Publishing mit XML

Workflow und Tools

Jochen Hein

1. Warum Publishing mit XML?

Publizieren mit Hilfe von XML-Formaten hat im Vergleich zu anderen Formaten Vor- und Nachteile. In den folgenden Abschnitten werden wir XML mit Office-Formaten und LaTeX vergleichen und dabei die verschiedenen Aspekte beleuchten.

Heute ist es vielfach notwendig, aus einem Dokument(enformat) eine Vielzahl von Ausgabeformaten zu erzeugen. Insbesondere kommen dabei HTML und PDF in Betracht, aber auch einfache Text-Dateien sind gefragt. Dabei sollten die Formate möglichst automatisch erzeugt werden können. Eine Orgie wie "Textprogramm starten", "Dokument öffnen", "Speichern unter" ist aufwändig und fehleranfällig. Für die verschiedenen Formate sollten keine speziellen Anpassungen der Quelle notwendig sein.

Mit der passenden XML-Struktur (Document-Type-Definition oder XML-Schema) ist es möglich, vollständig "logisches Markup" zu verwenden. Damit werden die Objekte im Text markiert, aber nicht deren Aussehen in den verschiedenen Formaten definiert. Diese Definition ist Aufgabe der Stylesheets. In traditionellen Office-Programme verwenden die meisten Anwender visuelles Markup, auch wenn mit der passenden Dokumentenvorlage logisches Markup möglich ist. Bei LaTeX findet man eine Mischform zwischen logischem und visuellem Markup - ein vollständig logisches Markup ist nur mit Hilfe von eigenen Makros möglich.

Dadurch dass XML-Dateien einfache, strukturierte Text-Dateien sind ist es einfach, diese Dateien zu bearbeiten (zum Beispiel mit Editoren oder Skripten) oder zu konvertieren. Dabei können eigene Tools oder Standard-Komponenten (XML-Parser, XSL-Prozessoren, Perl etc.) eingesetzt werden.

Mit einer geeigneten DTD-Familie können ähnliche Strukturen für Artikel, Bücher, Präsentationen und Webseiten verwendet werden. Mit DocBook steht ein derartiges Format frei zur Verfügung. Damit ist es für den Autor einfach, diese Formate zu verwenden.

Die Transparenz von XML (in Verbindung mit einer dokumentierten und stabilen DTD= sorgt dafür, dass kein Vendor-Lock-In stattfindet. Denn vielfach haben die in einem Unternehmen erzeugten und gepflegten Dokumente einen höheren Wert als die Lizenzen der beteiligten Programme - und dennoch kann der Software-Hersteller dem Kunden gewisse Vorschriften machen.

Auch wenn heute klassische Office-Pakete die Ausgabe von Dateien in verschiedenen Formaten (u.a. HTML) ermöglichen und frei verfügbare Druckertreiber zum Erstellen von PDF-Dateien existieren, so ermöglicht nur ein wirklich transparetes Format die freie Wahl des Ausgabe-Formates. Denn nur dann ist der Anwender in der Lage, spezielle Anforderungen erfüllen zu können.

2. XML in Kürze

XML besteht aus einer Reihe von Standards und Empfehlungen des W3C (http://www.w3c.org). Begonnen wurde XML als eine abgespeckte SGML-Version; heute ist XML mit allen Komponenten viel komplexer als SGML. Dennoch stehen heute mehr Tools zur Verfügung, um XML-Dateien zu bearbeiten als für SGML. Allerdings: Wenn man nur die Grundzüge von XML und SGML verwendet, so kann man beide

Arten von Tools einsetzen und damit die am besten geeigneten Tools einsetzen.

Bei XML-Dokumenten spricht man von zwei wesentlichen Arten von Dokumenten:

wohlgeformt
: Das Dokument entspricht den formalen Regeln von XML, im wesentlichen, dass jedes Tag auch geschlossen wird.

valide
: Das Dokument entspricht den formalen Regeln von XML und seine Struktur passt zur Document Type Definition bzw. dem XML-Schema.

