Anhand dieses Attributs identifiziert nämlich nachher der XSLT-Prozessor (oder wer auch immer), ob er für diesen Absender überhaupt zuständig ist (Briefkopf usw.). Außerdem muß das owner-Attribut des Adreßbuches dazu passen.

Wenn man formal nicht ausdrücklich setzt, wird der Vorgabe-Wert u. U. aus dem Adreßbuch-Eintrag genommen (auch wenn eine Adresse ausdrücklich als Inhalt von <to> angegeben sein sollte). Geht das nicht, wird "false" (Privatbrief) angenommen.

Das id-Attribut macht wahrscheinlich wenig Sinn.

7.2 Teile, Kapitel und Abschnitte

<heading> – Überschrift eines Kapitels o. ä.

Attribute:

xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: (*Text* | *Inline-Element*)*

Beschreibung: Alle Gliederungselemente (<part>, <chapter>, ...) beginnen mit diesem Element, das die Überschrift für den jeweiligen Abschnitt enthält.

<part> – Buchteil

Attribute:

xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: heading, chapter*

Beschreibung: Dieses Element umschließt einen Teil eines Buchs.

<chapter> – Kapitel

Attribute:

`kind`	`"preface"`, `"introduction"`, `"acknowledgements"` oder `"colophon"` – kennzeichnet spezielle Kapitel
`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `heading, aphorism?, (`*`Block-Element`* `|` *`Abbildung/Tabelle`*`)*, section*`

Beschreibung: Umschließt ein Kapitel. Es wird auf jeden Fall in das Inhaltsverzeichnis aufgenommen, es sei denn, `kind` ist gegeben. In `<aphorism>` kann man noch ein kleines gewitztes Zitat einer berühmten Person über das Kapitel drucken lassen. Es wird zur Zeit noch kein Unterschied gemacht zwischen `"acknowledgements"` und `"colophon"`. Ein eventuelles Vorwort (`"preface"`) wird *vor* das Inhaltsverzeichnis gedruckt.

<section> – Abschnitt

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `heading, (`*`Block-Element`* `|` *`Abbildung/Tabelle`*`)*, subsection*`

Beschreibung: Umschließt einen Abschnitt. Er wird auf jeden Fall in das Inhaltsverzeichnis aufgenommen.

<subsection>, <subsubsection>, <paragraph> und <subparagraph>

Diese Gliederungselemente sind völlig analog zu `<section>` definiert. Das `"subsection*"` im »möglichen Inhalt« ist durch die entsprechende, nächstniedrigere Gliederungsebene zu ersetzen.

Ins Inhaltsverzeichnis wird nur noch bis `<subsection>` aufgenommen.

<appendix> – Anhang

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `chapter*`

Beschreibung: Umschließt die Kapitel des Anhangs. Entspricht der `appendix`-Umgebung von LaTeX.

7.3 Absätze

<p> – Absatz

Attribute:

`skip`	Abstand vor der Absatz: `"small"`, `"med"` oder `"big"`
`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: (*Text* | *Inline-Element* | *Block-Element*)*

Beschreibung: Umklammert *einen* Absatz. `skip` entspricht dabei dem LaTeX-Befehl `\`*größe*`skip`, wobei der Inhalt des Attributes `skip` dann für *größe* eingesetzt wird.

<multipar> – Sequenz einfacher Absätze

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: (*Text* | *Inline-Element*)*

Beschreibung: Die Absätze, die hiermit umklammert werden, werden durch „`*`“-Zeichen voneinander getrennt. Leerzeilen, die nicht durch XML-Elemente eingeklammert werden, werden ignoriert. Man kann sie daher z. B. nach (oder vor) jedem „`*`“ einstreuen, um LaTeX-Feeling zu haben.

<newline> – Zeilenumbruch

Attribute:

`vspace`	Vertikaler Sprung

Möglicher Inhalt: Keiner.

Beschreibung: Entspricht `<br>` in HTML bzw. `\\` in LaTeX. Optional kann man auch angeben, wieviel Platz zur nächsten Zeile gelassen wird. Im Gegensatz zu LaTeX darf so ein Zeilenumbruch nur an relativ wenigen Stellen benutzt werden, also insbesondere nicht einfach so im Fließtext.

Das Attribut `vspace` wird dabei behandelt wie ein Abstand, der in LaTeX in eckigen Klammern nach dem `\\` angegeben würde.

<footnote> – Fußnote

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: (*Text* | *Inline-Element*)*

Beschreibung: Umklammert eine Fußnote, die an der aktuellen Position in den Fließtext eingefügt wird.

7.4 Font-Formatierung

`<em>` – Hervorhebung

Attribute: Keine.

Möglicher Inhalt: (*Text* | *Inline-Element*)*

Beschreibung: Hebt seinen Inhalt hervor, was praktisch eine Kursivstellung bedeutet.

`<visual>` – visuelle Formatierung

Attribute:

markup	"nm", "rm", "it", "sc", "bf", "sf", "sl", "tt", "vs" (Angabe ein Muß)

Möglicher Inhalt: (*Text* | *Inline-Element*)*

Beschreibung: Formatiert den Inhalt entsprechend dem Attribut `markup`. Das meiste ist wohl selbsterklärend, `"nm"` stellt auf „normal" um, allerdings ohne die Schrift*familie* zu verändern. `"vs"` („Versalien") ist für Worte wie „MOVPE" gedacht, also komplett großgeschriebene Worte, wie sie Naturwissenschaftler und Informatiker lieben.

7.5 Querverweise

`<ref>` – einfacher Querverweis

Attribute:

refid	ID dessen, auf das man verweisen will (Angabe ein Muß)

Möglicher Inhalt: (*Text* | *Inline-Element*)*

Beschreibung: Tut das, was man vom LaTeX-Befehl `\ref` kennt. Das „Zielobjekt", das durch `refid` bestimmt wird, muß also eine Abbildung, Abschnitt o. ä. sein, also etwas, dem man eine Nummer zuordnen kann. Der Inhalt dieses Elements ist eine Bezeichnung, die davorgeknallt wird, aber zum Link dazugehört, z. B.:

```
<ref refid="Übersichtstabelle">Tabelle</ref>
```

Daraus wird dann etwa „`Tabelle~2.1`" in LaTeX gebastelt, in HTML ist dann „`Tabelle 2.1`" *komplett* als Link formatiert, nicht etwa nur die „`2.1`".

`<vref>` – Querverweis mit Seite

Attribute:

refid	ID dessen, auf das man verweisen will (Angabe ein Muß)

Möglicher Inhalt: (*Text* | *Inline-Element*)*

Beschreibung: Siehe `<ref>`, nur daß `<vref>` unter Benutzung des LaTeX `varioref`-Pakets eventuell noch die Seitennummer/-bezeichnung dahinter schreibt. In HTML gibt es zwischen `<ref>` und `<vref>` keinen Unterschied.

`<pageref>` – Verweis auf eine Seite

Attribute:

`refid`	ID dessen, auf das man verweisen will (Angabe ein Muß)

Möglicher Inhalt: `(`*`Text`*` | `*`Inline-Element`*`)*`

Beschreibung: Fügt die Seitennummer des Objekts ein, das die `id` namens `refid` hat. In HTML wird ein `[hier]` eingefügt, auf das man klicken kann, in LaTeX wird ein eventueller Inhalt vor der Seitennummer ausgegeben. So wird also aus

```
Siehe <pageref refid="Superformel">Seite</pageref>.
```

`"Siehe Seite~12."` in LaTeX und `"Siehe [hier]."` (mit anklickbarem „hier“) in HTML.

`<cite>` – Literaturverweis

Attribute:

`refid`	Label(s) der Literatur, auf das man verweisen will (Angabe ein Muß, falls mehrere, dann durch Leerzeichen getrennt)
`kind`	`"text"`, `"paren"`, `"imparen"` oder `"nocite"`

Möglicher Inhalt: `(`*`Text`*` | `*`Inline-Element`*`)*`

Beschreibung: Fügt einen Literaturverweis auf `refid` ein. Das Attribut `kind` bestimmt die Art der Einfügung. Während `"text"` etwa

```
Müller et. al. (1978)
```

erzeugt, macht `"paren"` (für „parentheses“) daraus

```
(Müller et. al. 1978)
```

`"imparen"` (für „implicit parentheses“) produziert

```
Müller et. al. 1978
```

Man kennt das aus dem `natbib`-Paket von LaTeX. *Wichtig*: Wenn man eine BibTeXML-Datei benutzt, müssen dort die `key`-Felder wie `"text"` aussehen. `"nocite"` fügt gar nichts in den Text, wohl aber die Literaturstelle ins Literaturverzeichnis. (Es funktioniert übrigens

```
<cite refid="-" kind="nocite"/>
```

wie ein `\nocite{*}` in LaTeX. Ich muß ‘`-`’ statt ‘`*`’ benutzen, weil `refid` vom Type `NMTOKENS` ist, was in allen anderen Fällen sehr bequem ist; nur leider erlaubt das keine ‘`*`’.)

Der Inhalt eines `<cite>`-Elements wird zum optionalen Parameter von LaTeXs `\cite`-Befehl. So wird aus

```
<cite refid="Mueller1978"><em>erstes</em> Kapitel</cite>
```

folgendes: „Müller (1978, *erstes* Kapitel)“

Der Vorgabewert für `kind` ist `"text"`, es sei denn, das `<cite>`-Element wird unmittelbar von Klammern eingerahmt. Dann ist er `"imparen"`.

`<mathref>` – „Not“-Verweis auf eine Formel

Attribute:

`refid`	Label der Gleichung, auf dir man verweisen will (Angabe ein Muß)

Möglicher Inhalt: `(`*`Text`* `|` *`Inline-Element`*`)*`

Beschreibung: Siehe `<ref>`, nur daß `<mathref>` ausschließlich auf Gleichungen verweist, indem es als Label den Inhalt der ersten Spalte einer `<mlabeledtr>` benutzt, so wie auf Seite 74 als dritte Möglichkeit beschrieben. Eigentlich sollte eine solche Formel auch eine `id` haben, und auf die kann man dann mit `<ref>` o. ä. verweisen. Man braucht `<mathref>` also gar nicht.

(Ich mußte dafür ein eigenes Element anlegen, weil ich einerseits auf diese seltsamen Labels verweisen können wollte, aber andererseits (wegen der Validierung) mittels `<ref>`s nur auf richtige `id`s verwiesen werden kann.)

7.6 Einfache Mathematik

`<m>` – einfache Inline-Formel

Attribute:

`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Text`*

Beschreibung: Entspricht `$...$` in LaTeX, aber mit abweichender Syntax. Siehe Abschnitt 9.3.2 auf Seite 64. Achtung: ist eine `id` angegeben, wird daraus automatisch eine abgesetzte (und numerierte) Formel.

`<dm>` – einfache abgesetzte Formel

Attribute:

`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Text`*

Beschreibung: Entspricht `\[...\]` in LaTeX, aber mit abweichender Syntax. Siehe Abschnitt 9.3.2 auf Seite 64.

`<ch>` – chemische Formel

Attribute:

`display`	`"inline"` oder `"block"`
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Text`*

Beschreibung: Entspricht <m> bzw. <dm> (je nach Wert für `display`), produziert allerdings eine einfache chemische Summenformel. Siehe Abschnitt 9.3.2 auf Seite 64.

<theorem> – (mathematischer) Satz etc.

Attribute:

countlike	Verwandte Satz-Klasse, oder "(none)", oder "(global)"
layout	"plain" (Voreinstallung), "definition" oder "remark"
xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: heading?, subject?, (*Text* | *Inline-Element* | *Block-Element*)*

Beschreibung: Mit diesem Element fügt man einen (mathematischen) Satz, Korollar, Lemma, eine Definition, oder aber eine Bemerkung, Übung, Beispiel etc. ein. Ich nenne das hier zusammenfassend einen Satz. Man kann es überall dort einbauen, wo auch eine gleitende Umgebung erlaubt wäre.

Für dieses Element ist die Angabe des `class`-Attributes essentiell. Es wird interpretiert als die *Art* des Satzes. Man könnte z. B. sagen

```
<theorem class="Bemerkung">Eine kurze mathematische Bemerkung.</theorem>
```

Gibt kam keine `class` an, wird eine Voreinstellung benutzt (die ist dann „Satz“).

Das `countlike`-Attribut enthält den Namen einer anderen <theorem>-Klasse, die mit der aktuellen die Numerierung teilen soll. Normalerweise bekommt nämlich jede Klasse ihre eigene Zählung. Beispiel:

```
<theorem class="Korollar"
  countlike="Lemma"><subject>Müllers Korollar</subject>Ein kurzes
  Korollar.  Falls es Korollar Nummer 4 ist, wird das nächste Lemma die
  Nummer 5 tragen.</theorem>
```

Will man überhaupt keine Nummern für eine Satz-Klasse haben, setzt man `countlike` auf "(none)". Wenn man in Büchern nicht die Aufsplittung der Nummer in „Kapitel.Nummer“ haben möchte, setzt man `countlike` auf "(global)".

Die Bezeichnung für den Satz, der gedruckt wird, ist standardmäßig gleich dem `class`-Attribut. Wenn man etwas anderes will, muß man ein <heading>-Element einfügen.

Mit dem <subject>-Element kann man noch einen Beinamen/Beitext angeben, der in Klammern hinter die Satz-Bezeichnung gedruckt wird. Also so Sachen wie „Zornsches Lemma“, „Flächensatz“ oder den Schwierigkeitsgrad einer Übung.

Schließlich noch das `layout`-Attribut. Damit gibt man einen Druckstil vor. Mögliche Werte sind die drei vordefinierten Stile aus dem AMSLaTeX-Paket `amsthm`. Dies sind "plain", "definition" und "remark". "plain" ist die Voreinstellung.

Wichtig: Nur das *allererste* <theorem>-Element einer bestimmten Klasse darf die global gültigen Elemente und Attribute tragen, also <heading>, `countlike` und `layout`.

<proof> – mathematischer Beweis

Attribute:

xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `heading, (`*`Text`*`|` *`Inline-Element`* `|` *`Block-Element`*`)*>`

Beschreibung: Fügt einen mathematischen Beweis ein. Man kann es überall dort einbauen, wo auch eine gleitende Umgebung erlaubt wäre.

Wenn `<heading>` gegeben ist, wird dessen Inhalt anstelle des vorgegebenen Wortes „Beweis" an den Anfang des Textes gedruckt.

7.7 Sonstiges

`<url>` – externe Verweise

Attribute:

`name`	der URL (Angabe ein Muß)

Möglicher Inhalt: Keiner.

Beschreibung: Der URL mir in Schreibmaschinen-Schrift ausgegeben und mit der Adresse per Link verbunden.

`<hspace>` – horizontaler Leerraum

Attribute:

`dim`	Breite (Angabe ein Muß)

Möglicher Inhalt: Keiner.

Beschreibung: Macht einen horizontalen Sprung, wie der gleichnamige LaTeX-Befehl. `<hspace dim="1em"/>` sollte also dasselbe sein wie ein `\quad`.

`<relax>` – Entspannung

Attribute: Keine.

Möglicher Inhalt: Keiner.

Beschreibung: Garantiert wirkungslos. Ich mag halt nur keine Eingabe-Sprachen ohne einen Entspannungs-Befehl.

`<unit>` – physikalische Größe

Attribute: Keine.

Möglicher Inhalt: *`Text`*

Beschreibung: Fügt eine physikalische Größe ein. Also

```
<unit>3 m</unit>
```

ergibt in LaTeX-Sprache „`$3$\,m`“. Der Sinn ist der, daß man so global und nachträglich einstellen kann, wie groß der Leerraum zwischen Maßeinheit und Zahlenwert ist, außerdem werden die Ziffern im richtigen (mathematischen) Font dargestellt (in manchen Konfigurationen macht das einen Unterschied). Weiterer Vorteil: Etwas wie

```
Die Gravitationskonstante ist
<unit>6,672&middot;10^{-11} m^3 kg^{-1} s^2</unit>.
```

(auf Leerzeichen achten!) ergibt

Die Gravitationskonstante ist $6{,}672\cdot 10^{-11}\,\mathrm{m}^3\,\mathrm{kg}^{-1}\,\mathrm{s}^2$.

Also richtiges Spacing um’s Komma herum, Einheiten aufrecht mit kleinen Skips dazwischen. Man muß sicherstellen, daß das *erste* Leerzeichen zwischen Maßzahl und Einheit steht, oder alternativ beides durch eine „~“ trennen.

Wichtig: Funktioniert auch in MathML, weil **t**book auf einer leicht modifizierten Version von MathML basiert. Kann innerhalb `<math>` überall dort stehen, wo auch ein `<mrow>` erlaubt wäre.

`<latex>` – LaTeX-Code

Attribute:

`code`	LaTeX-Code (Angabe ein Muß)
`desperate`	`"true"` oder `"false"` (Standard) – Element der Endphase?

Möglicher Inhalt: Beliebig.

Beschreibung: Das ist eine Art `\special`-Befehl: Für die LaTeX-Ausgabe wird `code` verwendet, ansonsten der Inhalt dieses Elements interpretiert. Ist `desperate` auf `"true"` gesetzt, wird das `<latex>`-Element ignoriert.

Der Inhalt dieses Elements muß selbstverständlich **t**book-Code sein, kein HTML-Code!

Was ist dann der Sinn von `desperate`? Wenn man in die letzte Phase der Dokumenterstellung kommt, kann man damit die LaTeX-Ausgabe ein wenig feintunen, z. B. was den Seitenumbruch betrifft, denn dort kann man dann diese Elemente explizit aktivieren. In allen anderen Ausgabe-Formaten wird ein `desperate="true"` jedoch immer ignoriert. Typisches Beispiel:

```
<latex code="\newpage" desperate="true"/>
```

Man kann `<latex>` auch sehr schön in eigenen Entitäten benutzen. So ist z. B. **t**books Standard-Entität `&LaTeX;` wie folgt definiert:

```
<!ENTITY LaTeX "<latex code='\LaTeX{}'>LaTeX</latex>">
```

So bekommt LaTeX den String `\LaTeX{}` zu Gesicht, und die Browser sehen `LaTeX`.

`<wrap>` – Inline-Verpackung

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`*` | `*`Inline-Element`*`)*`

Beschreibung: Dieses Element formatiert nicht, es umfaßt Inline-Elemente, die dadurch eine Sprache oder eine ID zugewiesen bekommen. Man sollte sich nicht scheuen, dieses Element auch leer zu benutzen. Beispielsweise entspricht

```
<wrap id="TolleStelle"/>
```

ungefähr dem `\label{TolleStelle}` in LaTeX.

7.8 Zitiertes und Verbatimes

<quote> – Zitat

Attribute:

xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: (*Text* | *Inline-Element*)*

Beschreibung: Ein abgesetztes Zitat.

<verb> – Codeschnipsel

Attribute: Keine.

Möglicher Inhalt: *Text*

Beschreibung: Ähnlich zu `\verb` von LaTeX.

<verbatim> – Code-Auszug

Attribute:

xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: (*Text*| em | visual | ix | idx | indexsee)*

Beschreibung: Entspricht der `verbatim`-Umgebung von LaTeX. In diesem Zusammenhang ist XMLs `<![CDATA[...]]>`-Umgebung von Bedeutung.

<verse> – Gedichte

Attribute:

xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: (*Text* | *Inline-Element*)*

Beschreibung: Formatiert seinen Inhalt so, daß z. B. Zeilenumbrüche wie eingegeben erhalten bleiben, was beispielsweise für Lyrik essentiell ist.

<aphorism> – Zitat am Kapitelanfang

Attribute:

xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*Text* `|` *Inline-Element* `| caption)*`

Beschreibung: Umklammert ein nettes, kleines Zitat für den Kapitelanfang. `<caption>` enthält dabei den Ursprung, also typischerweise den Namen einer mehr oder weniger berühmten Person und eventuell eine Jahreszahl. Es darf nur höchstens ein `<caption>` vorhanden sein, und das muß dann ganz hinten stehen.

Achtung: Wenn man das Zitat in Englisch hat (und dementsprechend `xml:lang` gesetzt hat), der Rest des Textes aber in Deutsch ist, sollte man für die `<caption>` die Sprache wieder auf Deutsch setzen, da das den einleitenden Gedankenstrich beeinflußt. Es sei denn, man mag den langen Gedankenstrich.

7.9 Listen

<itemize> – Auflistung

Attribute:

xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `item*`

Beschreibung: Legt eine unnumerierte Liste der `<item>`s an.

<enumerate> – Aufzählung

Attribute:

xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `item*`

Beschreibung: Legt eine numerierte Liste der `<item>`s an.