Alle anderen Dokumente mögen so aussehen wie XML, sind es aber nicht. So ähnlich, wie man das auch von HTML-Dateien her kennt, allerdings sind die XML-Tools penibler als die meisten Web-Browser.

Zur Konvertierung von XML-Dokumenten in XML oder andere Formate stehen XSL-Prozessoren zur Verfügung. Diese sind sowohl frei als auch kommerziell erhältlich, zum Teil sind diese auch in Programmiersprachen (wie die ABAP/4) integriert oder stehen als Bibliothek zur Verfügung..

3. DocBook als Format

Für praktisch alle Veröffentlichungen im Computerumfeld kann die DocBook-DTD verwendet werden. Diese enthält vordefinierte Strukturen für Artikel, Bücher und Manpages. Für die Ausgabe stehen Stylesheets für HTML, PDF, RTF und andere Formate zur Verfügung.

Die Entwicklung von DocBook begann als SGML-Applikation, u.a. ins Leben gerufen von O'Reilly. Im Laufe der Zeit haben viele Anwender DocBook eingesetzt, darunter zum Beispiel Sun. DocBook wurde von der Davenport Group weiterentwickelt und wird heute offiziell von OASIS (Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/docbook) gepflegt. Viele Entwickler sind seit der Anfangszeit dabei, eine zentrale Figur ist Norm Walsh.

Norm ist nicht nur an der Entwicklung der DTD beteiligt, sondern auch an der Implementierung der Stylesheets. Mit diesem Stylesheets ist es auch für unbedarfte Anwender möglich, DocBook einzusetzen - auch ohne intensive Kenntnisse in XML, SGML, XSLT oder DSSSL.

Viele Freie Software Projekte haben begonnen, Teile der Dokumentation in DocBook zu veröffentlichen, als Beispiel dafür seien FreeBSD und GNOME genannt. Damit liegen die Dokumentationen ist einem transparenten, gut zu bearbeitenden Format vor und können von mehreren Entwicklern bearbeitet werden; Revisionen können einfach in der Versionsverwaltung des Projektes gespeichert werden. Insbesondere das FreeBSD-Projekt übersetzt die Dokumentation in mehrere Sprachen, auch hierfür steht zum Beispiel mit **poxml** ein hilfreiches Tool zur Verfügung.

Für spezielle Einsatzgebiete gibt es Erweiterungen der DocBook-DTD für Webseiten im HTML- oder XHTML-Format http://docbook.sourceforge.net/projects/website/index.html und Präsentationen [http://docbook.sourceforge.net/projects/slides/index.html]. Das sind Customizations, also Anpassungen, die neue Elemente (Tags) hinzufügen und nicht benötigte entfernen, sowie die Dokumentenstruktur entsprechend anpassen. Damit kann das Wissen um die allgemeine DocBook-Dokumentenstruktur auch für diese Formate verwendet werden. Der zweite Teil der Customizations besteht aus Anpassungen der Stylesheets, so dass zum Beispiel je Präsentationsfolie eine HTML-Seite erzeugt wird und diese Seiten untereinander verlinkt werden.

Nachteilig ist, dass DocBook eine große, für Anfänger recht unübersichtliche DTD ist. Hilfreich sind hier Editoren, die entwedern eine Eingabehilfe für Tags bereit stellen oder gar passende Menüs oder

Dokumentenvorlagen enthalten. Eine andere Möglichkeit wäre hier der Einsatz von simplified DocBook, ein einfacher DocBook-Subset.

Bei der Weiterentwicklung von DocBook wird darauf geachtet, dass nur dann inkompatible Änderungen vorgenommen werden, wenn diese lange vorher angekündigt waren. Die Regel ist, dass ein Objekt erst eine Major-Version als "veraltet" dokumentiert sein muss, damit es in der darauf folgenden Version entfernt werden kann. Damit können Anwender in Ruhe auf das jeweils neue Format migrieren. Auch diese Stabilität hilft bei der Entscheidung für das Format DocBook - es ist nicht zu erwarten, dass alle Dokumente in absehbarer Zeit mühsam manuell konvertiert werden müssen. Für die meisten Anpassungen wird es möglich sein ein einfaches XSL-Skript einzusetzen.

4. Workflow bei der Arbeit mit XML-Dokumenten

In Abbildung 1, „Übersichtsdiagramm zum Workflow mit XML-Dokumenten“ [4] (aus dem DocBook Tutorial, [lit-docbook-tutorial] [13]) finden Sie einen (unvollständigen) Überblick über die verschiedenen Ausgabeformate für DocBook und die Wege dahin.

XML- oder SGML-Dokumente können mit einem beliebigen Editor erstellt werden. Hilfreich ist es, wenn dieser die Strukturen von XML/SGML kennt und den Anwender beim Einfügen der Tags unterstützt.

Viele der Tools stehen als Kommandozeilentools zur Verfügung. Damit ist eine Automatisierung der Abläufe schnell und einfach einzurichten. Viele Unix-Anwender verwenden dafür **make**, Java-Anwender verwenden eher Ant. Beispiele für Makefiles finden Sie zum Beispiel in den Vorträgen auf meiner Webseite (http://www.jochen.org/~jochen/vortraege.html" [http://www.jochen.org/~jochen/vortraege.html]).

Einige Ausgabeformate können auf verschiedenen Wegen erreicht werden. Manchmal, indem man XML- und SGML-Tools einsetzt, manchmal stehen mehrere Tools (wie FO-Prozessoren) zur Verfügung und manchmal kann das Ergebnis nochmals umgesetzt werden. In diesen Fällen lohnt es sich, die verschiedenen Ergebnisse zu vergleichen, bevor man sich auf ein Tool festlegt.