<description> – glossarartige Liste

Attribute:

xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(term, item)*`

Beschreibung: Legt eine Liste analog zu LaTeXs `description`-Umgebung an. Dabei werden die `<term>`s als das interpretiert, das bei LaTeX als optionaler Parameter den `\item`s übergeben wird, also der Begriff, der erklärt werden soll. Die Erklärung steht dann im unmittelbar folgenden `<item>`.

`<item>` – Listenelement

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: (*Text* | *Inline-Element* | *Block-Element*)*

`<term>` – Description-Element

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: (*Text* | *Inline-Element*)*

Beschreibung: Siehe `<description>`.

7.10 Gleitumgebungen und Grafiken

`<figure>` – gleitende Abbildung

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `graphics, caption?`

Beschreibung: Umklammert eine Grafik und eventuell eine Bildunterschrift, die dann gemeinsam mit einer laufenden Nummer als Gleitelement plaziert werden. Wenn man auf die Abbildung-Nummer verweisen möchte, muß man ja eine ID angeben. Das muß die ID des `<figure>`-Elementes sein (und nicht etwa die der `<graphics>` oder die der `<caption>`).

Wenn man bei LaTeX zweispaltigen Druck aktiviert hat (siehe Abschnitt 6.11.4 auf Seite 25), dann wird automatisch ausgemessen, ob die Grafik über eine oder über beide Spalten reicht und die Grafik entsprechend gesetzt.

`<table>` – gleitende Tabelle

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `tabular, caption?`

Beschreibung: Umklammert eine Tabelle und eventuell eine Bildunterschrift, die dann gemeinsam mit einer laufenden Nummer als Gleitelement plaziert werden. Wenn man auf die Tabellen-Nummer verweisen möchte, muß man ja eine ID angeben. Das muß die ID des `<table>`-Elementes sein (und nicht etwa die der `<tabular>` oder die der `<caption>`).

Wenn man bei LaTeX zweispaltigen Druck aktiviert hat (siehe Abschnitt 6.11.4 auf Seite 25), dann wird automatisch ausgemessen, ob die Tabelle über eine oder über beide Spalten reicht und die Tabelle entsprechend gesetzt.

`<graphics>` – Bild

Attribute:

`file`	Dateiname *ohne Endung* (Angabe ein Muß)
`scale`	Vergrößerungsfaktor
`kind`	Quellbild-Sorte, `"vector"`, `"bitmap"`, `"overlay"` oder `"diagram"` (Angabe ein Muß)
`basefontsize`	Bezugs-Schriftgröße (Standard: Dieselbe wie im Dokument)
`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `psfrag*`

Beschreibung: Damit wird eine Abbildung bei der aktuellen Position (meist innerhalb einer `<figure>`) eingefügt. `kind` wird folgendermaßen interpretiert:

Bei LaTeX: `"vector"` und `"bitmap"` wird als eps- bzw. pdf-Datei gesucht (je nach TeX-Version). `"overlay"` wird als eps/pdf-Datei gesucht (für den Bitmap-Hintergrund) und genauso für den Vektor-„Überzug" gleicher Dimension, der aber noch ein „`l`" an `file` angehängt bekommt. `"diagram"` wird mit der Endung `.pic` einfach so eingelesen (`picture`-Umgebung, z. B. als Ausgabe von Gnuplot).

Bei HTML: `"vector"` und `"diagram"` werden als png-Datei gesucht, `"overlay"` und `"bitmap"` als jpg-Datei.

Das ist nun *meine* Implementierung. Letztlich liegt es aber am XML-Interpretierer, was passiert, und was man an Bild-Dateien bereitstellen muß. Geboten werden dem Interpretierer eben die Attribute `file` und `kind`, und der muß dann selber sehen, was er damit macht. Grafik-Einbindung scheint jeder LaTeXer etwas anders zu machen.

Die Bild-Dateien werden übrigens bequemerweise automatisch generiert, so daß man z. B. nie für pdf-Bilder selber sorgen muß, siehe dazu Abschnitt 6.5 auf Seite 19.

`basefontsize` kann `"10pt"`, `"11pt"` oder `"12pt"` sein. Der Sinn ist folgender: Manchmal ändert man die globale Schriftgröße des Dokuments. Damit die Labels in der Grafik, die mit `<psfrag>` eingefügt wurden, nicht aus allen Nähten platzen (oder umgekehrt, zu klein sind), kann man mit `basefontsize` wieder den alten Zustand lokal für diese Grafik wiederherstellen. Man kann es natürlich auch unabhängig davon dazu gebrauchen, die Label-Größe zu verändern.

`<caption>` – Bildunterschrift

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`* `|` *`Inline-Element`*`)*`

Beschreibung: Eine gleitende Abbildung oder Tabelle erhält nur dann eine Nummer, wenn eine `<caption>` enthalten ist. Die kann auch leer sein, dann steht eben bloß „Abbildung `<Nummer>`" darunter.

`<psfrag>` – Bildtext-Ersetzung

Attribute:

`tag`	Platzhalter in der eps-Datei (Angabe ein Muß)
`number`	Handelt es sich um eine Zahl? (`"true"`/`"false"`)
`contrast`	`"boxed"` unterlegt das Label mit einer weißen Box. `"inverse"` stellt es in Weiß dar.
`relsize`	`"large"` für groß, `"small"` für klein. Standard: `"normal"`.
`align`	Ausrichtung: `"left"` (Standard), `"right"`, `"center"`, `"ccenter"`
`interval`	Automatische Zahlen-Erzeugung
`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`*` | `*`Inline-Element`*`)*`

Beschreibung: Das `tag`-Attribut wird im eps-Bild gesucht und an seine Stelle der Inhalt des `<psfrag>`-Elements gesetzt:

```
<psfrag tag="x-Achse"><m>x</m>-Achse</psfrag>
```

Dabei kommt selbstredend das phantastische Psfrag-Paket zum Einsatz. Ist das Element leer, wird `tag` durch sich selbst ersetzt, was dann eben nur noch den Font und die Schriftgröße anpaßt:

```
<psfrag tag="Diagramm"/>
```

Will man ein Tag aus dem eps-Bild löschen, muß man es also z. B. durch ein Leerzeichen ersetzen:

```
<psfrag tag="war bloed"> </psfrag>
```

`align` gibt die Ausrichtung des eingesetzten Textes relativ zum ersetzten `tag` an:

`"left"`: Linkbündig auf derselben Grundlinie.

`"right"`: Rechtsbündig auf derselben Grundlinie.

`"center"`: Zentriert auf derselben Grundlinie.

`"ccenter"`: Horizontal und vertikal zentriert (die beiden gedachten „Boxen" von Tag und Ersetzungstext wären also perfekt ineinander geschachtelt).

`number` ist standardmäßig `"true"`, wenn `tag` offensichtlich eine Zahl ist oder irgendein `interval` angegeben ist, sonst `"false"`. Zahlen werden später so ausgegeben, als wären sie Teil einer Formel, was in Texten, die auch Minuskelziffern enthalten, einen Unterschied macht. Außerdem ist -4 hübscher als -4.

`interval` besteht aus drei Teilen, die durch Semikolons getrennt sind: Start, Ende und Schritt. So erzeugt `interval="0;10;1"` automatisch Ersetzungen für alle ganzen Zahlen zwischen 0 und 10. Das ist besonders praktisch für eps-Dateien aus Programmen wie Origin mit Achsen-Beschriftung. `tag` ist in diesem Fall übrigens eine Art Muster für das Eingabe-Format,

und der Inhalt des `<psfrag>`-Elements wird als Muster für's Ausgabe-Format interpretiert. Das geht irgendwie nach der DecimalFormat-Routine von Java 1.1, falls jemand was damit anfangen kann.

Am einfachsten erklären kann man es jedoch mit ein paar Beispielen:

```
<psfrag tag="#" interval="1;10;2"/>
```

ersetzt 1, 3, 5, 7 und 9 durch sich selbst (ändert also nur den Font). Jetzt was kompliziertes:

```
<psfrag tag="#.0" interval="-4.5;-6;-0.5">#,0</psfrag>
```

ersetzt `-4.5`, `-5.0`, `-5.5` und `-6.0` durch dieselben Zahlen, nur mit einem Komma statt dem Punkt. Schließlich macht

```
<psfrag tag="#.0" interval="-4.5;-6;-0.5">#,#</psfrag>
```

dasselbe, nur daß in der Ausgabe die Zehntel weggelassen werden, wo sie eh Null sind. Letztes Beispiel:

```
<psfrag tag="0.0" interval="-1.5;1;0.5">#,#</psfrag>
```

führt folgende Ersetzungen durch: `-1.5` → −1,5, `-1.0` → −1, `-0.5` → −0,5, `0.0` → 0, `0.5` → 0,5 und `1.0` → 1.

Übrigens: Es schadet überhaupt nicht, wenn man mit `interval` viel zu viele Psfrag-Ersetzungen erzeugt. So kann man ein und dasselbe `<psfrag>` in u. U. allen Diagrammen für schöne Achsenbeschriftungen nutzen.

7.11 Tabellen

`<tabular>` – Tabelle

Attribute:

`preamble`	Spaltenausrichtung
`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `tabhead?, tabbody`

Beschreibung: Die `preamble` sieht wie eine vereinfachte LaTeX-Tabellenpräambel aus, also z. B. „`lcc`“ für „drei Spalten, erste linksbündig, die beiden anderen zentriert“. Außer `l`, `r` und `c` ist allerdings nichts erlaubt, insbesondere keine senkrechten Linien (aus gutem Grund). Ist sie nicht angegeben, wird für alle Spalten „`c`“ angenommen.

`<tabhead>` – Kopfzeile

Attribute: Keine.

Möglicher Inhalt: `(hline | row | srow)*`

Beschreibung: Hier bekommt jede Spalte ihre Überschrift.

`<tabbody>` – Inhalt der Tabelle

Attribute: Keine.

Möglicher Inhalt: `(hline | row | srow)*`

`<row>` – Tabellenzeile

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `cell*`

Beschreibung: Umklammert eine Zeile in einer Tabelle und enthält die Zellen.

`<srow>` – Tabellenzelle, einfache Syntax

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Text`*

Beschreibung: Enthält eine Zeile einer Tabelle. Die Spalten werden durch „|“-Zeichen voneinander getrennt. Dies ist für einfache Tabellenzeilen, die keine weitere Formatierung benötigen, gedacht.

`<hline>` – Linie in einer Tabelle

Attribute:

`from`	Anfangsspalte (Standard: `"1"`)
`to`	Endspalte (Standard: letzte Spalte)
`trim`	`"lr"`, `"l"`, `"r"` oder `"no"` („kein“) – Trimmen der Enden

Möglicher Inhalt: Keiner.

Beschreibung: Erzeugt eine horizontale Linie in einer Tabelle. Normalerweise werden die beiden Enden der Linie, wenn die *innerhalb* der Tabelle liegen, links und rechts gestutzt. Mit `trim` kann man das explizit einstellen, und zwar für „links und rechts“, „links“, „rechts“ oder für keine Seite.

Die drei Standard-Linien vom `booktabs`-Paket sind übrigens immer da (auch in der HTML-Ausgabe) und müssen (dürfen) daher nicht explizit durch `<hline>`s gezeichnet werden.

`<cell>` – Tabellenzelle

Attribute:

`colspan`	Anzahl Spalten, die zusammengefaßt werden (Voreingestellt: `"1"`)
`align`	`"left"`, `"center"` oder `"right"` – Ausrichtung (Voreingestellt: Wert aus Präambel)
`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`* `|` *`Inline-Element`*`)*`

Beschreibung: Umklammert einen Eintrag in einer Tabelle. `colspan` entspricht dem `\multicolumn`-Befehl von LaTeX, und `align` ist wohl selbsterklärend.

7.12 Elemente der Frontmatter

`<title>` – Buchtitel

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`* `|` *`Inline-Element`* `| newline)*`

Beschreibung: Titel des Buches. Tritt auf dem Deckblatt auf, auf der zweiten Seite und eventuell in der Titel-Leiste des Browser-Fensters.

`<author>` – Autorenname

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`*`| newline | footnote)*`

Beschreibung: Der Name *eines* Autors. Er wird automatisch in Vor- und Nachname aufgespalten, damit man beide für die zweite Seite umdrehen kann. Dabei gehe ich folgendermaßen vor: Befindet sich ein „|“ im Inhalt, trennt der den Vor- vom Nachnamen, ansonsten nehme ich dafür das letzte Leerzeichen. Ganz tolle BIBTeX-Beispiele lassen sich damit zwar nicht nachempfinden, aber mir reicht's.

`<newline>` und `<footnote>` werden nur für `<article>` interpretiert, in `<book>` zählt nur die erste Text-Zeile.

`<subtitle>` – Untertitel

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`* `|` *`Inline-Element`* `| newline)*`

Beschreibung: Funktioniert genauso wie der Titel und erscheint sowohl auf dem Deckblatt, wie auch auf Seite zwei.

`<typeset>` – der Künstler

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Text`*

Beschreibung: Name des Schriftsetzers und eventuell Angaben zum verwendeten Font und verwendeten Programm (TEX).

`<date>` – Zeitpunkt des Drucks

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Text`*

Beschreibung: In der Formatierung ist man frei, d. h. es wird so unverändert im Dokument ausgegeben.

`<keywords>` – Schlüsselwörter

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Text`*

Beschreibung: Schlüsselworte, die das Dokument charakterisieren. Dieses Element dient ausschließlich zur Erzeugung von Meta-Informationen. Also für HTML für die `<meta>`-Elemente, und für pdfLATEX für das “Summary”-Fenster im Acrobat Reader.

`<year>` – Jahr des Drucks

Attribute:

`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Text`*

Beschreibung: Diese Jahreszahl(en) werden ans Copyright-Zeichen angehängt.

`<city>` – Stadt

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Text`*

Beschreibung: Die Stadt oder allgemeiner der Ort der Entstehung – ebenfalls für den Copyright-Vermerk.

Es kann auch in einem Brief für den Absende-Ort benutzt werden.

`<legalnotice>` – Hinweis zum Urheberrecht

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`*` | `*`Inline-Element`*` | p)*`

Beschreibung: Sermon zum Urheberrecht. Wenn dieses Element vorhanden aber leer ist, wird ein Standard-Text gedruckt.

`<abstract>` – Zusammenfassung

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `p+`

Beschreibung: Die Zusammenfassung wird bei einem Artikel an der gewohnten Stelle ausgegeben.

7.13 Literaturverzeichnis

`<references>` – Literaturverzeichnis

Attribute:

`bibfile`	BIBTEX-Dateiname
`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Block-Element*`*

Beschreibung: Fügt ein Literaturverzeichnis ein. Der Inhalt dieses Elements wird als (einleitender) Text über das Verzeichnis geschrieben. Also ist es meist leer.

`bibfile` enthält den Dateinamen für die BIBTEX-Datei. Eine eventuelle Endung wird ignoriert. Wird `bibfile` nicht angegeben, wird der Wert für `bib-filename` benutzt, der dem XSLT-Prozessor übergeben wird. Gibt's auch den nicht, wird `"biblio"` benutzt.

7.14 Stichwortverzeichnis

`<index>` – Stichwortverzeichnis

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: *`Block-Element*`*

Beschreibung: Fügt ein Stichwortverzeichnis ein. Der Inhalt dieses Elements wird als (einleitender) Text über das Verzeichnis geschrieben. Also ist es meist leer.

`<ix>` – Indexeintrag

Attribute:

`sortkey`	Suchschlüssel
`kind`	`"emph"`, `"bold"`, `"italic"`, `"start"` oder `"end"`
`xml:lang`	Sprache (geerbt)

Möglicher Inhalt: `(`*`Text`*` | `*`Inline-Element (außer Fußnote, Querverweis und Index)`*` | ix2)*`

Beschreibung: Der Inhalt wird als Indexeintrag benutzt. Es darf nur *ein* `<ix2>`-Element enthalten sein, und das muß, wenn vorhanden, ganz hinten stehen.

Mit `sortkey` kann man die Sortierung beeinflussen (das entspricht dem, was vor dem „`@`" im LATEX-Indexbefehl steht). Mit `kind` beeinflußt man das Markup der Seitenzahl im Index. `"start"` bedeutet dabei, daß hier eine Seitensequenz beginnt, die bis zum nächsten identischen Indexeintrag mit `"end"` reicht (entspricht „`|(`" bzw. „`|)`" in LATEX).

Umlaute werden automatisch korrekt einsortiert. Wie im normalen Fließtext auch, verlieren LATEXs aktive Zeichen ihre Sonderbedeutung. Die Zeichen `"!@|`, die in Index-Einträgen meist noch eine zusätzliche Brisanz besitzen, können ebenso sofort benutzt werden.

"emph" und "italic" führen zum selben Markup.

<ix2> – Indexeintrag, zweite Ebene

Attribute:

sortkey	Suchschlüssel

Möglicher Inhalt: `(Text | Inline-Element (außer Fußnote, Querverweis und Index))*`

Beschreibung: Befindet sich stets innerhalb eines <ix> und enthält die Verfeinerung des Indexeintrags. Also das LaTeX-Konstrukt

```
\index{Sänger!Twopac@2~Pac|emph}
```

sieht dann in XML so aus:

```
<ix kind="emph">Sänger<ix2 sortkey="Twopac">2 Pac</ix2></ix>
```

<idx> – Indexeintrag mit Einfügung

Attribute:

sortkey	Suchschlüssel
kind	"emph", "bold", "italic", "start" oder "end"
xml:lang	Sprache (geerbt)

Möglicher Inhalt: `(Text | Inline-Element (außer Fußnote, Querverweis und Index))*`

Beschreibung: Wie <ix>, allerdings wird der Inhalt auch an der aktuellen Position in den Fließtext eingefügt.

<indexsee> – Index-Siehe-Eintrag

Attribute: Keine.

Möglicher Inhalt: `ix, ix+`

Beschreibung: Dieses Element enthält also mindestens zwei <ix>. Damit wird im Index ein „*siehe*“-Verweis gebastelt, und zwar vom ersten auf den letzten. Sind mehr als zwei <ix>e umklammert, verweisen alle außer dem letzten auf das letzte.

7.15 Brief-Elemente

<to> – Empfänger

Attribute:

nickname	ID des Empfängers in der Adress-Datenbank.
xml:lang	Sprache (geerbt)
id	eindeutiger Bezeichner
style/class	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(Text | Inline-Element | newline)*`

Beschreibung: Enthält den Empfänger. Einzelne Zeilen können mit einem `<newline/>` voneinander abgesetzt werden. Wenn dieses Element gleichzeitig Inhalt hat und `nickname` gesetzt ist, wird nicht `nickname`, sondern der *Inhalt* für das Adreß-Feld benutzt. Wenn man also das `nickname`-Attribut nutzen möchte, muß man `<to>` leer lassen:

```
<to nickname="Jupp"/>
```

`<subject>` – Betreff-Zeile

Attribute:

`silent`	`"true"` oder `"false"`
`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`*` | `*`Inline-Element`*`)*`

Beschreibung: Enthält den Betreff des Briefes. Dieses Element muß zwar vorhanden sein, wird aber bei formellen Briefen (`formal` in `<letter>`) standardmäßig ausgegeben, in informellen Briefen standardmäßig unterdrückt. Mit `silent` kann man das Vorgehen explizit machen.

`<opening>` – Briefanrede

Attribute:

`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`*` | `*`Inline-Element`*`)*`

Beschreibung: Typischerweise eine Anredeformel.

`<closing>` – abschließende Grußformel

Attribute:

`kind`	`"above"`, `"below"` oder `"signature"`
`xml:lang`	Sprache (geerbt)
`id`	eindeutiger Bezeichner
`style/class`	Stiloptionen und -klasse (z. B. für CSS)

Möglicher Inhalt: `(`*`Text`*` | `*`Inline-Element`*`)*`

Beschreibung: Die Verabschiedungs-Grußformel unter dem Brief. `kind` steht für formelle Briefe (Attribut `formal` von `<letter>`) auf `"signature"`, für unförmliche Briefe auf `"above"`, aber das kann man auch explizit anders setzen:

`"above"` bedeutet, daß der Inhalt des Elements (also die Grußformel) *über* der Unterschrift sein soll, `"below"` für darunter, und `"signature"` für darüber, aber mit Namen des Absenders darunter.

Will man nur den Namen unter der Unterschrift haben, muß man also schreiben

```
<closing kind="signature"/>
```

und gut ist.

8 Interna von tbook

8.1 Dokumentation des Codes

In der Hoffnung, daß die ein oder andere Komponente meines Projektes jemandem nützt, habe ich versucht, alles so sorgfältig, wie es Zeit und Nerven zuließen, zu dokumentieren.