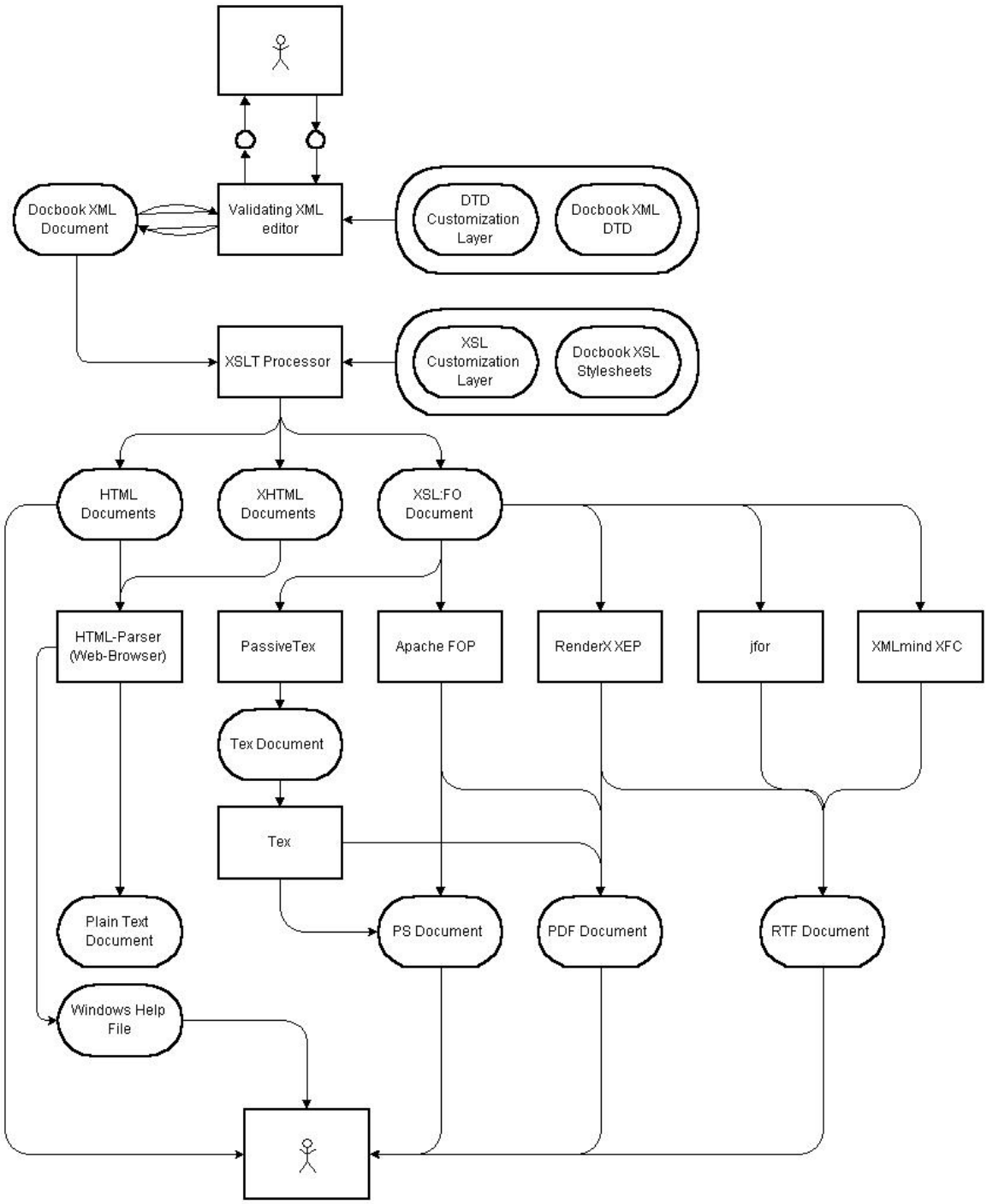

Abbildung 1. Übersichtsdiagramm zum Workflow mit XML-Dokumenten

5. SGML-Workflow

Bei SGML Dokumenten kommt als Stylesheet-Sprache DSSSL zum Einsatz. Ein frei verfügbarer Interpreter dieser Sprache ist **jade** bzw. **openjade**. Es stehen Stylesheets zum Umwandlung in HTML, RTF oder jadetex zur Verfügung. Die Print-Ausgabe mit **jadetex** ist in manchen Fällen immer noch besser als die Verwendung eines freien FO-Prozessors. Da die meisten XML-Dokumente auch mit den SGML-Tools verarbeitet werden können hat der Anwender die Qual der Wahl.

6. Automatisierung

Wichtig für den effizienten Einsatz von XML/SGML ist die Automatisierung all der Schritte, die zur fertigen Ausgabe führen. Nur damit ist ein ausreichendes Maß an Akzeptanz zu erreichen. Zur Automatisierung kann **make** oder Ant zum Einsatz kommen. Für beides findet man im Internet Beispiele u.a. die Dokumente zu diesem Vortrag unter http://www.jochen.org/~jochen/vortraege.html.

7. Tools für XML und SGML

Die hier vorgestellten Tools sind eine Auswahl der im DocBook Wiki [http://www.docbook.org/wiki/moin.cgi/FrontPage] erwähnten Programme. Schauen Sie dort einfach mal nach, was es sonst so gibt.

7.1. Editoren

Als XML-Editor kann im Prinzip jeder ASCII- oder Unicode-Editor verwendet werden. Dennoch ist es hilfreich, wenn der Editor das Einfügen oder Ändern von Tags unterstützt. Damit ist es für den Anwender einfacher, valide Dokumente zu erzeugen. Verbreitet ist der Einsatz von Emacs mit dem PSGML-Mode, es werden aber auch spezielle Editoren wie **conglomerate** eingesetzt. Das hängt sehr von den Vorlieben der Anwender ab.

DocBook beschreibt logisches Markup, dennoch möchten viele Anwender schon beim Schreiben eine Darstellung haben, die der späteren Darstellung im Ausdruck nachkommt. Das ist bei XML-Dokumenten nicht einfach zu realisieren, aber vielleicht sind die DocBook-Filter von OpenOffice.org für Sie interessant.

7.2. XSL-Prozessoren

Es stehen heute eine Reihe von XSL-Prozessoren zur Verfügung, die unter Unix am häufigsten verwendeten sind wohl **xsltproc** und **saxon**. Beide sind stabil und konform zu den XML-Standards. **xsltproc** ist in C geschrieben und sehr schnell, dafür können hier keine in Java geschriebenen Erweiterungen eingebinden werden. Diese Erweiterungen stehen häufig für Prozessoren wie **saxon** zur Verfügung.

Hilfreich ist außerdem der Einsatz eines Validators, der das Dokument auf Übereinstimmung zur verwendeten DTD prüft. Damit lassen sich auch ohne vollständige (und manchmal langlaufende) Konvertierung des Dokumentes Fehler in der Dokumentenstruktur aufdecken. Ein solcher Validator ist **xmllint**.

7.3. Darstellung von XML in Browsern

Für die Darstellung von einfachen Dokumenten ist vielfach kein XSL-Stylesheet notwendig. Statt dessen

kann ein CSS-Style verwendet werden. Damit können XML-Dokumente in Web-Browsern anprechend dargestellt werden - auch wenn nicht alle Features einer DTD unterstützt werden.

Für eine relativ Browser-unabhängige Darstellung empfiehlt es sich allerdings, das Dokument in HTML zu konvertieren und in diesem Format zu veröffentlichen. Noch sind viele CSS-Features in den Browsern nicht implementiert.

7.4. XML-FO: Formatting Objects

Im Rahmen von XML wurde mit XML-FO ein Format zur Beschreibung des Layouts von Dokumenten entwickelt. Dieses ist wiederum XML und kann mit FO-Prozessoren zum Beispiel nach PDF konvertiert werden.

Als freie Software stehen zwei FO-Prozessoren zur Verfügung: **fop** (aus dem Apache-Projekt) und **passivetex** (als TeX-Aufsatz). Beide sind für den "Hausgebrauch" einsetzbar, aber haben immer noch ihre Schwächen.

Für anspruchsvolle Anwender steht mit XEP allerdings eine leistungsfähige (kommerzielle) Lösung bereit.

8. Stylesheet-Customization

DocBook stellt eine große Sammlung an Tags zur Verfügung, die Stylesheet implementieren in der Regel ein "brauchbares" Layout. Jedes größere Projekt wird jedoch an Grenzen der Stylesheets stoßen und diese an eigene Anforderungen anpassen.

Der intuitive Weg, die Stylesheets zu ändern, wird spätestens mit der nächsten Version derselben zu erheblicher Arbeit führen. Aus ist es nur mit großem Aufwand möglich, diese Änderungen an andere Anwender weiter zu geben oder zum Beispiel im Rahmen eines Bug-Reports zu veröffentlichen.

Die DocBook-Styles sind so entwickelt, dass Anwender recht einfach eigene Anpassungen vornehmen können. Die folgenden Aussagen und Beispiele beziehen sich auf die XSL-Version. Die SGML/ DSSSL-Styles sind in einer Lisp-Abart geschrieben und anpassbar, was wir hier nicht näher betrachten.

Für das Customizing verwendet man sogenannte Customization-Layer (eine eigene Style-Datei). Diese binden intern die Standard-Styles ein und setzen Parameter oder überschreiben Templates. Da DocBook/ XSL verschiedene Styles für HTML- und FO-Ausgabe verwendet wird man auch zwei Customization-Layer verwenden. Ein Beispiel dafür finden Sie in Abbildung 2, „HTML customization“ [6] und Abbildung 3, „FO customization“ [7]. In diesen kann allerdings eine gemeinsam verwendeter Grundstock an Anpassungen als Datei eingebunden werden.

```
<?xml version='1.0'>
<xsl:stylesheet
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  version="1.0">
  <xsl:import href="html/docbook.xsl"/>
  <xsl:include href="common-customizations.xsl" />
  <xsl:param name="html.stylesheet" select="'corpstyle.css'"/>
</xsl:stylesheet>
```