Insbesondere die MathML-Routinen, sowohl was die Umwandlung nach LaTeX, aber auch was die Interpretierung innerhalb von XML betrifft, könnten sicherlich auch in anderer Umgebung von Nutzen sein. Dasselbe gilt für das Unicode-Filter in CWEB.

8.1.1 DTX-Dateien

Ein Großteil ist in DTX-Dateien organisiert, die man so durch LaTeX jagen kann, und die auch ein recht brauchbares Stichwortverzeichnis enthalten. In `tbookdtd.dtx` kann man die Zeile `\OnlyDescription` auskommentieren und erhält eine komplette Liste mit allen Unicodes, die **tbook** unterstützt, und wie sie in LaTeX realisiert sind.

Ich habe gute Erfahrungen mit dem `doc.sty`-Paket für nicht-LaTeX, insbesondere XSLT, gemacht. Einzig bei den xîndy-Stilen hatte `doc.sty` seine Schwierigkeiten, da xîndy bedauerlicherweise keine Escape-Varianten für nicht-ASCII-Zeichen bietet. Daher muß ich nach dem Entpacken mit `docstrip` noch ein Sed-Skript über die Ergebnisse laufen lassen.

8.1.2 CWEB

Dazu ist nicht viel zu sagen. `tbrplent` und `tbcrent` sind in CWEB geschrieben und lassen sich daher mit `cweave` und `pdftex` (nicht pdfLaTeX!) vorzüglich in ein (hoffentlich) lesbares Format konvertieren.

8.2 Installation von CWEB

Die ist nahezu trivial. Bei [Knuth und Levy, 2001] kriegt man die Quellcodes, diese dann in einem vorher angelegten Verzeichnis entpacken, `make`, `make install`, fertig. Das System ist so simpel, daß eigentlich nichts schiefgehen kann. Wenn's nicht schon geschehen ist, `cwebmac.tex` in ein Verzeichnis schieben, wo TeX es findet.

`tex cwebman` erzeugt eine Anleitung. Ich möchte noch anmerken, daß auf dem CTAN hervorragende Pakete existieren, um CWEB mit LaTeX statt mit TeX zu benutzen und ein CWEB-Programm als Kapitel in ein bestehendes LaTeX-Dokument einzubinden.

8.3 Installation von tbook

Im Prinzip muß man nur die `.tar.gz`-Datei in einem temporären (möglichst leeren) Verzeichnis auspacken, `make`, `make install`, fertig. Aber ein Blick ins `Makefile` und eventuelles Anpassen einiger Verzeichnisse dort kann nicht schaden.

Aud er **tbook** Homepage findet man **tbook**, Saxon und xîndy auch als RPM. Eventuell sollte man das als erstes ausprobieren.

8.4 Installation von jpeg2ps

Das Programm findet man bei [jpeg2ps, 1999]. Für Windows muß man lediglich die bereits fix und fertige EXE-Datei in den Pfad kopieren. Für Linux `make` aufrufen und die entstandene ausführbare Datei `jpeg2ps` in den Pfad kopieren, z. B. nach `/usr/local/bin`.

Ich empfehle, jpeg2ps immer mit der Option `-h` zu benutzen, ansonsten könnte dvips zicken.

8.5 Installation vom Saxon

Den Saxon kann man sich bei [Saxon, 2002] holen. Er kommt entweder als "Instant Saxon" für Windows, dann muß man, soweit ich es verstanden habe, nur noch eine EXE-Datei aufrufen. Für Linux bekommt man ein paar `.jar`-Dateien[16], die man irgendwo hinkopiert. Dann erstellt man eine Datei `saxon`, macht sie mit `chmod guo+x saxon` ausführbar und drin steht z. B.

```
CLASSPATH=~/xml/LaTeXML/saxon651/saxon.jar:$CLASSPATH
export CLASSPATH
java -ms15000000 com.icl.saxon.StyleSheet $1 $2 $3 $4 $5 $6 $7
```

Natürlich muß man seinen eigenen Pfad reinschreiben. Das `-ms15000000` hat bei mir die Sache etwas beschleunigt, wenn's Ärger macht, weglassen. Ich benutze Java Version 1.3. Naja, monatelang 1.1.8, aber da dauert's exakt doppelt so lange. (Und ich Dummerchen hatte die ganze Zeit 1.3 auf der Platte.)

Für neuere Versionen des Saxon (neuer als 6.5.1) muß man eventuell dieses `com.icl.saxon.StyleSheet` durch was anderes ersetzen, siehe dazu die Saxon-Homepage unter "Installation"–"Changes".

Für tbook muß man den Saxon jedoch nie direkt aufrufen. Es werden drei Shell-Skripte installiert, die eine XML-Datei `buch.xml` in die drei Endformate umwandeln:

```
tbtolatex buch
tbtohtml buch
tbtodocbk buch
```

(Für die Umwandlung nach DocBook siehe Abschnitt 8.11 auf Seite 56.) Eventuelle Parameter können auch noch folgen, wie z. B. hier:

```
tbtolatex buch assume-os='windows' ...
```

Dazu später mehr. Jeder vernünftige XSLT-Prozessor erlaubt solche Parameter, allerdings haben andere eventuell eine abweichende Syntax.

8.6 Installation von xindy unter Linux

Da ich die Installation von xindy unter Windows einigermaßen geschafft habe (ich habe sie aber wohl erfolgreich aus meinem Gedächtnis gestrichen), gehe ich davon aus, daß die Linux-Installation mehr Mühe bereitet. Daher hier ein paar Anmerkungen dazu. Vieles gilt aber auch für Windows.

Man benötigt *zwei* Teerbälle (Endung `.tar.gz`, unter DOS/Windows auch `.zip`). Der eine enthält die plattform-unabhängigen Teile des Programms, der andere die plattform-spezifischen. Letzterer hat dann irgendein System im Dateinamen, z. B. `xindy-2.1-ix86-linux-glibc21.tar.gz`, ersterer eben nicht, hier ist `xindy-2.1.tar.gz` die aktuellste Fassung. Erst mal die Dateien entkomprimieren, *aber noch nicht auspacken!*

Dann den plattform-unabhängigen Teil auspacken. Dadurch entsteht ein Verzeichnis `xindy`. Die plattform-abhängige Tar-Datei dort hineinschieben und *dort* auspacken. Jetzt das Unterverzeichnis `xindy-2.1` (oder wie auch immer) umbenennen in `current`.[17]

Wer einen Bug bereinigen möchte, der bei „*siehe*"-Verweisen auftreten würde, kann die Datei `current/src/tex2xindy.l` in einen Editor laden und dort die Zeile

```
<xref>{LEVEL}           { printf("\") (\""); }
```

[16] brauchen tut man, glaube ich, nur `saxon.jar`

[17] Laut xindy-Anleitung heißt das nicht schon `current`, damit man mehrere Versionen installieren kann. No comment.

durch

```
<xref>{LEVEL}          { printf("\" \""); }
```

ersetzen. (Dieser Bug ist schon gemeldet.)

Alles weitere findet in `binaries/<dingenskirchen>/` statt.

In `Makeindex.install` kann man noch verschiedene Pfade anpassen, die Vorgaben sind jedoch nicht schlecht. Es wandert halt alles nach `/usr/local/bin` und `/usr/local/lib`. In `Makeindex.platform` läßt sich noch Compiler-Name[18] u. ä. anpassen, aber auch das war bei mir (S.u.S.E. 7.2) nicht nötig.

Das gröbste ist geschafft. Man ruft nun

```
make all
```

auf. Wenn keine Fehlermeldungen erkennbar sind,[19] kann man dann – Root-Rechte vorausgesetzt – mit

```
make install
```

xîndy installieren und den ganzen Verzeichnisbaum, den man für die Installation angelegt hat, löschen. Die Dokus finden sich dann (mit Standardpfaden) in `/usr/local/lib/xindy/doc`.

Die LaTeX-/XML-Welt bräuchte endlich einen in C(WEB) oder Java geschriebenen Index-/ Glossar-Generierer, der mindestens die Funktionalität von xîndy hat. Oder xîndy kommt in die Standard-Distris.

8.7 Zum Stichwortprozessor xîndy

Es werden viel zu viele Dokumente ohne Stichwortverzeichnis veröffentlicht. Ein guter Index ist eine Kunst und kostet ein, zwei Tage Mühe, aber der Schweiß ist vorzüglich investiert. Das darf daher in der XML-Welt nicht fehlen. Ein Schreibwerkzeug ohne Index-Generierung ist unbrauchbar.

Normalerweise wird MakeIndex zur Erzeugung eines Stichwortverzeichnisses eingesetzt. Leider ist dieses Programm für alle nicht-englischen Sprachen ungeeignet. Es wird dennoch benutzt; man nimmt dann eben unkorrekte Sortierung, indirekte Eingabe der Umlaute (`"a`, `"o` ...) und andere Unannehmlichkeiten in Kauf.

Die (mir einzige bekannte) Alternative xîndy [xindy, 2001] beseitigt diese Probleme. Es hat folgende Vorzüge:

- Alle Sonderbuchstaben können verstanden werden, egal, auf welch verquere Art und Weise sie in der `idx`-Datei stehen,[20] in der LaTeX die Index-Einträge sammelt.
- Es kommt einigermaßen mit Unicode zurecht. (Nicht perfekt, aber erstaunlich gut; MakeIndex muß hier völlig passen.)[21]
- Die Sortier-Reihenfolge kann einfach und präzise angegeben werden.
- Man kann – neben den Grundbuchstaben – Sonderkategorien anlegen, z. B. für Symbole.

[18] xîndy selber ist wohl schon vorhanden, aber beispielsweise der Lex-Scanner `tex2xindy` muß noch compiliert werden.

[19] Das Makefile empfiehlt, jetzt `make testsuite` aufzurufen. Überflüssig, außerdem geht die Testsuite *aus Prinzip* schief und verwirrt daher nur.

[20] Beispiel: `\index{Grüße}` aus der LaTeX-Datei kommen als »`Gr\"u\IeC {\ss }e`« in der `idx`-Datei an.

[21] In diesem Zusammenhang mal ein dickes Lob an die Erfinder von UTF-8: Diese Kodierung ist so gut durchdacht, daß viele nicht-Unicode-Tools den Unicode recht annehmbar packen. Ohne es zu merken.

- Das Ergebnis-Layout kann extrem flexibel (beinahe *zu* flexibel) bestimmt werden, auch für HTML-Ausgabe.

Die Nachteile sind:

- Es ist in LISP geschrieben. Die Bedienung ist daher manchmal etwas seltsam, außerdem erschwert das die Portierung auf andere Plattformen und die Weiterentwicklung.
- Es ist in keiner der mir bekannten LaTeX-Verteilungen enthalten.
- Die Dokumentation ist schwer verständlich.
- Einige der sehr exotischen Sonderfunktionen (z. B. korrekte Behandlung von Bibelstellen wie 1. Mose, Kap. usw.) irritieren bei der individuellen Anpassung.
- Es gibt offenbar keine Möglichkeit, Nicht-ASCII-Zeichen durch eine ASCII-Escape-Sequenz darzustellen, also wie z. B. das `^^8f` von TeX. Dieser Nachteil ist doppelt lästig, da es einerseits eine Qual ist, eine Distribution mit Nicht-ASCII-Textdateien zu schnüren, und er andererseits bestimmt recht einfach auszumerzen wäre.
- Mit Verlaub – dem Projekt fehlt momentan wohl die rechte Promotion.

Dennoch lohnt sich die Installation schon für gewöhnliches LaTeX, falls man Texte auch mal in nicht-englischer Sprache verfaßt. Für ein XML-System ist es die einzige Wahl. Wenn xîndy einmal läuft, ist es bequem und zuverlässig.

Die Stylesheets produzieren, wenn man den Saxon benutzt, eine Batchdatei namens `makeidx`. Ruft man sie auf, wird xîndy mit den richtigen Parametern gestartet. Für die HTML-Fassung wird dabei das alternative Eingabefilter `tb2xindy` benutzt. Zusammen mit `tbhtml.xdy`, `tblatex.xdy` und `tbook.xdy` sorgt es für eine konsistente und hoch-qualitative Indexgenerierung sowohl in der HTML- als auch in der LaTeX-Fassung.

8.8 tbook und Windows

XML läßt sich genauso gut auch unter Windows benutzen, alle wesentlichen Tools und die allermeisten unwesentlichen wurden portiert oder sind sogar unter Windows entwickelt worden. **t**book wird unter Windows dieselbe Arbeit leisten wie unter Linux. Allerdings setzt das Änderungen im Makefile voraus, außerdem dürften einige der benötigten Hilfsprogramme nicht von Anfang an bei Windows dabei sein (Ghostscript & friends, p?mto* etc).

An einer wesentlichen Stelle verhält sich **t**book (hoffentlich) ganz manierlich: Mit dem Parameter `assume-os=windows` sollten die generierten Batchdateien die Endung `.bat` erhalten und Windows-konform sein. Beispielsweise wird aus „`cat`“ „`type`“. Es ist aber ungetestet. Besonders beim Einsatz der doppelten Anführungsstriche war ich unsicher.

8.9 XSLT-Transformation

(Im folgenden rufe ich den Saxon explizit auf, damit man besser sieht, was passiert. In der Praxis benutzt man `tbtolatex` und `tbtohtml`.)

Man bearbeitet seine XML-Datei, speichert sie ab und ruft zunächst die XSLT-Trafo auf. Beim [Saxon, 2002, wird zusammen mit dem XML-Parser Ælfred[22] verteilt], der allgemein als der beste XSLT-Prozessor gilt, ginge das mit

```
saxon -o buch-raw.tex buch.xml tblatex.xsl
```

[22] Alfred der Große, * um 848, König der Angelsachsen, hat viel geschrieben.

Das wandelt `buch.xml` in `buch-raw.tex` um, unter Benutzung des XSLT-Stylesheets `tblatex.xsl`. Um nach HTML umzuwandeln, gibt man

```
saxon -o buch.html buch.xml tbhtml.xsl
```

ein. (Handelt es sich bei `buch.xml` um eine XML-Datei einer anderen XML-Anwendung wie z. B. DocBook oder TEI, muß man natürlich die Stylesheets dieser jeweiligen XML-Anwendung übergeben.)

Den [Xalan, 2001] habe ich nicht ausprobiert, aber sein XML-Interpreter, genannt Xerxes[23], ist ein echter Ressourcenfresser, der mir viel Ärger gemacht hat.[24] Zumindest in der Java-Fassung.

Anfangs hatte ich immer den [XT, 2002] benutzt, aber davon muß man leider abraten. Der ist zwar der Vater aller XSLT-Prozessoren, aber wie das mit Vätern eben so ist – sie werden alt, und ich habe den Eindruck, daß der Autor James Clark den Staffelstab endgültig an Michael Kay, Autor des Saxons, abgegeben hat. (In jeder Hinsicht: Clark *war* Editor der XSLT Spezifikation, und Kay *ist* es.) Offiziell wird er aber noch weiterentwickelt, und zwar von Bill Lindsey. XT ist zwar sehr schnell, aber einige Dinge funktionieren einfach nicht. Von diesen Dingen benötige ich genau eines unbedingt, und zwar für das `interval`-Attribut beim `<psfrag>`-Element. Wer das auskommentiert, kann den XT benutzen.

Aber zurück zu den Umwandlungen mit dem Saxon. Wenn alles gutgeht, bleibt der Saxon still. Die HTML-Datei ist sofort benutzbar. Die LaTeX-Datei nicht; sie enthält noch die lästigen Unicodes. Zu deren Umwandlung siehe Abschnitt 9.1.3 auf Seite 62.

8.10 PDF-Erzeugung

Im Prinzip muß man für die PDF-Erzeugung bloß `pdflatex` anstelle von `latex` aufrufen. Nun hat man aber u. U. noch Grafiken im schönen, alten EPS-Format, eventuell mit `<psfrag>`-Elementen. Das kann pdfLaTeX nicht verarbeiten.

Wenn man jedoch den Saxon benutzt, werden ja dessen Erweiterungen[25] verwendet, um eine Batchdatei `makepdfs` anzulegen, die dann die PDFs erzeugt.

Zu diesem Zweck wird eine LaTeX-Datei namens *`Dokumentname`*`-image-gallery.tex` von `tbtolatex` generiert. Dahinter verbirgt sich eine Art Sparfassung des Dokuments, nämlich nur die Grafiken, ohne Kopf- und Fußzeilen, und `makepdfs` nutzt diese Datei, um mittels dvips und ps2pdf aus allen EPS-Dateien PDF-Geschwister zu stricken.

Was ich nicht verstehe: Warum stellt der Acrobat Reader eigentlich schon fünf Versionsnummern lang Bitmap-Buchstaben armseliger dar als jeder noch so **@§%$**[26] DVI-Viewer?

Die Batchdatei `makewebs`, die JPEGs und PNGs für die HTML-Fassung erzeugt, macht übrigens auch von dieser Gallerie-Datei Gebrauch.

8.11 Umwandlung nach DocBook

Wie schon gesagt, wandelt

```
tbtodocbk buch
```

[23] Xerxes I., * um 519 v. Chr., altpersischer König, ziemlicher Raufbolzen.

[24] Ich hatte nämlich mal einen zu üppigen Java-`CLASSPATH` und habe so ungewollt im Saxon den Ælfred durch den Xerxes verdrängt. (Klingt das nicht klasse? Und in GNU ist der Bison der Yacc . . .)

[25] die glücklicherweise zu einem großen Teil in die nächste XSLT-Spezifikation einfließen; der Autor des Saxon, Michael Kay, ist dort Editor;

[26] gallische Schimpfworte, die ich nicht übersetze

die **tbook**-Datei `text.xml` in eine DocBook XML-Datei um, die `buch-db.xml` heißt. Diese Datei ist gültiges DocBook 4.2.

Um eine bequeme Umwandlung nach RTF hinzubekommen (d. h. ohne daß ich das RTF-Format erlerne), sah ich nur eine Möglichkeit: Eine XSLT-Trafo nach DocBook und mit dessen fix und fertigen Tools eine Umwandlung nach RTF. Dieser Weg hat den weiteren Vorteil, daß es grundsätzlich eine gute Idee ist, die eigene XML-Anwendung an den Quasi-Standard DocBook anzustöpseln.

Die Umwandlung nach DocBook ist kein Problem. Man nimmt sich die Umwandlung nach XHTML und modifiziert sie einfach. Für viele Tags ist das ein Suchen–Ersetzen. Vieles kann man rausschmeißen, weil DocBook bereits für Inhaltsverzeichnis, Stichwortverzeichnis etc. sorgt. Eigentlich stößt man nirgends auf größeren Widerstand.

Das gilt übrigens ganz generell. Egal, in welches XML-Format man konvertieren möchte, man kann immer die XHTML-Umwandlung als Schablone benutzen. Da irgendwie alle Dokument-DTDs miteinander verwandt sind, läuft es für viele Elemente auf ein Suchen–Ersetzen hinaus. (Beispielsweise benutzt MS Word mittlerweile eine Abart von XHTML als Eingabe-Dateiformat. Es dürfte nicht schwer sein, das zu bedienen.)

8.11.1 RTF-Erzeugung

Die ist für mich recht spannend, weil ich viele Kollegen und Freunde habe, die Word benutzen und denen ich dann meine Texte zur Verfügung stellen könnte.

Unter Linux kann man einen Aufruf wie `docbook2rtf` versuchen, um die Umwandlung hinzukriegen. Letztlich kommen dafür Jade und DocBooks DSSSL-Stylesheets zur Anwendung. Bei mit hat `docbook2rtf` hoffnungslosen Unsinn gemacht.

An Ende habe ich dann Jade selber aufgerufen, und zwar prinzipiell so:

```
jade -E0 -t rtf -d /usr/[...]/print/docbook.dsl \
  /usr/share/sgml/xml.dcl buch-db.xml
```

`buch-db.xml` war dabei die DocBook-Datei. Jade erzeugt dann `buch-db.rtf`.

Das Ergebnis ist nicht über alle Maßen herrlich, aber sehr brauchbar. Ein Inhaltsverzeichnis wird freundlicherweise automatisch generiert. Allerdings gingen Indexeinträge und kompliziertere Formeln bei der Umwandlung verloren. Beim Index kann ich das nicht ganz verstehen, offenbar ist Jade hier der limitierende Faktor, obwohl es an sich sehr einfach sein müßte.