Abbildung 2. HTML customization

```
<?xml version='1.0'>
<xsl:stylesheet
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  version="1.0">
  <xsl:import href="fo/docbook.xsl"/>
  <xsl:include href="common-customizations.xsl" />
  <xsl:param name="paper.type" select="'A4'"/>
</xsl:stylesheet>
```

Abbildung 3. FO customization

Wenn man einen Customization-Layer entwickelt hat, dann ruft man den XSL-Prozessor mit diesem Style anstelle der Original-Dateien auf:

xsltproc --output myfile.html mystyles.xsl myfile.xml

Für viele Anpassungen sind bereits Parameter vorgesehen, die der Anwender in seinen Stylesheets aufnehmen kann. Die Datei `param.xml` in den Stylesheet-Verzeichnissen beschreibt diese Parameter genauer. Ein Beispiel für Parameter sehen Sie in Abbildung 4, „Beispiel für Stylesheet-Parameter“ [7]. Hier wird die Trennlinie zwischen Kopfzeile und Text sowie Text und Fußzeile nicht ausgegeben.

```
<xsl:param name="header.rule">0</xsl:param>
<xsl:param name="footer.rule" select="0"/>
```

Abbildung 4. Beispiel für Stylesheet-Parameter

Sollten die Parameter nicht ausreichen oder für die gewünschte Änderung keine Parameter vorgesehen sein, so muss man die Templates ändern. Dafür müssen die Originale nicht verändert werden, es reicht, wenn Sie die Templates neu definieren. Ein ganz einfaches Beispiel sehen Sie in Abbildung 5, „Beispiel für Templates“ [7], in dem ich die Erzeugung von Kopf- und Fußzeilen abgeschaltet habe. Das ist eine Voraussetzung für die Aufnahme dieses Dokumentes in den Konferenzband.

```
<xsl:template name="header.content"></xsl:template>
<xsl:template name="footer.content"></xsl:template>
```

Abbildung 5. Beispiel für Templates

Da die Stylesheets sehr modular aufgebaut sind und recht lesbar geschrieben sind, ist es recht einfach, das richtige Template zu finden, in den Customization-Layer zu kopieren und nach eigenen Wünschen anzupassen.

Bei einer neuen Version der Styles ist es immer mal wieder notwendig, die einzelnen Customizations zu prüfen und bei Bedarf anzupassen. Allerdings sind alle Änderungen zentral vorhanden, so dass sich der Aufwand dafür in Grenzen hält.

Die Entwickler der Stylesheets sind jedoch für Hinweise auf Fehler und im Einzelfall fehlende Erweiterungsmöglichkeiten dankbar, so dass möglicherweise in einer folgenden Version der Customization-Layer deutlich vereinfacht werden kann.

9. Erfahrungen aus Buch-Konvertierung von LaTeX nach XML

Das Original-Manuskript lag in LaTeX mit verschiedenen eigenen Styles vor. Viele LaTeX-Konstrukte, die normalerweise visuelles Markup verwenden, wurden in eigene Makros gekapselt, deren Name logisches Markup repräsentierte. Mit Hilfe des Tools **ltx2x** und eigenen Umsetz-Regeln wurde das Manuskript konvertiert.

Dabei waren einige Klippen zu umschiffen:

- Der Umlaut "Ö" führte dazu, dass das Tool in eine Endlos-Schleife geriet. Mit Hilfe eines **sed**-Skriptes wurden Umlaute durch LaTeX-Makros ersetzt und diese passend konvertiert.
- Makros, für die noch keine Konvertierungsregeln angelegt wurde, werden still und leise ignoriert. Eine manuelle Prüfung der entstehenden XML-Datei ist damit unbedingt notwendig - möglichst bevor manuelle Änderungen durchgeführt werden.
- Bei der Konvertierung der Tabellen war eine einfache Makro-Ersetzung nicht ausreichend. Durch einige Makros ließen sich manuelle Änderungen aber auf das Löschen überflüssiger Tags beschränken.

Für eine dauerhafte Pflege des Dokumentes in LaTeX und das regelmäßige Erzeugen der DocBook-Quellen ist dieses Verfahren nicht geeignet. Für die einmalige Umstellung und die ohnehin erforderliche (inhaltliche) Arbeit mit dem Manuskript waren der Aufwand und das Ergebnis angemessen.

10. Manuskripte in andere Formate konvertieren

Verlage verwenden gelegentlich eigene Document-Type-Descriptions. Diese können Ähnlichkeiten zu anderen DTDs enthalten, müssen das aber nicht. Je näher die DTD in der Dokumentenstruktur am Originalformat ist, desto einfacher wird die Konvertierung. Die Umsetzung erfolgt mit Hilfe eines XSLT-Skriptes.

Bei der Publikation bei Addison-Wesley (Pearson Education) kam die verlagseigene Pearson-DTD zum Einsatz. Diese ist in der Struktur DocBook relativ ähnlich, so dass eine erste Konvertierung recht einfach war. Im Detail tauchten dabei jedoch verschiedene Schwierigkeiten auf:

- Anhänge werden in DocBook durch jeweils ein Tag `appendix` markiert. In der verlagseigenen-DTD werden die Anhänge durch `chapter` in einer `appendix`-Umgebung markiert.

  Bei der Umsetzung zwischen den Formaten muss nun der erste Anhang durch `appendix` und `chapter` ersetzt werden. Alle anderen Anhänge werden dann als `capter` markiert. Die Verarbeitung aller Anhänge wird dann mit dem Ende-Tag geschlossen. Das passende Stylesheet-Fragment sehen Sie in Abbildung 6, „Beispiel für die Umsetzung der Anhänge“ [9].

```
<xsl:template match="appendix[1]">
  <appendix>
    <chapter>
      <xsl:call-template name="transform.id.attribute"/>
      <xsl:apply-templates/>
    </chapter>
    <xsl:for-each select="following-sibling::appendix">
      <chapter>
        <xsl:call-template name="transform.id.attribute"/>
        <xsl:apply-templates/>
      </chapter>
    </xsl:for-each>
  </appendix>
</xsl:template>

<xsl:template match="appendix[position() != 1]">
</xsl:template>
```

Abbildung 6. Beispiel für die Umsetzung der Anhänge

- Index-Einträge werden in DocBook durch das Tag `indexentry` markiert und werden in der Druckausgabe nicht dargestellt, sondern nur zur Erstellung des Index verwendet. Die "Processing Expectation" bei der verlagseigenen DTD ist jedoch, dass Index-Einträge sowohl im Text als auch im Index gedruckt werden.

  Der andere Weg, also Text zweimal (je einmal im Text und im Index) statt einmal zu erzeugen ist technisch machbar. Der vorliegende Fall würde verlangen, bisher nicht markierten Text zu ignorieren - was technisch nicht machbar ist. Für dieses Dokument habe ich mich dafür entschieden, die DocBook Styles so zu modifizieren, dass auch hier der markierte Begriff ausgegeben wird.

- Die Bibliographie in DocBook kann recht komplex sein. Die Pearson-DTD kennt selber kein Literaturverzeichnis, so dass in eine passende Darstellung konvertiert werden musste. Dabei habe ich das Markup im DocBook-Original so gestaltet, dass die einzelnen Einträge leicht konvertiert werden konnten. Auch wenn dabei nicht alle Features der DocBook-DTD verwendet wurden.

  In DocBook gibt es zwei Varianten, um ein Literaturverzeichnis zu erstellen. Ein Eintrag kann mit dem Tag `biblioentry` markiert werden. Dabei kümmern sich die Stylesheets um Interpunktion und die Verknüpfung mehrerer Autorennamen durch Kommata oder das Wort "und". Damit ist ein Eintrag einfach zu erfassen (sofern er den Erwartungen der Stylesheets entspricht), aber das Stylesheet ist relativ komplex. Die zweite Variante ist der Eintrag mittels `bibliomixed`. Hier kümmert sich der Autor um die passende Verknüpfung aller Teile eines Eintrags und das Stylesheet ist relativ einfach.