Für einfache Formeln wie $E = mc^2$ versucht **tbook**, das Markup irgendwie in DocBook nachzubilden. Das Ergebnis hat mich sehr angenehm überrascht. Die allermeisten Formeln sind praktisch unbeschadet in Word angekommen. Bei Gleichungs-Arrays jedoch werden nur die Symbole hintereinander geschrieben. Das muß dann nachbearbeitet werden. Bald wird DocBook jedoch MathML verdauen können.[27]

Abgesetzte Gleichungen werden in 1 pt kleinerer Schrift geschrieben, weil sie intern als `<blockquote>`-Element realisiert sind, und solche Blockquotes eben etwas kleiner gedruckt werden. Das entspricht wohl dem Geschmack Norman Walshs[28], aber nicht meinem, und wird auch normalerweise nicht so gehandhabt. Was soll's. Außerdem ist für alle Absätze des Dokuments die Silbentrennung ausdrücklich abgeschaltet. Zwar ist auch alles linksbündig, dennoch sollte man sie in Word unter Format–Absatz wieder einschalten.

Grafiken, Tabellen, Kreuzverweise, nicht-englische Sprache und Literaturverzeichnis waren kein Problem. Für die Grafiken werden die Web-Versionen benutzt. Bei Bedarf sollte man auf die Postscript-Versionen umsteigen.

---

[27] **tbook** wäre über den XSLT-Parameter `docbook-equations` bereits jetzt in der Lage, DocBook mit MathML zu erzeugen.

[28] der Maintainer von DocBook

8.12 Ein Glossar

Ich mag Glossare nicht, ich empfinde sie als “poor man’s index”, aber diese Erweiterung ist mit xindy hervorragend hinzukriegen. Man muß dafür allerdings den Scanner `tb2xindy` ein wenig anpassen, und zwar nach dem Muster, daß `tex2xindy` vorzeichnet.[29]

Und natürlich müssen die Stylesheets und die DTD ergänzt werden.

8.13 BIBTEX – für LATEX und HTML

Wie man BIBTEX für ein LATEX-Dokument einsetzt, ist klar. Entsprechend simpel fällt die Datei `makebib` für den LATEX-Fall aus – sie ruft lediglich BIBTEX auf. Interessant ist jedoch die Frage, wie man ein Literaturverzeichnis in eine HTML-Datei bringt.

8.13.1 Möglichkeit Eins: XML

Mein ursprünglicher Ansatz war, meine ganze Bibliographie statt in eine BIBTEX-Datei in eine XML-Datei zu schreiben. Grundlage hierfür war die DTD des BIBTEXML-Projektes. Die Idee ist, daß BIBTEXML einen leistungsfähigen Parser bereitstellt, mit dem man bestehende Altlasten-`bibs` in das neue XML-Format umwandelt, dort noch ein wenig nachbessert, und im folgenden nur noch die XML-Datei pflegt. BIBTEXML garantiert dann für eine verlustfreie temporäre Umwandlung ins BIBTEX-Format, nur um BIBTEX aufzurufen.

Für die HTML-Fassung hätte ich dann die XML-Datei direkt eingelesen und interpretiert. Nachteil: Ich muß in XSLT BIBTEX nachbilden, was natürlich eine irre Arbeit ist, und angesichts der vielen BIBTEX-Stilen, die so rumfliegen, wahnsinnig unflexibel ist.

8.13.2 Möglichkeit Zwei: BIBTEX

Ich habe dann doch etwas anderes gemacht, zumal die Leute vom BIBTEXML-Projekt wohl momentan nichts tun: Ich habe mit dem Paket `custom-bib` von Patrick Daly nicht nur einheitliche Stile für die vier Sprachen für die LATEX-Fassung erzeugt, sondern parallel dazu vier HTML-Stile. `custom-bib` kann nämlich auch HTML erzeugen, wenn auch nur rudimentär, und mit kleinen Änderungen an der Hauptdatei `merlin.mbs` konnte ich `custom-bib` überreden, XML-Code zu erzeugen.[30]

Nachteil dieser Methode: Das erzeugte XML-Literaturverzeichnis ist voll von LATEX-Befehlen, außerdem hat BIBTEX den Bug, nach 78 Spalten die Zeile mit einem `%`-Zeichen abzubrechen. (Ein Bug ist das deshalb, weil es sich nicht abschalten läßt.) Ein nachgeschalteter Filter namens `bibfix` behebt all diese Probleme. Einfache Formeln werden artig mit `<m>...</m>` eingeklammert.

Man kann `bibfix` an eigene typische LATEX-Konstrukte in BIBTEX-Dateien anpassen. Glücklicherweise muß man in puncto Umlaute kaum etwas tun, da die gängigen Varianten, wie Umlaute eingegeben werden, erkannt werden (z. B. `"G{\"o}del"`). Ganz toll ist es natürlich, wenn man schon in der Vergangenheit für BIBTEX Latin-1-Dateien benutzt hat. `bibfix` ist ein sehr einfacher Flex-Scanner, und seine Anpassung ist, falls überhaupt nötig, sehr rasch zu bewerkstelligen, wenn man ihn sich ein wenig anschaut.

Die Literaturverzeichnisse, die so in den Browser kommen, sehen sehr gut aus. Außerdem ist dieses Verfahren recht robust. Wenn man fortan ein wenig vorsichtiger mit LATEX-Befehlen in der BIBTEX-Datei ist, gibt’s keinen Grund, BIBTEX in der HTML-Welt fallenzulassen.

---

[29] `tb2xindy` müßte neu und besser geschrieben werden. Es ist recht anfällig.

[30] Diese kleinen Änderungen sind in Form des Patchfiles namens `merlinht.dif` bei tbook dabei. Im `Makefile` findet man den `patch`-Aufruf, um sich ein eigenes `merlinht.mbs` zu basteln.

8.14 Die LaTeX-Installation

... sollte prächtig ausgestattet und vom allerneuesten sein, ansonsten macht's ja aber auch unter gewöhnlichem LaTeX keinen Spaß. Es ist sicher müßig, alle Pakete aufzuzählen, die meisten dürften bei den großen Distris ohnehin dabei sein. Der Rest ist auf dem CTAN unter `macros/latex/contrib` zu finden. Vorsicht: Bei pdfTeX und dem `graphics`-Paket ist es unbedingt notwendig, die neuesten Versionen zu haben. (Auch wenn die heiße Phase von pdfTeX allmählich vorüber sein sollte.)

Es sollten neben US Amerikanisch (`USenglish` und `english`) und den vier Deutsch-Varianten auch Englisch (`UKenglish`), Französisch (`french`), Italienisch, Spanisch und Katalanisch bei Babel installiert sein. Aber es ist selbstredend nur zwingend, die Sprachen zu installieren, die man wirklich benutzt.

Es muß ferner vorausgesetzt werden, daß die Textschriften im T1-Encoding vorliegen und auch ein vernünftiges TS1-Encoding bereitgestellt wurde. Vernünftig heißt, daß man neben den Zeichen, die im Adobe-Standard-Encoding stehen, auch für ein schmuckes Euro-Logo gesorgt hat.

Der Dingbats-Zeichensatz muß zur Verfügung stehen, beziehungsweise genauer: `\ding{38}` sollte keinen Unsinn liefern.

Ich habe noch nie die Koma-Klassen benutzt. Der Grund ist nicht eine gewisse Abneigung, sondern schlicht, daß ich bislang noch keinen Bedarf dafür hatte. Ich muß jedoch zugeben, daß Texte, die über den Index oder das Literaturverzeichnis gedruckt werden, mit Koma einfacher zu realisieren sind. Eine Verarbeitung des XML-Outputs mit Koma ist genauso gut möglich und eine Anpassung sollte nahezu trivial sein, probiert habe ich's aber nicht.

Last but not least: `jpeg2ps` und `ps2pdf` sind nette kleine Helferlein, auf die ich nicht mehr verzichten kann.

8.15 Generierte Dateien

Es werden von den Batchdateien und den Stylesheets eine Vielzahl von Dateien generiert, unter anderem mehrere für jede Grafik. Die Tabelle 3 auf der nächsten Seite gibt einen Überblick darüber. An sich muß das einen nicht kümmern, aber es wäre doch ärgerlich, wenn eigene Dateien überschrieben würden.

Die Ursprungs-Bilddateien, also die JPEGs und EPSes, bleiben bei allem natürlich unangetastet. Es sei denn, man hat eine Bitmap namens `toll.jpg` und eine namens `toll-web.jpg`. Dann wird die HTML-Version von `toll.jpg` die PDF-/PS-Version von `toll-web.jpg` überschreiben. Aber ein mögliches `-web`-Anhängsel ist die einzige wenigstens einigermaßen nicht-verschwindende Gefahr.

9 Spezielle Probleme mit XML und XSLT

9.1 Das Problem des Unicodes in LaTeX

In einem XML-Dokument dürfen alle Zeichen des Unicodes vorkommen. Dabei gibt es drei Möglichkeiten, wie ein Zeichen in der Datei vorkommen kann:

1. Das Zeichen wird direkt eingegeben. So macht man es natürlich meistens, sonst würde die Datei ja unlesbar. Wichtig ist jedoch, daß man Zeichen jenseits von Latin-1, also z. B. das Euro-Symbol, Griechisch, Kyrillisch, Chinesisch, aber auch die ganzen Mathe-Symbole, ebenfalls *direkt* eintippen darf. Dafür muß die Datei in einem Unicode-fähigen Encoding abgespeichert werden, das ist meist UTF-8. Außerdem muß der Editor mitspielen.

Datei	Bedeutung
*.eps	Von makeepss erzeugte EPS-Versionen aller JPEG-Bitmaps, für die PS-Fassung des Dokuments.
{*Name*}-image-sizes.xml	Die Bitmap-Dimensionen für das <img>-Tag der HTML-Ausgabe.
{*Name*}-image-gallery.tex,.dvi,.log	Datei mit allen Bildern und sonst nichts, zur Umwandlung in EPS-Fassungen zur Weiterverarbeitung (PDF, JPEG, PNG).
{*Name*}-image-gallery-raw.tex	Dieselbe Datei, nur daß die Unicodes hier noch nicht aufgelöst sind.
{*Name*}-eqn-gallery.tex, .dvi,.log	Datei mit allen Formeln und sonst nichts, zur Umwandlung in PNG-Fassungen.
{*Name*}-eqn-gallery-raw.tex	Dieselbe Datei, nur daß die Unicodes hier noch nicht aufgelöst sind.
*-web.jpg,.png	Von makewebs erzeugte Bitmaps für die HTML-Ausgabe.
eqn-*.png	Von makeeqns erzeugte Formel-Bitmaps für die HTML-Ausgabe.
*-image-size.txt, {*Name*}-image-sizes.xml, {*Name*}-eqn-image-sizes.xml	Von makeeqns und makeeqns erzeugte Dateien zur Generierung von Höhen-/Breitenangaben bei Grafiken.
*.pdf	Von makepdfs erzeugte PDFs für alle Nicht-Bitmaps, für die PDF-Fassung des Dokuments.
{*Name*}.idx	Index-Datei von xindy für LaTeX.
{*Name*}-idx.xml	Index-Datei von makeidx für xindy.
{*Name*}-ind.xml	Index-Datei von xindy für tbtohtml.
{*Name*}-bib.aux	LaTeX-aux-Datei, die aber nur die Literaturverweise enthält, für makebib in der HTML-Fassung.
{*Name*}-bib.bbl	Temporäre Datei, die BIBTeX erzeugt, und die sofort wieder gelöscht wird.
{*Name*}-bib.xml	Literaturverzeichnis für die HTML-Fassung.
*-image-comment.txt	Bild-Kommentare zur Einbettung in PNGs mittels pnmtopng.
makeepss, makepdfs, makewebs, makeeqns, makeidx, makepage, makeclean	Batchdateien, die tbprepare ins aktuelle Verzeichnis kopiert.
*-web.pnm	Bilddateien, die während makewebs temporär angelegt werden.
*-for-conv.ps	Postscripts mit jeweils einer Grafik, die während makewebs und makepdfs temporär angelegt werden.

Tabelle 3: Vom XSLT-Prozessor und den Batchdateien generierte Dateien. {*Name*} bezeichnet den Namen des Dokuments. „*“ bezeichnet den Namen irgendeiner Grafik.

2. Das Zeichen wird mit seiner Unicode-Nummer bezeichnet, z. B. `…` für eine Ellipse „...“. Das geht mit jedem Datei-Encoding.

3. Das Zeichen wird als eine Namens-Entität angesprochen, also z. B. `…` für die Ellipse. Was die Entität bedeutet, muß dann allerdings in der DTD oder dem Dokument ausdrücklich gesagt werden, da ist nichts vordefiniert.

Daß da nichts vordefiniert ist, könnte man ausnutzen: So könnte ich dafür sorgen, daß, wenn die XSLT-Trafo LaTeX-Output erzeugt, aus `…` der String „`\dots{}`“ wird, und damit das ganze von LaTeX verstanden werden kann. Damit dieser Trick jedoch klappt, muß man außer bei Latin-1-Codes die Zeichen durch ihre Namens-Entitäten ansprechen, d. h. die beiden ersten Möglichkeiten sind tabu.

Erschwerend kommt hinzu, daß einige Zeichen wie „`&`“ oder „`%`“ in LaTeX eine besondere Bedeutung haben. Diese kann man daher auch nicht direkt eingeben, sondern müßte besondere Entitäten benutzen.

9.1.1 Mögliche Auswege

Der Ansatz mit solchen Namens-Entitäten für diese „pathologischen“ Zeichen wäre aber ein wenig ärgerlich, zumal der GNU Emacs seit Version 21 mit dem Unicode zurechtkommt. Es wäre schon klasse, wenn man nicht nur `&`, `#` oder `$`, sondern auch z. B. ‘Ę’, ‘fi’, ‘ª’ oder das Euro-Symbol und sogar Mathematik wie δ, $\leq$, $\int$ oder Σ direkt so im Texteditor sehen könnte. Außerdem machen diese mißbrauchten Entitäten das Schreiben der XSLT-Trafos zur Qual.

Es gibt zwei Lösungen für dieses Problem:

- Man schreibt ein Programm, das vor oder nach der Umwandlung XML → LaTeX alle pathologischen Zeichen durch gültiges LaTeX ersetzt.

- Man bringt LaTeX bei, diese Codes *doch* zu verstehen.

Der erste Weg ist möglicherweise solider und umfassender, während der zweite Weg eleganter ist. Man kann sich dabei von XMLTeX [XMLTeX, 2000] oder dem Unicode-Paket [unicode-Paket, 2000] inspirieren lassen.

Ich habe trotzdem den ersten Weg beschritten.

9.1.2 Ein zwischengeschalteter Unicode-Filter

Ich habe lange hin und herüberlegt, wie man mit einem extra Programm LaTeX die Unicodes beibringen kann. Das Problem ist viel vertrackter, als es zunächst scheint, nicht wegen der Unicodes, sondern wegen LaTeX’s aktiven Zeichen: Ein `$` in der XSLT-LaTeX-Ausgabe kann von einem Text-Knoten des Autors stammen, z. B. ein Dollar-Preis, dann muß es zu einem `\$` werden. Oder es wurde von XSLT *generiert*, um eine Formel abzugrenzen; dann bleibt es, wie es ist. Wie soll ich das aber unterscheiden?

Ich hätte alle aktiven Zeichen in der XSLT-Datei mit einem Präfix versehen können. Das macht die schöne Datei aber völlig unleserlich. Also habe ich zwei Unicode-Zeichen geopfert, nämlich die beiden Steuerzeichen “start of string” und “string terminator” (`0x98` = 152 und `0x9c` = 156), um damit Text-Knoten zu umklammern. So erkennt der Postprozessor, ob er sich in *originalem* oder *generiertem* Material befindet. Ich weiß nicht, wofür diese Steuerzeichen eigentlich gedacht sind, aber außer meinen eigenen Programmen kriegt die sowieso keiner zu Gesicht.

Dieses Filter-Programm nimmt sich dann die XSLT-Ausgabe (die fast LaTeX ist) und ersetzt alle Unicodes durch Latin-1 bzw. LaTeX-Kommandos, und ersetzt aktive Zeichen in Text-Knoten durch die escapierten Versionen. Das kann dann so zu LaTeX gehen. Siehe dazu Abschnitt 9.1.3 auf der nächsten Seite.

Den zweiten Weg, also LaTeX den Unicode beizubringen, habe ich nie ausprobiert. War wahrscheinlich auch besser so. Meine LaTeX-Dateien sind (im gegensatz zu denen, die beispielsweise PassiveTeX bekommt), durchsetzt von aktiven Zeichen, die auch als solche interpretiert werden müssen. Das wäre also, falls überhaupt möglich, eine heillose Fummelei geworden.

9.1.3 Unicode-Filter `tbrplent`

Mit

```
saxon buch.xml tblatex.xsl | tbrplent > buch.tex
```

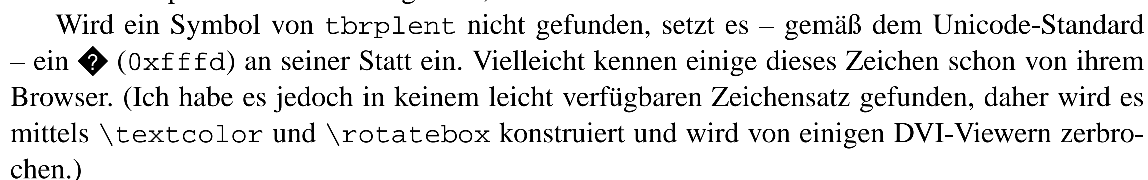
erhält man `buch.tex`, die man dann LaTeX übergeben kann. Das nahezu triviale Filter `tbrplent` ist selbstverständlich in der schönsten Programmiersprache der Welt geschrieben: CWEB [Knuth und Levy, 2001]. Dieses Filter benutzt bei der Umwandlung die Datei `tbent.txt`, in die man ruhig einen Blick werfen kann, sie ist recht gut lesbar.

Die Abdeckung des Unicodes ist eine schwierige Angelegenheit. LaTeXs Zeichenbehandlung erscheint gegenüber dem recht homogenen Unicode wie ein chaotischer Hühnerhaufen. Hier ein bißchen Buchstaben, da ein bißchen aktive Zeichen, dort ein Batzen IPA Lautschrift, ein Häppchen Kyrillisch, ein paar Dingbats und viel Mathematik. Für all diese Konstrukte sind in LaTeX u. U. völlig unterschiedliche Herangehensweisen, insbesondere verschiedene Pakete und Fonts, nötig.

Die Dingbats werden vom Unicode-Filter `tbrplent` gesondert behandelt und in `\ding{...}`-Befehle umgesetzt. Das geschieht aber nur, weil die relative Position der Dingbats untereinander im Dingbat-Zeichensatz und im Unicode identisch ist.[31] Für alle anderen Fälle wäre das nicht ökonomisch, hier ist eine explizite Auflistung in `tblatex.ent` nötig. Ein Muster dafür wird ganz unten in der Datei gegeben. Auf diese Weise kann man dann z. B. die IPA-Lautschrift verfügbar machen.

Es gibt noch viele Unicode-Glyphen, sich sich sehr leicht einbauen lassen, z. B. ist „`(a)`“ ein eigenes Zeichen! (Pos. `0x2496`) Ich habe aber keine Lust, all diese Zeichen einzutippen, zumal ich weder ihre Bedeutung, noch die Notwendigkeit ihrer Eigenständigkeit erfassen kann.

Man *kann* so auch etwa Kyrillisch oder Neugriechisch einbauen, das lohnt sich dann aber nur für verstreute Zitierungen (z. B. Eigennamen) oder wenn man bloß einzelne Buchstaben braucht. Ansonsten kommt man wohl nicht um eine Anpassung des ganzen Systems an die Unicode-TeX-Variante Omega (bzw. Lambda) herum. Dort sind auch in den nicht-lateinischen Alphabeten Dinge wie rechts-nach-links-Druck, saubere Trennung und typographische Feinheiten garantiert. Außerdem muß man dann nicht für jeden Glyphen den Font umschalten. Damit habe ich mich allerdings noch überhaupt nicht auseinandergesetzt, weil ich es nicht brauche.



Wird ein Symbol von `tbrplent` nicht gefunden, setzt es – gemäß dem Unicode-Standard – ein � (`0xfffd`) an seiner Statt ein. Vielleicht kennen einige dieses Zeichen schon von ihrem Browser. (Ich habe es jedoch in keinem leicht verfügbaren Zeichensatz gefunden, daher wird es mittels `\textcolor` und `\rotatebox` konstruiert und wird von einigen DVI-Viewern zerbrochen.)

Man kann nun in einer Art Kraftakt den ganzen Unicode abklappern und schauen, wie man die Zeichen in LaTeX nachbildet. Ich habe das erstmal ganz unverbindlich mit den Zeichen getan, die ich in TeX, LaTeX, AMS, `wasysym` und `marvosym` gefunden habe. Neue Zeichen werden nach folgendem Muster in `tblatex.ent`[32] eingefügt:

```
<!ENTITY uc--x262f          "\rotateright{\Yingyang}">
```

Am `"uc--??"` erkennt `tbcrent`, daß ein LaTeX-Makro folgt, das eben diesen Unicode ersetzen soll. Denselben Effekt hätte

[31] Und ich bleibe dabei: ✠ ist *kein* Malteser Kreuz, wie der Unicode behauptet.

[32] beziehungsweise in `tbookdtd.dtx`, weil diese Datei die Datei `tblatex.ent` enthält

```
<!ENTITY yingyang          "\rotateright{\Yingyang}">
<!ENTITY yingyang          "&#x262f;">
```

Wäre eben nur mehr Tipparbeit.[33] `\rotateleft` und `\rotateright` sind eigene Makros, die das Symbol um 90° in die jeweilige Richtung drehen. Das herkömmliche `\reflectbox` ist auch manchmal hilfreich. Zwei Dinge sind zu beachten: Endet die LaTeX-Makro-Sequenz mit einem parameterlosen Makro, muß noch ein {} eingefügt werden:

```
<!ENTITY euro    "\texteuro{}" > <!--euro sign, U+20AC NEW -->
```

Alles, was so nur in Formeln gilt, muß in `$...${}` eingefaßt werden:

```
<!ENTITY Gamma             "$\Gamma${}">
```

Schließlich bleibt noch die Möglichkeit einer expliziten Unterscheidung:

```
<!ENTITY middot            "\ifmmode\cdot\else\textperiodcentered\fi{}">
```

Nach dem Hinzufügen neuer Zeichen muß man `tbents.txt` aktualisieren, indem man `tbcrent` aufruft.

9.2 Die Behandlung von Whitespace

Wenn man XML-Dateien mit XSLT umwandelt, kann man dabei in der XSLT-Datei angeben, wie Leerraum (Leerzeichen, Zeilenumbrüche, Tabstops) zwischen Elementen behandelt werden soll. Das heißt: Ein Element ist zu Ende, und bevor das nächste beginnt, kommen beispielsweise ein paar Leerzeichen.

Ich mache damit folgendes: Dieser Leerraum wird stets völlig ignoriert, außer in Umgebungen, die sowohl *`Text`*, als auch *`Inline-Element`* enthalten können.

Wenn ich also schreibe

```
<em>Hallo</em> <visual markup="bf">Welt</visual>!
```

wird daraus wie gewünscht „*Hallo* **Welt**!“ und nicht etwa „*Hallo***Welt**!“. Das ist nicht ganz selbstverständlich, denn beispielsweise zwischen allen MathML-Dokumenten *muß* der XSLT-Prozessor den Leerraum löschen, sonst kommt kein richtiges LaTeX heraus.

Wahrscheinlich ist mein Gerede hier überflüssig. Es sollte meistens das Erwartete passieren. Ich erwähne es vor allem deshalb, weil Leute, die XSLT-Dateien schreiben, es beachten müssen, und es nicht ganz offensichtlich ist.

Der Leerraum zwischen Element und Text-Inhalt, der auch nicht-Whitespace enthält, wird übrigens nie gelöscht. Also ergibt

```
H<em>␣allo</em>␣Welt
```

stets ein „H␣*allo*␣Welt“.

9.2.1 Leerzeilen im Fließtext

Leerzeilen im Fließtext werden immer ignoriert, sie sind jedoch verboten. (Es wird eine Fehlermeldung ausgegeben.) Der Grund ist einfach: In LaTeX beispielsweise beginnen sie einen neuen Absatz, obwohl kein Absatz durch das XML-Markup begonnen wurde. Das verstößt gegen die Spielregeln und ist außerdem unnötig. *Zwischen* Blockelementen darf man Leerraum einfügen, wie man lustig ist.

[33] Aber so funktioniert das ganze System standardmäßig: Alle `*.ent`-Dateien bilden Entitätsnamen auf Unicodes ab, und `tblatex.ent` bildet die Namen auf LaTeX-Ersetzungen ab. `tbcrent` macht dann nicht viel mehr als diese beiden Mappings zum dritten Mapping Unicode $\rightarrow$ LaTeX zu verknüpfen. Hat „historische“ Gründe.

9.3 XML-Syntax vs. Text-Syntax

XML ist (momentan noch) eine Spielwiese für Puristen. Das heißt, daß XMLer häufig auf dem Standpunkt stehen, es solle doch bitte *alles* XML sein. Zwar ist einzusehen, daß man die XML-Syntax bevorzugt benutzen sollte, aber ich meine, daß dies in jedem Einzelfall begründet werden muß. So sind z. B. auch XSLT, XSL:FO oder XML Schema[34] reine XML-Formate, was mehr schadet als nützt. Die Grammatiken werden dadurch sehr umständlich, obwohl sie verhältnismäßig wenig leisten müssen.

Dieser Purismus zeigt sich auch an einer anderen Front: Wenn sich in einem Dokument ein Element weiter aufsplitten läßt, wird es vom Puristen grundsätzlich in *Elemente* aufgesplittet, ein `<Autor>` also z. B. in einen `<Vorname>`n und `<Nachname>`n. Das macht XML-Dokumente sehr, sehr umständlich. Siehe MathML oder Abschnitt 11.1 auf Seite 70.

Anderes Beispiel: eine Tabellenzeile. In HTML

```
<tr>
  <td>A</td>
  <td>B</td>
  <td>C</td>
  <td>D</td>
</tr>
```

in LaTeX schlicht `A&B&C&D\\`. Ich habe mich daher an folgenden Grundsatz gehalten:

> *Erlaubt ist nicht nur das, was XML ist, sondern alles, was in vertretbarem Aufwand von einem XSLT-Prozessor verarbeitet werden kann.*

Und das ist viel mehr. (Gültige XML-Dateien bleiben meine Dokumente in jedem Falle.) Der „vertretbare Aufwand“ ist dabei natürlich ein dehnbarer Begriff. Aber solange noch genügend Stack da ist, ist auch noch genügend Hoffnung da.

In folgenden Elementen benutze ich eine nicht-XML-Syntax: `<multipar>`, `<m>`, `<dm>`, `<ch>`, `<srow>`, `<unit>` und `<author>`.

9.3.1 Das <multipar>-Element

Ein besonderes Element ist dabei `<multipar>`. Es enthält Absätze, die statt durch `</p><p>`, durch einfache „`*`“ voneinander getrennt sind. Das allein wäre noch nicht so schwer zu realisieren; allerdings lassen sich auch sämtliche Inline-Elemente, z. B. `<em>` oder auch `<math>`, an beliebiger Stelle einstreuen. Die XSLT macht daraus XHTML aus vielen `<p>`s, das garantiert wohlgeformt ist. Ich benutze dabei nämlich keine Tricks wie

```
<xsl:text disable-output-escaping="yes">...
```

die die Sache zwar wesentlich einfacher, aber auch gefährlicher gemacht hätten.

9.3.2 <m>, <dm>, <ch> und <unit>

Diese vier Tags fügen einfache mathematische Gleichungen in das Dokument ein. Sie benutzen dabei eine nicht-XML-Syntax, nämlich eine vereinfachte und leicht modifizierte LaTeX-Syntax. Das Problem ist ja, daß MathML extrem viel Schreibarbeit erfordert. Insbesondere für kleine Mathe-Fetzen ist das dumm. Wenn ich z. B. in LaTeX nur

```
Das Doppelte der Variablen~$x$.
```

[34] eine moderne Variante einer DTD

schreiben würde, müßte ich dafür in XML

```
Das Doppelte der Variablen <math><mi>x</mi></math>.
```

schreiben, was natürlich albern ist. Ich habe mich daher für folgendes entschieden:

Mit `<m>` ("inline math") kann ich `<m>x</m>` schreiben. Das ist schon wesentlich kürzer. Was Makros angeht, dürfen `\frac`, `\sqrt` und `\text` benutzt werden. (Bei `\sqrt` ist das optionale Argument erlaubt.) Ein Index muß immer *vor* einem eventuellen Exponenten stehen, also stets `a_b^c` und nie `a^c_b`. Vergrößerbare Klammern werden nicht mit `\left` und `\right` ausgedrückt, sondern mit `{(...)}`. Also wird aus

```
<m> {( \frac12 ]} </m>
```

das folgende:

$$\left(\frac{1}{2}\right]$$

Für `\int`, `\alpha` etc. gibt man natürlich direkt die MathML-Entitäten `∫` und `α` ein. Die XSLT-Trafo wandelt diese Formeln dann nach Standard-LaTeX um, beziehungsweise, für die HTML-Ausgabe, in MathML.

Mathematische Akzente werden einfach direkt (ohne Leerschritt) vor das betreffende Zeichen geschrieben. Ist das Zeichen dabei ein `{...}`-Ausdruck, wird, wenn's geht, automatisch eine `\wide...`-Version des Akzentes benutzt.

Standardfunktionen werden einfach hingeschrieben, ohne Backslash. Bei Zahlen muß man ein wenig aufpassen: Man kann ruhig `3,2` schreiben, das Komma wird als Dezimaltrennzeichen erkannt. Soll es das nicht, muß man `3, 2` eingeben. Selten (z. B. nach Akzenten, die nämlich stärker binden) muß man den Interpreter drauf stoßen, daß es *eine* Zahl ist. Dann ist `{3,2}` einzugeben.

Hier mal ein Beispiel mit so ziemlich allem, was möglich ist:

```
<math><mover accent="true"><mrow><mn>1</mn><mo>-</mo><msub><mi>x</mi>
<mrow><mtext>eff</mtext></mrow></msub></mrow>
<mo>&Hat;</mo></mover><mo>&ne;</mo><mrow><mo>(</mo>
<munderover><mo>&int;</mo><mn>0</mn><mo>&infin;</mo></munderover>
<mi>sin</mi><mo stretchy="false">(</mo><mover accent="true">
<mi>x</mi><mo>~</mo></mover><mo stretchy="false">)</mo>
<mfrac><mrow><mroot><mrow><mn>1</mn><mo>/</mo><mi>e</mi></mrow>
<mn>3</mn></mroot></mrow><mi>&beta</mi></mfrac><mi>d</mi><mi>x</mi>
<mo>)</mo></mrow><mo>&ne;</mo><munder><mi>lim</mi><mrow><mi>x</mi>
<mo>&rarr;</mo><mo>&infin;</mo></mrow></munder><mfrac><mn>1</mn>
<mi>x</mi></mfrac></math>
```

Sagte ich schon, das D. Carlisle einer der MathML Editors ist? (Siehe Seite 7.) Wer will, kann das auch so in **tbook** eingeben und erhält im Browser oder in LaTeX:

$$\widehat{1-x_{\text{eff}}} \neq \left(\int_0^\infty \sin(\tilde{x}) \frac{\sqrt[3]{1/e}}{\beta} dx\right) \neq \lim_{x\to\infty} \frac{1}{x}$$

Mit dem `<m>`-Tag kann man dafür aber auch viel einfacher schreiben:[35]

```
<m>&Hat;{1-x_{\text{eff}}} &ne; {( &int;_0^&infin; sin(&tilde;x)
                                  \frac{\sqrt[3]{1/e}}&beta; dx )}
                              &ne; lim_{x &rarr; &infin;}\frac1x</m>
```

[35] Man kann es nicht oft genug sagen: Das ganze `&...;`-Gedöns kann der Emacs auch als *ein Zeichen* darstellen!

Sieht doch fast wie LaTeX aus, und erspart einem MathML: Gut, dieser Kelch ist also an uns vorübergegangen.

Leider funktionieren bislang in <m> keine Matrizen, aber theoretisch ist das zu schaffen. Mit diesen Elementen kann man bei der Eingabe weitgehend auf MathML verzichten. Alternativ könnte man ein Filter in einer high-level Sprache schreiben und damit MathML gänzlich überflüssig machen.

Wichtig: <m> funktioniert auch innerhalb von MathML, weil **tbook** auf einer leicht modifizierten Version von MathML basiert. Es kann innerhalb <math> überall dort stehen, wo auch ein <mrow> erlaubt wäre, wobei man es manchmal mit <mrow> einklammern muß, weil es in mehrere MathML-Elemente expandiert wird.

Keine Sorge, gültiges MathML Präsentations-Markup wird weiterhin erkannt, und für HTML wird nur gültiges MathML erzeugt. Übrigens gilt dasselbe für das **tbook**-Element <unit>. Man hat so das beste aus beiden Welten: MathML für's Grobe und Ungewöhnliche, <m> für den Rest.

<dm> ("display math") macht dasselbe wie <m>, nur als abgesetzte Formel.

<ch> funktioniert wie <m> und <dm>, allerdings für einfache chemische Formeln. Die „Variablen“ (Elemente) stehen hier aufrecht. Also z. B.

```
<ch>2Mg^{2+} + 2 CO_3^{2-} &rightarrow; 2 MgCO_3</ch>
```

ergibt „$2\,\mathrm{Mg}^{2+} + 2\,\mathrm{C}\,\mathrm{O}_3^{2-} \longrightarrow 2\,\mathrm{Mg}\,\mathrm{C}\,\mathrm{O}_3$“. Ein simples

```
<ch>GaAs</ch>
```

wird zu „$\mathrm{Ga}\,\mathrm{As}$“ (man beachte den kleinen Leerraum). Gibt man

```
<ch>Al_xGa_{1-x}As</ch>
```

ein, so erhält man „$\mathrm{Al}_x\,\mathrm{Ga}_{1-x}\,\mathrm{As}$“, also genau das gewünschte: Im Index wird x als Variable aufgefaßt. Im Prinzip kann man also auch Mathematik in chemischen Formeln benutzen, allerdings sind zwei Dinge zu beachten:

1. Es werden Buchstabensequenzen, die mit einem Großbuchstaben beginnen, auf den Kleinbuchstaben folgen, als chemische Elemente oder deren Platzhalter[36] angesehen und aufrecht gedruckt. Das gilt jedoch nur im `\displaystyle`, also nicht in Indizes, Exponenten, Brüchen oder Wurzeln.

2. `\thinmuskip` ist verkleinert (von 3 auf 2 mu). Dadurch werden einige Abstände etwas kleiner als gewohnt.

9.4 Schwächen von XSLT

> Wo wir gerade bei diesen Tags sind, eine kleine Selbst-Beweihräucherung: Auf diese Elemente für einfache Formeln bin ich etwas stolz; im Prinzip mag es keine Schwierigkeit sein, die Formelsyntax von simplifiziertem LaTeX zu parsen und umzusetzen. Aber für einen Physiker, der alles in C++ macht und außer Pascal und Basic auch nie was anderes gesehen hat, ist XSLT schon irgendwie eine Herausforderung: Variablen, aber man darf sie nie ändern, keine Schleifen, keine vernünftigen String-Funktionen, eigentlich nur Rekursion bis zum Abwinken.

Ich bin im großen und ganzen überzeugt von XSLT. Es ist definitiv der richtige Ansatz, XML weiter zu verarbeiten. Aber gerade die aufwendigeren Routinen für <multipar> oder <m> etc. haben dafür gesorgt, daß ich nicht restlos begeistert bin. Die Sprache ist einfach zu wenig mächtig.

[36] z. B. „A“ für Alkalimetalle

„Sollen auch Nicht-Programmierer benutzen können, außerdem erlaubt das so effizientere Interpreter," heißt es von offizieller Seite auf die Frage, warum XSLT so primitiv ist. Naja. Irgendwie wohl einfach wieder dieser verblendende Purismus: Man will XML-Entwickler zwingen, auf nicht-XML-Syntax in ihren Anwendungen zu verzichten. Ich empfinde das als Bevormundung, zumal nicht-XML-Syntax ja weiterhin möglich ist, nur viel Schweiß erfordert und zu lahmen Trafos führt.

Es ist nicht einzusehen, daß XSLT keine Schleifenstrukturen kennt, und es insbesondere nicht erlaubt ist, Variablen, die man einmal initialisiert hat, wieder zu verändern. Effizientere Verarbeitung hin oder her. [Saxon, 2002], ein sehr beliebter XSLT-Prozessor, hat veränderliche Variablen wenigstens als Erweiterung, aber natürlich schreibt man nur ungern Stylesheets, die bloß noch mit einem bestimmten Programm zusammenarbeiten. Der Autor von Saxon, Michael Kay, ist der Editor der nächsten XSLT-Version. Vielleicht tut sich ja was.

9.5 Aufteilung auf Unterverzeichnisse: XML Catalog

Es wäre sehr schön, wenn man ein XML-Projekt auf Unterverzeichnisse aufteilen könnte. Das scheint aber zur Zeit nur bedingt möglich zu sein. Natürlich kann man allen Verweisen auf Dateien in XSLT-Stylesheets Unterverzeichnisnamen voranstellen, aber viel schöner wäre es, wenn insbesondere Dateien, die nur die Entitäten auflisten (Endung meist `.ent`), in einer Art Pfad gesucht würden.

Mit dem alten SGML ging das auch, und zwar mit sogenannten *Catalogs*. Da gab es dann eine Datei `catalog` oder `CATALOG`, und in der wurden symbolische Namen der Stylesheets oder der Dokumente auf physisch und lokal vorhandene Dateinamen abgebildet [DocBook, 2002]. Diese Katalogdateien wurden den SGML-Applikationen dann explizit bekannt gemacht, oder sie standen an einer hineinkompilierten Stelle im Verzeichnisbaum.

In XML scheint es noch nichts vergleichbares zu geben. Man ist also darauf angewiesen, alles im aktuellen Verzeichnis zu haben, oder in den Stylesheets alle Dateinamen mit einem expliziten und festen Präfix zu versehen.

Sehr wohl gibt es jedoch Pläne, diese untragbare Situation zu verbessern: Mit *XML Catalog* [Cowan, 2000; XMLCatalog, 2001] soll man entsprechendes Mapping auch mit XML-Parsern hinkriegen. Es klingt ein wenig seltsam, aber wesentlich mehr als diese beiden Literaturstellen habe ich dazu nicht gefunden. Insbesondere scheint sich das W3C dafür nicht zu interessieren (ohne eine Alternative anzubieten). Offenbar wird XML in sehr, sehr üppigen Verzeichnissen betrieben.

Ohne jetzt die hervorragende Arbeit der XML-Parser-Autoren schmälern zu wollen – so schwer kann doch die Implementierung nicht sein. (Ich muß zugeben, daß ein paar XML-Parser es mittlerweile doch gebacken bekommen haben; aber wirklich nur wenige.)

Damit in den Verzeichnissen, in denen man sein Texte schreibt, nicht das völlige Chaos ausbricht, kann man alle DTDs und Entitäten-Dateien in eine einzige, große Datei zusammenkopieren. Und/oder in allen beteiligten Dateien explizite Pfade angeben. Ich mache es erstmal so. Bis XML Catalog.

10 Das Problem der Sprachen

Als Sprachen stehen in **tbook** zur Verfügung:

`"en"`	Englisch
`"de"`	Deutsch
`"fr"`	Französisch
`"it"`	Italienisch
`"es"`	Spanisch
`"ca"`	Katalanisch
`"en-US"`	Amerikanisch
`"en-GB"`	Britisch
`"de-DE-1901"`	BR Deutsch, alte Rechtschr.
`"de-AT-1901"`	Österreichisch, alte Rechtschr.
`"de-DE-1996"`	BR Deutsch, neue Rechtschr.
`"de-AT-1996"`	Österreichisch, neue Rechtschr.

Groß-/Kleinschreibung spielt laut RFC 3066 [RFC [3066], 2001] bei den Codes keine Rolle, auch wenn die ISO empfiehlt, Ländercodes großzuschreiben. Es gibt ein Fallback auf die Grundsprachen, wenn ein Dialekt nicht bekannt ist (z. B. `"fr-CA"` → `"fr"`), alle anderen Probleme werden als Fehlermeldung mitgeteilt (und Englisch benutzt).

Die ganzen Jahreszahlentags sind meine Erfindung. Sie widersprechen nicht dem RFC, das ein drittes Feld zu freien Verfügung zuläßt. Offiziell sind die aber (noch) nicht.

> Ich habe diese Tags an der entsprechenden Stelle (nämlich der Mailing-Liste unter `http://www.alvestrand.no/mailman/listinfo/ietf-languages`) vorgeschlagen, sofort haben sich die ganzen Standardisierungs-Fetischisten drauf gestürzt. Das Problem wurde völlig zerredet, nach zwei Tagen waren wir schon bei der Frage, wie man zwischen neuer und alter chinesischer Schreibung unterscheiden sollte.[37] Mittlerweile sind diese Tags jedoch angenommen worden und somit Standard.

Übrigens: Bei z. B. `"en"`, `"de"`, `"de-1996"` oder `"de-AT"` fehlt die Angabe des Landes oder die Angabe der Rechtschreibvariante (oder beides). In diesem Fall versucht das Stylesheet erst einmal, die Unterform zu erraten,[38] bevor es ein Fallback auf Amerikanisch bzw. BR Deutsch und neue Rechtschreibung ausführt.[39]

10.1 Zur Implementation

10.1.1 LaTeX

Für die LaTeX-Ausgabe sind Sprachen kein Problem, Babel ist ja da. Ich muß nur alle Sprachen, die im XML-Dokument vorkommen, „einsammeln“ und in einer Liste dem Babel-Paket übergeben. Eventuell kriegen auch noch andere Pakete die Sprachliste, eventuell abgespeckt (d. h. ohne Dialekte), eventuell auch nur die Hauptsprache des Wurzelelements.

Ansonsten muß der Umwandler eben die Sprachen mit `\selectlanguage` einschalten. Ich habe dabei darauf geachtet, daß so wenige dieser Befehle wie möglich eingefügt werden, denn häufig sind sie redundant, wenn ein Eltern-Element schon dieselbe Sprache hat.

Außerdem muß man natürlich darauf achtgeben, daß man die Sprachen lokal zum jeweiligen Element hält.

---

[37] Wer Aachener ist, soll sich bei mir melden, vielleicht kriegen wir ja `"i-oecher"` bei der IANA durch.

[38] Es sucht dabei nach Vorkommen von *daß–dass*, *heuer*/*Jänner* bzw. *behavio(u)r*/*neighbo(u)r*. Eine möglicherweise zu simple Methode, aber recht wirkungsvoll und vor allem recht schnell.

[39] Ich als Anhänger der traditionellen Schreibweise hatte erst ein Fallback auf alte Orthographie verdrahtet. In der Mailingliste zum RFC 3066 hat man mich dann aber davon überzeugt, daß man bei `"de"` wohl meist die aktuellste Fassung erwartet. Und ich kann, obwohl ich kein Monarchist bin, mit `"de-1901"` gut leben.

10.1.2 Andere Ausgaben

Für HTML ist die Sprache an sich überhaupt kein Problem, das `xml:lang`-Attribut wird einfach durchgeschliffen (und in das `lang`-Attribut kopiert, zur besseren Kompatibilität). Momentan bedeutet es aber noch rein gar nichts, da sich kein HTML-Betrachter, den ich kenne, mit diesem Attribut auseinandersetzt.

Allerdings muß ich an vielen Stellen Babel nachahmen. Beispielsweise soll das Inhaltsverzeichnis ja auch „Inhaltsverzeichnis“ und nicht „Table of Contents“ heißen. Dafür brauche ich String-Tabellen, in denen Tags auf Textfetzen gemappt werden. Aus einem Tag wie `"Table of Contents"` wird dann „Inhaltsverzeichnis“, oder natürlich was anderes, wenn eine andere Sprache aktiviert ist.

Einige dieser Textfetzen sind in Babel nicht enthalten und werden daher auch in der LaTeX-Ausgabe benutzt.

10.1.3 Sprachabhängiger LaTeX-Code

Es gibt auch sprachabhängige *Code*fetzen, d. h. sprachabhängige Befehle oder Argumente. Der Bibliographie-Stil beispielsweise ist im Englischen vielleicht `plain`, im Deutschen jedoch `dinat`. Hier kommen ebenfalls die String-Tabellen zum Einsatz.

10.2 Dezimaltrennzeichen

Alte Leier: Engländer und Amerikaner schreiben eine Dezimalzahl wie 8.8, Deutsche und wohl sehr viele andere halten sich an den ISO-Standard, der 8,8 vorschreibt. Ich habe versucht, wo es nur geht, darauf zu achten. Das betrifft zwei Bereiche: Zum einen habe ich in Stylesheets überall dort, wo geprüft werden sollte, ob es ich um eine Zahl handelt (z. B. MathML oder `<psfrag>`), dafür gesorgt, daß auch mit Komma die Zahl als Zahl erkannt wird.

Zum anderen werden bei der LaTeX-Ausgabe Zahlen stets in `$ $` eingefaßt, um zu verhindern, daß Minuskelziffern benutzt werden (auch wenn die eher selten überhaupt vorhanden sind). Außerdem kommen Zahlen natürlich auch in normaler Mathematik vor. Normalerweise führt das zu falschem Abstand um das Komma herum, wie $8, 8$ demonstriert. In all diesen Fällen wird daher entweder das Komma eingeklammert `{,}` oder ein lokal gehaltenes

```
\mathcode'\,="013B
```

vorangestellt, was Abhilfe schafft.

10.3 Das Stichwortverzeichnis

Theoretisch müßte man das Stichwortverzeichnis nach sprachspezifischen Regeln erzeugen. Ich habe vorerst darauf verzichtet, da die aktuelle Implementation auch diakritisch geschmückte Zeichen sinnvoll einsortiert. Und wenn die Sortierreihenfolge selbst in Deutschland nicht ganz einheitlich ist (DIN 5007 vs. Duden), kann man davon ausgehen, daß Sortierregeln wohl überall eher eine Geschmackssache sind. Ich persönlich ziehe den Duden-Weg vor. Das war damals auch einer der Gründe, xindy statt MakeIndex zu benutzen.

Eine sprachspezifische Sortierung ist aber hinzukriegen, und zwar mit denselben Mitteln, die auch `dinat` anstelle von `plain` einsetzen. (Siehe oben.) Es müssen halt verschiedene Index-Stildateien aktiviert werden.

11 Das Modul für Adressen

Ich habe drei Module, die zum Gesamtsystem dazugebunden werden: MathML, BIBTEXML und die Adreß-Datenbank. Auf MathML werde ich in Abschnitt 12 auf der nächsten Seite eingehen und für BIBTEXML muß ich auf die offizielle Seite unter BibTEXML [2001] verweisen.

Für Adressen, wie sie in Briefen naturgemäß auftauchen, wollte ich eine eigene Adreß-Datei haben, also ein Art Adreß-Buch. Das Konzept ist zu dem, das die Koma-Klassen bieten, sehr ähnlich. Es sollte jedoch eine XML-Datei sein, sonst kriegt man keine HTML-Fassung hin. (Okay, Briefe haben im Web wohl wenig verloren . . .) Am besten fängt man damit an, eine DTD für deren Syntax zu verfassen.

11.1 Bestehende Adreß-DTDs

Doch halt! Vielleicht hat das ja schon einmal einer gemacht, ja vielleicht hat sich ja schon eine Art Quasi-Standard durchgesetzt, wie MathML im Bereich der Mathematik.

Klein Torsten, gar nicht dumm, fragt mal in der Newsgroup rum. Und tatsächlich: OASIS, der Spezifikator von DocBook, hat sowas erstellt, zu bewundern unter CIQ [2002]. Diese DTDs (es sind insgesamt drei) machen die Kindheitsprobleme von XML deutlich: Völlig übertrieben und an der Realität vorbeigestaltet.

Die Spezifikation deckt wirklich nur Namen von Personen, Behörden oder Firmen und deren Adressen ab. Und sie ist hundert DIN-A4-Seiten lang! Ich glaube, sie berücksichtigt sogar Einfuhrzölle . . . Den Vogel abgeschossen haben sie allerdings mit der Tatsache, daß es keine Möglichkeit für Email-Adressen gibt. Dafür muß man eine weitere DTD einbinden, die noch Dinge wie Hobbys und Einkommen enthält. Da waren wirklich Könner am Werk.

11.2 Fazit: eine eigene DTD

(Hatte ich die Überschrift nicht schon einmal irgendwo?) *Meine* Adreß-DTD ist dann eine ganze Bildschirmseite lang geworden. Ich habe alle Felder in Attributen organisiert, was man eventuell später noch mal ändern muß, wenn es sich als ungünstig erweist. Die Frage ist eben: Brauche ich explizite Formatierung (kursiv, Sprache, . . .) in Adreß-Feldern? Ich habe diese Frage zunächst einmal für alle Felder mit Nein beantwortet.

Die Syntax einer Adressen-XML-Datei ist nahezu trivial: Alles wird vom Wurzel-Element `<addresslist>` eingeklammert, dazwischen stehen beliebig viele `<addressentry>`-Elemente. Die Adresse wird dann innerhalb dieses Elements auf die Attribute verteilt.

Hier nun ein Beispiel für eine solche Adreßbuch-Datei, mit einem einzigen Eintrag:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE addresslist PUBLIC "-//Torsten Bronger//DTD address book 1.0//EN"
                             "tbadrdb.dtd">
<addresslist owner="Bugs Bunny &lt;bugs.bunny@warner.com&gt;">
  <addressentry nickname="Duffy" sex="male" firstnames="Duffy"
    recipient="Duck" street="ACME Street 125" city="Toontown"
    postalcode="0001" country="Toonland" countrycode="us"
    phone="001-555-5555" email="duffy.duck@warner.com"
    private="yes"/>
</addresslist>
```

Die Empfänger meiner Briefe werden über `nicknames` angesprochen, also letztlich IDs. Es gibt ein Feld `keywords`, in das man z. B. Dinge wie „Kegelclub" reinschreiben kann, um dann später einen Serienbrief an den ganzen Kegelclub zu schreiben. Das Feld `private` gibt an, ob es

ein Privatmensch oder Geschäftskunde/Firma/Behörde ist. Danach wird dann später der Default-Wert des Attributes `formal` vom `<letter>`-Element bestimmt. Dem `private`-Attribut kann man auch Adreßbuch-übergreifend einen Vorgabewert geben, indem man es in `<addresslist>`, also dem Wurzel-Element, setzt. Die restlichen Felder sind wohl selbsterklärend.

Das Feld `addressdetails` kann so sämtlichen sonstigen Sermon enthalten, der nicht in `recipient` paßt, z. B. „zu Händen von" oder „Dezernat Schießmichtot". Jedes „`*`" in diesem Feld beginnt eine neue Zeile.

Im `owner`-Attribut des `<addresslist>`-Elements steht das eigene Namenskürzel. Es wird später mit dem `from`-Attribut des `<letter>`-Elements abgeglichen.

Wenn sich doch noch die Leute auf eine vernünftige Standard-DTD für Adressen einigen sollten, sollte man sich daran anschließen. Es wäre beispielsweise verführerisch, irgendwann sein Adreßbuch aus Outlook oder dem Navigator exportieren und direkt für XML-Briefe verwenden zu können. Oder umgekehrt. Dafür gibt es zwei Möglichkeiten:

- Man schreibt zwei XSLT-Trafos zwischen der eigenen Adreß-DTD und dieser anderen DTD (also für beide Richtungen).
- Man schreibt eine XSLT-Trafo hin zur anderen DTD, konvertiert damit einmalig alle seine Adreß-Listen und stellt das komplette Brief-System auf das neue Format um.

Beide Wege sollten an einem Tag zu verdrahten sein.

Noch ein Nachschlag: Die KOMA-Klassen von LaTeX verwenden auch eine Art Adreßbuch für den Briefstil. Es ist eine nahezu triviale, wenn auch erstmal wenig nützliche Aufgabe, eine XSLT Trafo dahin zu schreiben. Wenig nützlich deshalb, weil selbst bei Benutzung der KOMA-Klassen das Haupt-Stylesheet die Trafo auch selbst vornehmen kann. Der umgekehrte Weg wäre interessanter, setzt aber eine Programmierung in einer höheren Programmiersprache/Skriptsprache voraus.

12 MathML

MathML ist eine XML-Anwendung für das Beschreiben mathematischer Ausdrücke. Die normativen Richtlinien habe ich aus MathML-W3C [2001]. MathML ist so konzipiert, daß es recht einfach in bestehende XML-Anwendungen quasi als Modul eingebettet werden kann.

Für eine Einführung in MathML sei auf die Spezifikation und [Goossens und Rahtz, 1999, Kap. 8] verwiesen.

Das Problem an MathML ist einerseits seine Umständlichkeit, aber auch die Tatsache, daß die offizielle W3C-Empfehlung Raum für Interpretation läßt. Deshalb muß man sich bei der Umsetzung quasi noch selber ein paar private Zusatzregeln ausdenken. Dabei sollten drei Dinge beachtet werden:

1. Die Fähigkeiten von LaTeX sollten gut ausgelastet werden,
2. das MathML, das man schreibt, müssen auch andere Interpreter (v. a. Browser) richtig verstehen und
3. MathML, das eventuell aus anderen Quellen stammt, sollte auch funktionieren.

Dabei sind die Punkte 1 und 2 ungefähr hundertmal wichtiger als Punkt 3.

[Bei Punkt 3 geht es mir letztlich auch nur darum, daß man nicht zu minimalistisch arbeitet. Man kann MathML eh nicht komplett implementieren, aber man sollte es sich auch nicht *zu* einfach machen; einerseits zwingt man dann dazu, neben MathML auch noch das eigene MathML „light" zu lernen, und außerdem würde MathML kaputt gehen, wenn jeder Implementator so arbeiten würde.]

12.1 Meine Implementierung

Die erste Vereinfachung, die ich vorgenommen habe, ist, daß ich mich auf das *Presentation Markup* von MathML beschränkt habe. Es gibt auch noch das *Contents Markup*, das typographisch schlechter ist und eigentlich für den Austausch zwischen Mathematik-Programmen gedacht ist. Es gibt zwar auch Ansätze, das Contents-Markup in LaTeX umzuwandeln [DB2LaTeX, 2002], aber werden das auch die Browser verstehen? Außerdem gibt es da eventuell bald andere Möglichkeiten, siehe Abschnitt 13.1 auf Seite 77.

Ich kenne zwei Versuche, MathML mittels XSLT nach LaTeX umzuwandeln, nämlich das schon genannte DB2LaTeX [2002] und Gurari [2000], letzteres versteht aber Gurari selbst mehr als „Übung“. Allerdings bietet auch das erstere kaum ausreichende Unterstützung. Was beispielsweise keiner berücksichtigt, ist, daß man manchmal Formeln eine Nummer geben möchte.

Ich habe das Presentation Markup recht gut abgedeckt, auch die Attribute werden, soweit ich es für sinnvoll hielt, unterstützt. Im Zweifel kann man also gemäß Spezifikation drauflos schreiben, mit ein wenig Glück funktioniert das dann. An einigen Stellen habe ich die MathML-Spezifikation auf meine Weise interpretiert (d. h. beschränkt). Darauf gehe ich jetzt ein.

12.1.1 `underbrace`

Um wie in LaTeX sowas wie

$$\underbrace{x+\ldots+x}_{n\ \mathrm{mal}}$$

(also eine `\underbrace`) hinzubekommen, muß man bei mir in MathML folgendes schreiben:

```
<math>
  <munder>
    <mrow>
      <mi>x</mi> <mo>+</mo>
      <mo>&hellip;</mo>
      <mo>+</mo> <mi>x</mi>
    </mrow>
    <munder>
      <mo>&UnderBrace;</mo>
      <mrow>
        <mi>n</mi> <mtext> mal</mtext>
      </mrow>
    </munder>
  </munder>
</math>
```

Analoges gilt für `\overbrace`-Konstrukte. Übrigens kann man mit einem `accent="false"` am zweiten `<munder>`-Element (wie gefordert) verhindern, daß der Text unter der Klammer verkleinert wird.

12.1.2 `<mtext>`

`<mtext>` ist dafür gedacht, Text in eine Formel zu bringen, so wie `\mathrm{}` in LaTeX. Ich habe das so implementiert, daß dieser Formeltext zu seinen „Nachbarn“ ohne Zwischenraum steht. Den Zwischenraum muß man also in den `<mtext>` schreiben:

```
... <mi>n</mi> <mtext> mal</mtext>
```

Ein einfacher Leerschritt anstelle des ` ` reicht nicht! Das ist auch so von der Spezifikation gedacht. An allen Stellen in MathML wird führender und folgender Leerraum sowie Leerraum um Tags herum ignoriert.

12.1.3 Gleichungs-Array

In LaTeX kennt man `eqnarrays`. In MathML auch, allerdings geht das dort von hinten durch die Brust ins Auge, über eine Tabelle mit nur einer Zeile:

```
<math>
  <mtable groupalign="right center left">
    <mtr>
      <mtd>
        <mrow>
          <maligngroup/>
          <mn>1</mn><mo>+</mo><mn>1</mn>
          <maligngroup/>
          <mo>=</mo>
          <maligngroup/>
          <mn>2</mn>
        </mrow>
      </mtd>
      <mtd>
        <mrow>
          <maligngroup/>
          <mn>4</mn>
          <maligngroup/>
          <mo>=</mo>
          <maligngroup/>
          <mn>2</mn><mo>&CenterDot;</mo><mn>2</mn>
        </mrow>
      </mtd>
    </mtr>
  </mtable>
</math>
```

Dieser ganze Sermon wird dann zu einem deprimierend simplen

$$\begin{aligned} 1+1 &= 2 \\ 4 &= 2\cdot 2 \end{aligned}$$

Man *muß* dafür aber `groupalign` auf `"right center left"` setzen, sonst wird zwar richtig untereinander geschrieben, aber nicht mehr ausgerichtet. (Warum untereinander stehende Gleichungen in MathML als Tabellen-Spalten *nebeneinander* beschrieben werden, habe ich allerdings nicht verstanden.)

12.1.4 tbook und MathML mischen

Obwohl **tbook** gültiges MathML versteht und in der HTML-Ausgabe auch garantiert nur gültiges MathML erzeugt, nutzt es intern eine leicht aufgeweichte Version von MathML. Diese erlaubt es, die Elemente `<m>`, `<unit>` und `<ch>` auch *innerhalb* von MathML zu benutzen, was die Sache ganz wesentlich vereinfacht. Dieses furchtbare Gleichungs-Array aus dem vorangehenden Abschnitt schmilzt so zusammen auf

```
<math>
  <mtable groupalign="right center left">
    <mtr>
      <mtd>
        <m>1+1 # = # 2</m>
```

```
      </mtd>
      <mtd>
        <m>4 # = # 2 &CenterDot; 2</m>
      </mtd>
    </mtr>
  </mtable>
</math>
```

Das ist zwar immer noch aufwendiger als die LaTeX-Version, aber schon ein guter Schritt nach vorne. Ich benutze „#” statt „&“ als Ausrichtungs-Marker, weil man ja sonst in einer XML-Datei „&“ schreiben müßte, und Doppelkreuze fast ebenso selten in Formeln auftauchen dürften wie der Ampersand.

> *Technischer Hinweis:* Benutzt man einfache Formelsyntax innerhalb von MathML, werden diese einfachen Formeln für die HTML-Ausgabe in eine Sequenz von MathML-Elementen expandiert. Sie werden nicht von einem impliziten `<mrow>` eingeklammert! Es gibt nur eine Ausnahme, und die sieht man in obigem Beispiel: Ist ein `<m>`-Element einziges Kind von einem `<mtd>`, so wird ein implizites `<mrow>` zwischen beiden generiert.
>
> Der Grund dafür ist, daß MathML's Pendant zu den Doppelkreuzen das `<maligngroup/>` ist, und das darf nur innerhalb von `<mrow>` stehen (siehe Beispiel weiter oben), so unverständlich das auch für mich ist. Ich glaube, dieses implizite Verhalten hilft und dürfte nie schaden.
>
> Das *erste* `<maligngroup/>` wird von **tbook** automatisch eingefügt. Man muß die „#“ also nur dort einsetzen, wo in LaTeX ein „&“ stehen würde.

12.1.5 Gleichungen durchnumerieren/Labels setzen

MathML benutzt ziemlich verschlungene Wege, um Gleichungen mit Nummern auszustatten. In meiner MathML-Untermenge implementiere ich drei davon.

1. Ein `id`-Attribut beim `math`-Element:

```
<math id="Einstein">
  ...
```

2. Ein `id`-Attribut bei einem `<mtd>`-Element in einem Gleichungs-Array:

```
<math>
  <mtable groupalign="right center left">
    <mtr>
      <mtd id="Zweistein">
        ...
```

3. Ein `<mlabeledtr>`-Element:

```
<math>
  <mtable groupalign="right center left">
    <mlabeledtr id="Dreistein">
      <mtd>
        <mtext> (2.1) </mtext>
      </mtd>
      <mtd>
        ...
```

Hierauf kann man dann über `<ref refid="Dreistein">` *oder* `<mathref refid="(2.1)">` verweisen. Die (2.1) wird dabei aus dem Text entfernt und durch die richtige laufende Nummer ersetzt.

(Im nachhinein glaube ich, daß es ein Fehler war, diesen dritten Weg zu unterstützen. Er ist wohl vor allem für programm-generierten MathML-Code gedacht. Was soll's.)

(Korrekt plazierte) Labels führen also zwangsläufig zu automatisch numerierten Gleichungen, und ich kann i. a. über `<ref refid="`*`label-id`*`">` darauf verweisen.

12.1.6 Wie gut werden Gleichungen?

Nun, für LaTeX glaube ich, daß die Gleichungen ähnlich schön gesetzt werden wie handgemacht. Man hat halt nur nicht mehr alle Freiheiten (was glücklicherweise aber auch viel Gefrickel vermeidet).

Für die Browser, die meinen MathML-Code ja quasi verbatim so zugeschoben bekommen, hängt es von der Qualität des Browsers ab, was er mit gültigem MathML-Code anfangen kann. Das ist zur Zeit (Juni 2002) noch etwas dürftig.

Mozilla und Netscape 7 sind die einzigen nennenswerten Browser, die's überhaupt können. Nur: Der Zeilenumbruch in Formeln ist unsinnig (man kann aber, ähnlich wie in LaTeX auch, mit { } bzw. `<mrow>` einen Zeilenumbruch verhindern), die Standard-Unterstützung ist befriedigend und das Erscheinungsbild ausreichend. Alle Steuerzeichen (Skips, Umbruch-Empfehlungen etc.) werden als *Text* ausgegeben. Und weitere Bugs und Ungenauigkeiten. Aber das alles wird rasch besser. Dann wird man sehen.

Microsoft ist mitverantwortlich für MathML 2.0, und ich glaube nicht, daß sich der Internet Explorer dem verschließen wird.

12.2 Einbettung in die DTD

Wie kriege ich MathML in meine XML-Anwendung? Es geht hier also zunächst nur um die *Syntax*, also die Frage der Validierung. Die *Transformation* hin zu LaTeX ist noch eine ganz andere Frage.

Es wäre natürlich albern, MathML oder auch nur eine Untermenge quasi abzutippen und so in die eigene DTD einzufügen. Letztlich geht es jedoch ganz einfach. Die Zeilen

```
<!ENTITY % MathML.dtd PUBLIC "-//W3C//DTD MathML 2.0//EN"
                             "mathml2.dtd">
%MathML.dtd;
```

irgendwo in der DTD fügen MathML mit allen Elementen und Entitäten zur eigenen XML-Anwendung hinzu. Wenn man nichts weiter angibt, kann man dann in den entsprechenden XML-Dokumenten die MathML-Elemente ohne Namensraum-Präfix benutzen, man kann also `<math><mi>x</mi></math>` schreiben anstatt `<m:math><m:mi>...`

Ganz entsprechend bekommt man andere Modul-DTDs in die eigene Haupt-DTD.

12.3 Schwächen von MathML

Was ist von MathML zu halten? Diese Frage hat einen ernsten Hintergrund. Ich mußte mich bei meinen Bemühungen sehr intensiv mit MathML befassen, und habe dabei so manches Mal vor dem Bildschirm meinem Unmut luftgemacht.

Fest steht, daß mit MathMLs 'Presentation Markup' hochqualitative Formelbeschreibung möglich ist. Praktisch alles, was mit LaTeX zu erzeugen ist, läßt sich in MathML nachbilden. Leider erlaubt MathML auch Absurditäten, die sich kaum in LaTeX umsetzen lassen. Ärgerlicher sind jedoch einige generelle Schwächen. Die größten Probleme von MathML 2.0 sind:

1. *Viel* zu kompliziertes Markup. Für alle nicht-trivialen Gleichungen ist MathML derart umständlich, daß nur noch Automaten MathML ökonomisch erzeugen können. Ich muß natürlich einräumen, daß ich von LaTeX verwöhnt bin, denn dieses ist wahrscheinlich die einfachste Möglichkeit überhaupt, Formeln einzugeben; dennoch: Wenn sich XML brüstet, „menschlich" zu sein, und ein Teil von XML, nämlich MathML, derart aus der Reihe tanzt, ist das sehr schade.

2. Die Namen für die Entitäten sind hoffnungslos chaotisch. Ich mußte einen Großteil der MathML-Entitäten abtippen und habe dabei immer wieder versucht, ein System zu entdecken; das gibt es jedoch nur in Ansätzen. Es ist schon nicht einzusehen, daß insbesondere ganz exotische Glyphen gleich sechs oder sieben Synonyme haben,[40] aber selbst die Regel, daß wenigstens der LaTeX-Name darunter ist, hat Ausnahmen.

   In den zum Standard gehörenden Entitätsdateien reichen sich die Fehler die Hand. Zudem beißen sie sich mit den Standard-HTML-Entitäten, obwohl doch das eine als Modul für das andere gemacht wurde. Mein CWEB-Programm `tbcrent` zeigt daher einige Warnungen an, für die ich absolut nichts kann. Ich könnte zwar die Fehler bereinigen, sehe aber nicht ein, das dann auch noch zu warten. Durch geschickte Wahl der Reihenfolge, in der die fehlerhaften Dateien in `hmml2dst.dtd` und in `tbcrent` eingelesen werden, kann man jedoch für die Anwendung sichtbare Probleme ausschließen.

   In diesem Zusammenhang: Ich verstehe die Auswahl der Glyphen nicht. Einige der Symbole haben in der Mathematik nun gar nichts verloren, und wenn Benutzer sie doch verwenden, sollte man ihnen das abgewöhnen (z. B. bei Dingbats). Andere Symbole, die völlig in die Sequenz gepaßt hätten, wurden ausgelassen.

   Beispiele: Die Unicode-Pfeile wurden fast alle aufgenommen, nicht jedoch der Blitz, der hierzulande so gerne für den Widerspruch benutzt wird. Es gibt zwar `epsi` und `straightepsilon`, aber `epsilon` nicht. Ferner definiert MathML `varepsilon` mit „lateinisches (!) kleines offenes e" (Nr. 0x25B); damit ist es Teil der IPA-Lautschrift-Zeichen. Bitte … ?!

   Überdies ist mir unverständlich, daß sich Unicode- und MathML-Konsortium nicht soweit einigen konnten, daß man mit den – ja nicht gerade wenigen – 65536 Zeichen vom Unicode auskommt. Etliche Zeichen liegen jenseits der Zwei-Byte-Grenze, was einigen XML-Tools, wie z. B. nsgmls, Schwierigkeiten macht.

3. Ungenaue MathML-Spezifikation. Ein paar (unwichtige) Symbole konnte ich bis heute nicht vernünftig identifizieren. Die Beschreibungen sind dürftig, die Bitmaps zu klein, alles in allem sind die Glyphen-Tabellen schlampig (d. h. fehlerhaft) und uneindeutig. Wie z. B. sieht ein $\cup$ mit Serifen aus?! (Ʊ :-)

   Unglaublich: Der Unicode sagt, daß Zeichen #2032 zu einem $'$ werden soll, wie in f'. Einer der MathML-Editors meinte dann beiläufig in einer Newsgroup, nö, in MathML ist's ein ′, d. h. man muß es erst noch hochstellen. Wohlgemerkt, in den eigenen MathML-Tabellen ist es so wie im Unicode aufgeführt – noch heute. Und sowas bei einem alles andere als seltenen Zeichen! (Die *zweite* Ableitung #2033 ist dann übrigens *doch* bereits hochgestellt. Ruhig, Torsten, ganz ruhig.)

   Wichtige typische Mathematik-Konstrukte, wie z. B. Gleichungs-Arrays oder Matrizen, werden nur durch *Beispiele* der Spezifikation abgedeckt, die aber zu spärlich sind, als daß

[40] Kostpröbchen? Das Zeichen „$\nleqq$" ist auf folgende Weisen in MathML darstellbar: `≦̸` oder `≧̸` oder `≦̸` oder `⩽̸` oder `⩽̸` oder `⩽̸` oder `≤⃒`. Alle sieben Wege würden zum Ziel führen. Wer sich jedoch wie ich *keinen* dieser Namen merken kann, darf auch „>" benutzen.

sie die Spezifikation tragen könnten. Es gibt häufig viele Wege zum Ziel, und die Beispiele schlagen den Standard-Weg vor. Verbindlich ist das aber natürlich nicht.

4. Die Semantik der MathML-Elemente ist stark kontextsensitiv. Daher muß ein Renderer aufwendige Fallunterscheidungen für ein und dasselbe Konstrukt, z. B. einen Operator, machen. Salopp gesagt ist die zu geringe Zahl von Tags von der notwendigen Funktionalität überfordert.

   Beispiel: `<mover>` wird sowohl dafür verwandt, die Obergrenze über ein `\sum` zu schreiben, als auch dafür, einen Dach-Akzent $\hat{x}$ zu erzeugen! Auch der Fall einer dehnbaren `\overbrace` wird darüber abgehandelt.

   `<mtable>` ist eigentlich für Matrizen gedacht, ist aber unverändert ebenso für (u. U. numerierte) Gleichungs-Arrays zuständig. Plötzlich haben Tabellenzellen `<mtd>` eine völlig andere Bedeutung, u. a. stehen sie untereinander! Ohne gefährliche Heuristik kommt der Renderer da dann nicht mehr weiter.

   Größter Unsinn: `<mi>x</mi>` ist die *Variable x*, `<mi>sin</mi>` ist die *Standard-Funktion* sin, und `<mi>Text</mi>` ist *Formeltext* (in LaTeX `\textrm{Text}`). Ob die „Mehrbuchstabichkeit“ die Unterscheidung macht, steht nirgends. (Selbst dann wäre es Unsinn, zumal mit `<mtext>` eine saubere Alternative angeboten wird.)

Fazit: Der MathML-Standard ist zu schwammig und schlampig. Vieles an wichtiger Mathematik ist auf mehrere Weisen zu lösen, die Renderer werden vermutlich nur eine davon vernünftig darstellen. Die wahre Semantik werden also Netscape und Microsoft bestimmen. MathML hat Glück, daß die Anfangsgemeinde der Implementierer klein und gut verkabelt ist, ansonsten hätte es eine Fehlgeburt werden können.

In diesem Zusammenhang: Über die XSLT-, XSL-, XPath- oder HTML-4.01-Spezifikationen kann ich mich nicht beklagen; die sind sauber, eindeutig und sogar als Tutorium zu gebrauchen.

13 Ausblick

13.1 Für meine Anwendung

Uff, die war viel Aufwand. Aber andererseits habe ich nur in einem XML-System nachgebildet, was ich von LaTeX her kenne und benötigt habe. Keinen Deut mehr. Ein Trost ist, daß die Grundannahme, nämlich daß es überhaupt geht, richtig war. Aber zurück zum Ausblick.

- Wenn XSLT irgendwann einmal in der nächsten Version vorliegt [XSLT2-W3C, 2002], wird folgendes möglich sein:
  - Vieles, was momentan noch über Erweiterungsfunktionen realisiert ist, wäre dann gemäß dem Standard kodiert. (Z. B. Ausgabe in mehrere Dateien.)
  - In `<multipar>` werden Absätze nicht mehr durch „`*`“, sondern, wie in LaTeX, durch Leerzeilen voneinander getrennt. Ähnliches für einige andere Elemente. (Suche nach regulären Ausdrucken.)
  - Viele String-verarbeitende Routinen werden einfacher und/oder besser. (Neues XPath, siehe XPath2-W3C [2002].)
  - Eventuell kann man dann auch das Contents Markup in MathML abdecken, indem man bestehende XSLTs benutzt, um Contents in Presentation Markup umzuwandeln [MathML-c2p, 2001], das dann wieder in ein Node-Set umwandelt wird, um das dann wiederum durch meine Presentation-Markup-Templates zu jagen. (Typenumwandlung in ein Node-Set.)

  - Aufteilung des HTML-Dokuments auf Tochterdokumente mit jeweils einem Kapitel. (Ausgabe in mehrere Dateien.)
  - Viele Vereinfachungen und Beschleunigungen durch die neuen Strukturen und Funktionen.

- Eine Unterstützung für (Folien-)Vorträge.
- Mehr AMSLaTeX-Funktionen ausnutzen.
- Doch senkrechte Linien in Tabellen?
- Den Effekt von `dcolumn.sty` nachahmen.
- Den Unicode noch besser unterstützen.
- Alle Grafiken und damit assoziierte Dateien in einem eigenen Unterverzeichnis halten.

13.2 Zukunft von XML – meine Sicht

Die sieht sicherlich rosig aus, was die reine Verbreitung betrifft. Es wird in Browsern, Datenbanken und verschiedenen Dokumentverarbeitern eine große, teilweise die beherrschende Rolle spielen.

Was jedoch den Nutzen für den Verbraucher angeht, wird die Rolle klein sein. Insbesondere wird jeder Hersteller seine eigenen, netten DTDs definieren, und damit wird XML als genereller Dateiformat-Lieferant herhalten müssen. Wenn also ein Produkt mit seinem „XML-Export" beworben wird, bedeutet das meist: „Ja, damit können Sie Dateien schreiben, die Sie mit einem XML-Parser parsen könnten; verstehen können die allerdings nur wir." DocBook-Export, XHTML 1.1-Strict-Export, da *würde* es interessant.[41]

Man muß immer im Kopf behalten: XML vereinheitlicht den Low-Level-Parser, nicht mehr. Die übergeordnete Syntax (d. h. die der Elemente untereinander), muß immer noch individuell verstanden und umgesetzt werden. Das „Umsetzen" ist natürlich der bei weitem schwierigste Teil, und „kompatibel" sind zwei XML-Dateien i. a. in keiner Weise. Als reiner Dateiformat-Stifter bringt XML also so gut wie nichts.

Datenbanken Ich habe mich zwar mit Datenbanken, aber nie im Zusammenhang mit XML beschäftigt, daher nur ein kleines Beispiel:

Bei Datenbank-Abfragen wird ermöglicht, die Ergebnis-XML-Datei mit XSLTs, die u. U. in wenigen Stunden erstellt sind, in eine andere XML-Datei umzuwandeln, mit der dann ein anderes Programm zurechtkommt oder die als XHTML im Web dargestellt werden kann. Meist im Stapel-Betrieb. Die Datenbank selber liegt meist nicht in XML vor, ansonsten müßte die Firma anbauen. Aber selbst eine reine XML-Datenbank kann sinnvoll sein, siehe [BibTeXML, 2001].

Das ist aber nur als Hinweis zu verstehen, wo man XML noch benutzt; ich hatte mit sowas nie zu tun.

Web Im Web wird XML für mehr Homogenität in HTML und seinen assoziierten Standards sorgen (MathML, SVG, . . . , obwohl die CSS wohl nie durch etwas XMLiges ersetzt werden werden), das wird das Web-Chaos aber nur für die Hersteller und auch nur um einen infitesimalen Betrag verkleinern. Es *wird* sich an der Front vieles verbessern, auch für den Verbraucher, aber das hat

[41] Wobei HTML-Export auch nicht immer semantisch sein muß. Beispielsweise mißbraucht die Google-Suchmaschine `http://www.google.com` bei ihrer Konvertierung PDF → HTML den HTML-Standard als reine Seitenbeschreibungssprache. Entsprechend unbrauchbar ist oft das Ergebnis.

mit den Ideen von XML nichts zu tun, sondern einfach damit, daß die Zeit für die ärgerlichen Gummi-Standards abgelaufen ist.

Wichtig ist hingegen die Möglichkeit, ähnlich wie bei den Datenbanken, Web-Inhalte in einem Zwischen-XML-Format zu parken und für die Publikation (u. U. stehenden Fußes) in HTML umzuwandeln. So kann man Web-Präsentationen in Design und Funktionalität beliebig verändern und läßt den Inhalt dabei unangetastet.[42]

Dokument-Erzeugung XML erlaubt elegante Autorensysteme mit einer hohen Qualität und nahezu unbegrenzten Verwendungsmöglichkeiten für die Dokumente. Da jedoch die meisten abwinken, wenn sie kein Icon für „fett" in der Symbolleiste sehen, dürften die Früchte, ähnlich wie bei LaTeX, wenigen Auserwählten vorbehalten bleiben. Und das, obwohl XML-Werkzeuge dieselbe Funktionalität wie etwa Word werden bieten können, inklusive Wysiwym[43], das dem Nutzen und Komfort nach gleichzustellen ist gegenüber Wysiwyg. Eine kritische Bewertung von Wysiwyg findet man bei [Cottrell, 2001].

Ob es immer Leute geben wird, die LaTeX oder XML direkt im Texteditor eingeben, hängt davon ab, wie gut diese XML-Autorensysteme werden, oder besser gesagt, wie gut die DTDs sind, auf denen sie basieren werden. Eine gute DTD könnte sich, wie LaTeX seinerzeit, zum Quasi-Standard mausern und wird dann auch mit schönen Wysiwym-Eingabesystemen ausgestattet werden.

Man sollte sich dem dann nicht verschließen, letztlich zählt das Ergebnis. Die typischen Argumente eines LaTeXers gegen Word greifen hier jedenfalls nicht mehr. Wenn *ich* irgendwann einmal für immer auf ein grafisches Frontend umsteigen werde, werde ich sicher ein sentimentales Kribbeln im Bauch spüren. Aber ich habe ja auch Fontinst und musicTeX hinter mir.

Es ist vielleicht vergleichbar mit der Frühzeit des Automobils: Damals gab es furchtbar viele Tüftler, und es gab furchtbar viel zu frickeln und auszuprobieren. Entsprechend groß war natürlich auch die Flexibilität, die man beim Zurechtschneidern „seines" Autos hatte.

Heute gibt es viel, viel weniger Tüftler. Kaum jemand vermißt die Flexibilität. Die meisten brauchen Lenkrad, Pedale und irgendwas zum anschalten. So ein Minimal-Interface ist allemal effizienter, und auch komfortabler. (Der *Spaß* steht natürlich auf einem anderen Blatt . . .)[44]

13.3 LyX

Ich glaube, Systeme wie LyX sind der Weg, der mittelfristig gegangen werden muß. WYSIWYG ist aus Prinzip undurchführbar, wenn man für mehr als eine Verwendung schreibt, und die Menschen werden einsehen, daß sie für alle Abstände in ihren Texten nicht nur nicht verantwortlich sein müssen, sondern auch nicht sein sollten. LyX' WYSIWYM ("... mean") ist nicht als Übergang oder Kompromiß zu sehen, es ist vielmehr eine mögliche Lösung.

XML und WYSIWYM vertragen sich hervorragend. LyX würde erheblich robuster und wartbarer, wenn es LaTeX als Basis zugunsten einer XML DTD aufgeben würde. Man hätte dann das beste aus (fast) allen Welten, und direktes LaTeX wäre nur noch für ungewöhnliche Projekte notwendig.

[42]Ein sehr schönes Beispiel dafür ist das h2g2-Projekt [h2g2, 2002] der BBC: Alle Artikel des Lexikons sind in GuideML (einer XML-Anwendung) verfaßt und werden bei Abruf stehenden Fußes in HTML umgewandelt. Dabei kann man zwischen zwei verschiedenen "Skins" (Oberflächen-Stilen) wählen.

[43]what you see is what you mean, siehe [Jackson und Voß, 2001]

[44]Vielleicht werde ich in einigen Jahrzehnten zu einem erheblich jüngeren sagen: »Ja, heute, da diktiert man seine Texte ja direkt ins Buch, aber damals, da haben wir noch alle Ligaturen von Hand aufgebrochen, . . . sah übrigens früher alles besser aus . . . «

14 Hat sich's gelohnt?

(Für mich zeitlich sicher nicht. Was ich in die die Umwandlungen gesteckt habe, werde ich wohl nie mehr herausbekommen. Ich muß zugeben, vom Aufwand sehr unangenehm überrascht worden zu sein. Allerdings hat's auch Spaß gemacht, und außerdem geht es darum ja nicht: Kaum einer, der XML benutzt, wird bei Null anfangen. Er wird bestehende Anwendungen leicht oder nicht verändert benutzen.)

XML ist tatsächlich in der Lage, LaTeX als Schreibwerkzeug abzulösen. Für den Textsatz arbeitet LaTeX jedoch weiter als Backend. Andere Backends, wie z. B. Jade oder FO-Prozessoren, sind nicht in der Lage, anspruchsvolle Typographie zu liefern. Außerdem sind Formeln, Literatur- und Stichwortverzeichnis dort ein großes Problem. An diesen Schwächen wird sich auf absehbare Zeit nichts ändern. XML via XSLT zu high-level LaTeX, das ist der einzig praktikable Weg zum perfekten Druck mit XML.

Wie mein eigenes Projekt tbook leider eindrucksvoll demonstriert, ist eine Komplettlösung ein sehr inhomogener Kraftakt. Verschiedenste Tools, teilweise Handgemachtes, muß einem Flohzirkus gleich gebändigt werden. Das ist jedoch unvermeidlich, da hier Spezialisten am Werk sind, für die es so schnell keinen Rundum-Ersatz geben wird. Außerdem war das bei LaTeX einst auch nicht besser.

Für Browser-Ausgabe ist anspruchsvolle Typographie unwichtig, da ohnehin der Browser der limitierende Faktor ist. Deshalb kommt man auch hier mittels XSLT zu hervorragenden Ergebnissen. Andere automatische Umwandlungen, die von LaTeX aus bislang kaum zu schaffen waren, wie etwa nach RTF oder ins FrameMaker-Format, sind mit geringen Abstrichen möglich.

Als Eingabeformat ist XML glasklar strukturiert, recht simpel und zwingt, sich auf das wesentliche (den Inhalt) zu konzentrieren. Wer die direkte Eingabe solcher unmenschlichen Computersprachen nicht mag: Die strenge Syntax erlaubt XML-Editoren, die viel komfortabler als LaTeX-Editoren sind. Insbesondere können LyX-ähnliche Systeme sehr von XML profitieren.

Einziger Wermutstropfen ist, daß die heute existierenden XML-Anwendungen zur Elfenbeinturm-Kategorie gehören. Ich würde niemals wagen, damit ein wichtiges Projekt zu beginnen, das außerhalb des Feldes der technischen Dokumentation oder des klassischen Dramas liegt.

Die Jahreszahlen im Literaturverzeichnis zeigen, wie heiß die Entwicklung ist. Die Weichen werden jetzt gestellt. Es könnte gut sein, daß viele LaTeXer bald mit XML ein neues Zuhause finden können – vielleicht ohne dabei XML direkt zu sehen.

Was meine eigenen XSLT-Dateien angeht, habe ich meine Ziele von Seite 4 erreicht.

15 Glossar

Normalerweise halte ich nicht viel von einem Glossar; für diesen Text erschien er mir jedoch sinnvoll.

CSS Cascading Style Sheets. Stylesheet-Sprache. Ursprünglich für HTML entwickelt, können sie auch für XML gute Dienste leisten. Sie können ein Dokument nicht irgendwie transformieren, sondern nur zu jedem Element angeben, wie es aussehen soll: Also ob Block oder Inline, Farbe, Schriftart, etc. [CSS2-W3C, 1998; CSS3-W3C, 2002]

DocBook Wahrscheinlich die momentan am häufigsten benutzte XML-DTD. (War ursprünglich eine SGML-DTD.) Damit kann man besonders gut Dokumente schreiben, die im weitesten Sinne im Gebiet der technischen Dokumentation, z. B. von Computer-Programmen, liegen. [DocBook, 2002]

DSSSL (sprich: Dissl) Stylesheet-Sprache, für SGML-Dokumente entwickelt. Damit kann man angeben, wie ein SGML-Dokument, das einer bestimmten DTD entspricht, endgültig aussehen soll, unabhängig vom Ausgabe*format*. [Jade, 1998], der bekannteste DSSSL-Prozessor, erzeugt dann nach Wunsch eine RTF-, Plain TEX-, HTML- oder MIF-Datei (FrameMaker). Kann auch mit Formeln umgehen. [Goossens und Rahtz, 1999, Kap. 7.5]

DTD Eine Datei, in die Syntax einer XML-Anwendung angibt. Da steht dann z. B. drin, daß ein `<author>` ein `<firstname>` und `<lastname>` enthält, oder daß eine `<caption>` nur in einer `<figure>` vorkommen kann. Eine DTD *definiert* eine *XML-Anwendung*, und mit einer DTD kann man eine XML-Datei *validieren*. [Goossens und Rahtz, 1999, Kap. 6.5.4]

Der DTD-Standard wird vielleicht irgendwann durch XML Schema unterstützt.

Element Ein durch zwei sich entsprechende Tags (z. B. `<tabular>` und `</tabular>` eingeklammerter Abschnitt eines XML-Dokuments. Kann Attribute (im Start-Tag), Text-Passagen und Kinder-Elemente enthalten.

Eine XML-Datei besteht aus *einem* Element (Wurzelelement).

FO-Prozessor Programm, das eine XML-Datei, die Formating Objects (XSL:FO) enthält, in ein Ausgabeformat, z. B. PDF, HTML, RTF oder Plain Text, umwandeln kann.

Beliebte Freeware FO-Prozessoren sind PassiveTEX und FOP [PassiveTEX, 2002; FOP, 2002]. Allerdings sind beide als Prä-Alpha einzustufen.

gültig (engl. "valid") Ein XML-Dokument *muß* wohlgeformt und *kann* gültig sein.[45] Es ist gültig, wenn es anhand einer DTD, die es nennt, erfolgreich *validiert* werden kann.

HTML Formal eine SGML-Anwendung, wird vielleicht bald in Form vom neuen XHTML zu einer XML-Anwendung. Und ansonsten, naja, eben das Dateiformat aller Web-Dokumente.

Jade *Der* DSSSL-Interpreter. Siehe dort.

MathML Eine XML-Anwendung, die als Modul zu anderen Anwendungen hinzugefügt werden kann und mit der man mathematische Formeln schreiben kann. [MathML-W3C, 2001]

SGML Standard Generalized Markup Language von 1986. Vorgänger von XML und gleichzeitig eine (riesige) Obermenge desselben.

SVG Scalable Vector Graphics.[46] Diese XML-Anwendung soll schöne bunte Vektor-Bildchen in die Welt des World Wide Webs bringen, das bislang von Bitmaps und Flashs geprägt war. Es läßt sich wie MathML perfekt in HTML einbetten.

Tag Kann z. B. `<book>` oder `<br/>` sein.

TEI Text Encoding Initiative. Internationales Projekt, um Texte aller Art in den Computer zu bringen, insbesondere aus dem Bereich der Philologie. Es ist eine riesige SGML-Anwendung. Eine spacke Untermenge, TEI Lite [TEI Lite, 2001], ist auch als XML-Anwendung zu haben. Dafür gibt seit kurzem auch XSL-Stylesheets.

Unicode Ein Super-Latin-1 mit 65536 Zeichenpositionen, von denen aber lange nicht alle belegt sind (wäre ja auch Unsinn ...). Der neueste Emacs 21 kommt damit zurecht, außerdem moderne Browser, ansonsten ist die Unterstützung eher dürftig. Die praktische Umsetzung geschieht allermeist mit UTF-8. XML-Tools *müssen* mit Unicode zurecht kommen können. (Goossens und Rahtz, 1999, Anh. C.2 und Unicode, 2002)

[45] Einige sprechen im Deutschen von „valide", naja.

[46] Klasse, nicht? "Scalable". Wir alle erinnern uns noch an die dunkle Zeit *nicht*-skalierbarer Vektorgrafiken.

UTF-8 Ein Textdatei-Format, besser gesagt eine Art und Weise, Unicode-Zeichen in eine Datei zu schreiben. Dafür braucht man ja *zwei* Byte (65535 Zeichen!), nicht wie früher *ein* Byte.

Damit das nicht zu verschwenderisch wird und bestehende Editoren wenigstens eine Chance erhalten, werden alle ASCII-Zeichen genauso kodiert wie früher. Umlaute und scharfes S werden jedoch zu Zwei-Byte-Sequenzen. Das ganze geht auf Kosten des fernen Ostens, dessen Zeichen in UTF-8 drei Byte lang werden. [Goossens und Rahtz, 1999, Anh. C.2.3]

Validieren Das ist der Vergleich der Syntax eines wohlgeformten XML-Dokuments mit dem, der in der zugehörigen DTD beschrieben ist. Wenn's übereinstimmt, ist's ein *gültiges* XML-Dokument, sonst nicht.

W3C World Wide Web Consortium. Die geben die normativen Richtlinien zu XML, HTML und anderen Web-Standards heraus. `http://www.w3c.org`

wohlgeformt Ein XML-Dokument ist wohlgeformt, wenn es dem XML-Standard entspricht. Hört sich erstmal blöd an, schließlich spricht ja auch keiner von wohlgeformten TeX-Dateien. Man will sich damit vom *gültigen* XML-Dokument absetzen, das ja noch mehr bedeutet.

XML-Anwendung Dieser Begriff ist sehr beliebt (und ich glaube sogar offizieller Natur) und etwas verwirrend. Er bezeichnet *nicht* ein XML-Tool, auch nicht das bloße Benutzen von XML, sondern schlicht und ergreifend eine DTD. DocBook, TEI, XHTML, MathML, das sind alles XML-Anwendungen, die durch ihre DTD definiert sind.

XML Extensible Markup Language. Untermenge von SGML, logische Auszeichnungssprache für Dokumente und Datenbanken. [XML-Deutsch, 2000; Partl, 2000]

XML Schema Eine Weiterentwicklung der DTD. Man kann nicht gerade sagen, daß man sich um diesen Standard reißt, die meisten sind wohl mit ihren DTDs ganz zufrieden (ich auch). Aber langfristig kommt man wohl nicht drum herum. [Schema-W3C, 2001]

XPath Ein Standard, der XSLT zuarbeitet. Damit sind z. B. String-Manipulationen in XSLT möglich, vor allem aber muß man ja bei einer Trafo von XML nach XML die Elemente im Quell-XML irgendwie anwählen können. Das macht man mit XPath. [XPath-Deutsch, 1999]

XSL Ein zwei-Teile-Standard für die Verarbeitung von XML-Dateien: *XSLT* kann ein XML-Dokument inhaltlich neu strukturieren, bevor es dann eventuell mit *XSL:FO* formatiert wird.

XSL:FO Die XSL Formatierungs-Objekte. Was CSS für HTML, DSSSL für SGML, das sind die FOs für XML. Nur leider ist es gar nicht so dumm, CSS oder DSSSL auch für XML zu benutzen. Mal sehen, was aus den FOs wird. [XSL-W3C, 2001]

Sie kommen im Gegensatz zu DSSSL nicht mit Formeln zurecht, die müssen die FO-Prozessoren gesondert behandeln, was kaum einer macht außer PassiveTeX.

XSLT Hinreißende Möglichkeit, XML-Dateien in andere XML-Dateien, in HTML oder LaTeX, auch Plain Text, umzuwandeln. [XSLT-W3C, 1999]

XSLT-Prozessor Ein Programm, das anhand eines XSLT-Stylesheets ein XML-Dokument umwandelt, eigentlich in eine andere XML-Anwendung. Aber auch HTML oder Plain Text sind mögliche Ausgabeformate. Dabei kann, muß aber nicht validiert werden.

Beliebte XSLT-Prozessoren sind [Saxon, 2002], [Xalan, 2001] und der gute alte [XT, 2002].

Literatur

[BibTEXML 2001] PREVITALI, Luca ; LURATI, Brenno ; WILDE, Erik: *An XML Representation of BibTEX*. 2001. – URL `http://dret.net/netdret`. – URL ist Web-Seite von Erik Wilde 70, 78

[CIQ 2002] OASIS: *OASIS – Technical Committees – CIQ*. April 2002. – URL `http://www.oasis-open.org/committees/ciq/ciq.shtml`. – *Die* Adreß-DTD für Masochisten 70

[Clark 1999] CLARK, James: *XSLT in Perspective*. Juli 1999. – URL `http://www.jclark.com/xml/xslt-talk.htm`. – Vorlesungsskript 9

[Cottrell 2001] COTTRELL, Allin: *Textverarbeitungen: Dumm und ineffizient*. Oktober 2001. – URL `http://www.ecn.wfu.edu/~cottrell/wp/german.html`. – Übersetzt von Hannes Keil und Claudia Thormann 79

[Cowan 2000] COWAN, John: *XML Catalog proposal, draft 0.4*. September 2000. – URL `http://home.ccil.org/~cowan/XML/XCatalog.html` 67

[CSS2-W3C 1998] BOS, Bert ; LIE, Håkon Wium ; LILLEY, Chris ; JACOBS, Ian u. a.: *Cascading Style Sheets, Level 2*. Mai 1998. – URL `http://www.w3.org/TR/REC-CSS2` 9, 80

[CSS3-W3C 2002] BOS, Bert ; LIE, Håkon Wium ; LILLEY, Chris ; JACOBS, Ian u. a.: *Cascading Style Sheets, Level 3*. Mai 2002. – URL `http://www.w3.org/Style/CSS/current-work`. – Under construction 9, 80

[DB2LATEX 2002] CASELLAS, Ramon: *DB2LATEX*. Januar 2002. – URL `http://db2latex.sourceforge.net`. – XSLT Style Sheets für DocBook nach LATEX Trafo 5, 72

[DocBook 2002] WALSH, Norman: *DocBook*. April 2002. – URL `http://www.docbook.org`. – Offizielle DocBook Homepage 5, 67, 80

[dtd2xs 2002] SCHWEIGER, Ralf: *dtd2xs*. März 2002. – URL `http://puvogel.informatik.med.uni-giessen.de/dtd2xs/`. – Java-Konvertierer zwischen DTD und XML Scheme 13

[Fitzgerald 2001] FITZGERALD, Michael: *XSL Essentials*. Wiley, 2001 9

[FOP 2002] Apache XML Project (Veranst.): *FOP*. Mai 2002. – URL `http://xml.apache.org/fop/index.html`. – Der Formating-Objects-Prozessor FOP 9, 81

[GELLMU 2002] HAMMOND, William F.: *GELLMU – Introductory Survey – A Bridge for Authors from LATEX to XML*. März 2002. – URL `http://www.albany.edu/~hammond/gellmu/` 7

[Goossens und Rahtz 1999] GOOSSENS, Michel ; RAHTZ, Sebastian: *Mit LATEX ins Web*. Addison-Wesley, 1999 8, 12, 71, 81, 82

[Gurari 2000] GURARI, Eitan M.: *XSLT from XHTML+MathML to LATEX*. Juli 2000. – URL `http://www.cis.ohio-state.edu/~gurari/docs/mml-00/xhm2latex.html` 72

[h2g2 2002] British Broadcasting Company: *h2g2*. 2002. – URL `http://www.bbc.co.uk/dna/h2g2/`. – Letzter Slash im URL ist wichtig 79

[Jackson und Voß 2001] JACKSON, Laura E. ; VOSS, Herbert: LyX – Open Source Document Processor, Teil 1. In: *Die TEXnische Komödie* 3/2001 (2001) 79

[Jade 1998] CLARK, James: *Jade – James' DSSSL Engine*. Oktober 1998. – URL http://www.jclark.com/jade/ 81

[jpeg2ps 1999] MERZ, Thomas: *jpeg2ps – convert JPEG compressed images to PostScript Level 2*. Juli 1999. – URL ftp://ftp.dante.de/tex-archive/nonfree/support/jpeg2ps 52

[Kastrup u. a. 2002] KASTRUP, David u. a.: *Preview LATEX: Emacs/LATEX inline Preview*. August 2002. – URL http://sourceforge.net/projects/preview-latex/ 23

[Knuth und Levy 2001] KNUTH, Donald E. ; LEVY, Silvio: *The CWEB System of Structured Documentation*, Januar 2001. – URL http://www-cs-faculty.stanford.edu/~knuth/cweb.html 52, 62

[Lie 1999] LIE, Håkon: *Formatting Objects considered harmful*. April 1999. – URL http://people.opera.com/howcome/1999/foch.html 9

[MathML-c2p 2001] NAMIKI, Takao: *MathMLc2p*. April 2001. – URL http://www.math.sci.hokudai.ac.jp/~nami/MathML/party2/node12.html. – XSLT für Contents → Presentation Markup von MathML; wahrscheinlich entnommen aus dem Mozilla-Projekt 77

[MathML-W3C 2001] CARLISLE, David ; ION, Patrick ; MINER, Robert ; POPPELIER, Nico: *Mathematical Markup Language (MathML), Level 2*. Februar 2001. – URL http://www.w3.org/TR/MathML2. – Die DTD gibt's unter http://www.w3.org/TR/MathML2/DTD-MathML-20010221.zip 71, 81

[Partl 2000] PARTL, Hubert: *XML Kurz-Info*. September 2000. – URL http://www.boku.ac.at/htmleinf/xmlkurz.html. – Kurzeinführung in XML 4, 82

[PassiveTEX 2002] RAHTZ, Sebastian: *PassiveTEX*. Februar 2002. – URL http://www.tei-c.org/Software/passivetex 9, 81

[qwertz 1997] GORDON, Thomas F.: *The qwertz Project*. Oktober 1997. – URL http://www.chemie.fu-berlin.de/chemnet/use/suppl/sgml/qwertz-sgml.html 8

[RFC [2070] 1997] YERGEAU, Frangois ; NICOL, Gavin T. ; ADAMS, Glenn ; DUERST, Martin J.: *Request for Comments #2070: Internationalization of the Hypertext Markup Language*. Januar 1997. – URL http://www.ietf.org/rfc/rfc2070.txt 20

[RFC [3066] 2001] ALVESTRAND, Harald T.: *Request for Comments #3066: Tags for the Identification of Languages*. Januar 2001. – URL http://www.ietf.org/rfc/rfc3066.txt 68

[Saxon 2002] KAY, Michael: *The SAXON XSLT Processor*. 2002. – URL http://saxon.sourceforge.net 53, 55, 67, 82

[Schema-W3C 2001] FALLSIDE, David C.: *XML Schema Part 0: Primer*. Mai 2001. – URL http://www.w3c.org/TR/xmlschema-0/ 13, 82

[TEI Lite 2001] BURNARD, Lou: *TEI U5: Encoding for Interchange: an introduction to the TEI*. Januar 2001. – URL http://www.tei-c.org/Lite. – TEI Lite Homepage 5, 81

[Unicode 2002] Editorial Committee of Unicode, Inc.: *Unicode 3.2*. Januar 2002. – URL http://www.unicode.org 81

[unicode-Paket 2000] UNRUH, Dominique: *Das Unicode-Paket*. 2000. – URL ftp://ftp.dante.de/tex-archive/macros/latex/contrib/supported/unicode 61

[Xalan 2001] Apache XML Project: *Xalan-C++ version 1.3*. 2001. – URL http://xml.apache.org/xalan-c/index.html. – Ein XSLT-Prozessor 13, 56, 82

[Xerxes 2002] Apache XML Project: *Xerxes-C++*. April 2002. – URL http://xml.apache.org/xerces-c/index.html. – Ein validierender XML Prozessor 13

[xindy 2001] KEHR, Roger ; SCHROD, Joachim: *xindy – A Flexible Indexing System*. Oktober 2001. – URL http://sourceforge.net/projects/xindy/ 54

[XML-Deutsch 2000] BRAY, Tim ; PAOH, Jean ; SPERBERG-MCQUEEN, C. M.: *Extensible Markup Language (XML) 1.0*. Oktober 2000. – URL http://www.mintert.com/xml/trans/REC-xml-19980210-de.html. – Ins Deutsche übersetzt von Henning Behme und Stefan Mintert 4, 82

[XML-W3C 2000] BRAY, Tim ; PAOLI, Jean ; SPERBERG-MCQUEEN, C. M.: *Extensible Markup Language (XML) 1.0*. Oktober 2000. – URL http://www.w3.org/TR/2000/REC-xml-20001006 4

[XMLCatalog 2001] WALSH, Norman (Hrsg.): *XML Catalogs – Committee Specification 06 Aug 2001*. August 2001. – URL http://www.oasis-open.org/committees/entity/spec.html 67

[XMLTEX 2000] CARLISLE, David: *XMLTEX*. 2000. – URL ftp://ftp.dante.de/tex-archive/macros/xmltex 61

[XPath-Deutsch 1999] CLARK, James ; DEROSE, Steve: *XML Path Language (XPath) Version 1.0*. Dezember 1999. – URL http://www.informatik.hu-berlin.de/~obecker/obqo/w3c-trans/xpath-de/. – Ins Deutsche übersetzt von Oliver Becker 82

[XPath2-W3C 2002] KAY, Michael ; BERGLUND, Anders ; FERNANDEZ, Mary F. u. a.: *XML Path Language (XPath) 2.0*. April 2002. – URL http://www.w3.org/TR/xpath20. – Working Draft 77

[XSL-W3C 2001] DEACH, Stephen ; GROSSO, Paul ; ZILLES, Steve u. a.: *Extensible Style Sheet Language (XSL)*. Oktober 2001. – URL http://www.w3.org/TR/xsl 9, 82

[XSLT-W3C 1999] CLARK, James: *XSL Transformations*. November 1999. – URL http://www.w3.org/TR/xslt 9, 82

[XSLT2-W3C 2002] KAY, Michael: *XSL Transformations (XSLT) Version 2.0*. April 2002. – URL http://www.w3.org/TR/xslt20/. – Working Draft 77

[XT 2002] LINDSEY, Bill ; CLARK, James: *XT*. April 2002. – URL http://www.blnz.com/xt/index.html. – Ein XSLT-Prozessor 56, 82