  Für die Konvertierung in die Pearson-DTD verwendete ich die zweite Variante.

- Bei Verweisen mit Hilfe des Tags `xref` generieren die DocBook-Styles automatisch die passenden Texte (Kapitel, Abschnitt, Abbildung etc.) zusätzlich zur referenzierten Nummer. Bei dem Prozessieren der verlagseigenen DTD existiert diese Processing Expectation nicht, so dass bei der Konvertierung die Texte eingefügr werden mussten. Den Abschnitt aus der Konvertierung finden Sie in Abbildung 7, „Text-Generierung für Querverweise“ [11].

```
<xsl:template match="xref"
              xmlns:xlink="http://www.w3.org/1999/xlink">
  <xsl:variable name="linkend" select="@linkend"/>
  <xsl:variable name="targets" select="id(@linkend)"/>
  <xsl:variable name="target" select="$targets[1]"/>
  <xsl:variable name="refelem" select="local-name($target)"/>

  <xsl:call-template name="check.id.unique">
    <xsl:with-param name="linkend" select="$linkend"/>
  </xsl:call-template>

  <xsl:call-template name="anchor"/>

  <xsl:choose>
    <xsl:when test="count($targets) = 0">
      <xsl:message>
        <xsl:text>XRef to nonexistent id: </xsl:text>
        <xsl:value-of select="@linkend"/>
      </xsl:message>
      <xsl:text>[??? </xsl:text>
      <xsl:value-of select="@linkend"/>
      <xsl:text>]</xsl:text>
    </xsl:when>
    <xsl:when test="count($targets) > 1">
        <xsl:message>Uh, multiple linkends
        <xsl:value-of select="@linkend"/>
      </xsl:message>
      </xsl:when>
  </xsl:choose>

  <xsl:choose>
    <xsl:when test='$refelem = "chapter"'>
      <xsl:text>Kapitel </xsl:text>
    </xsl:when>
    <xsl:when test='$refelem = "section"'>
      <xsl:text>Abschnitt </xsl:text>
    </xsl:when>
    <xsl:when test='$refelem = "figure"'>
      <xsl:text>Abbildung </xsl:text>
    </xsl:when>
    <xsl:when test='$refelem = "example"'>
      <!-- xsl:choose>
        <xsl:when test="$target/@role='listing'">
          <xsl:text>Listing </xsl:text>
        </xsl:when>
        <xsl:otherwise -->
          <xsl:text>Listing </xsl:text>
        <!-- /xsl:otherwise>
      </xsl:choose -->
    </xsl:when>
    <xsl:when test='$refelem = "table"'>
      <xsl:text>Tabelle </xsl:text>
    </xsl:when>
    <xsl:when test='$refelem = "appendix"'>
      <xsl:text>Anhang </xsl:text>
    </xsl:when>
    <xsl:otherwise>
```

```
      <xsl:message>
        <xsl:text>Cant't handle xref to $refelem:</xsl:text>
        <xsl:value-of select="@linkend"/>
      </xsl:message>
      <xsl:text>(??? $refelem)</xsl:text>
    </xsl:otherwise>
  </xsl:choose>

  <xref>
    <xsl:attribute name="xlink:href"
              xmlns:xlink="http://www.w3.org/1999/xlink">
       <xsl:value-of select="@linkend"/>
    </xsl:attribute>
  </xref>
</xsl:template>
```

Abbildung 7. Text-Generierung für Querverweise

Ich habe mich für DocBook als Quellformat entschieden, weil ich diese DTD schon länger kenne und verwende. In den Original LaTeX-Quellen wurde außerdem logische Markup verwendet, das einfach in DocBook konvertiert werden konnte. Bei einer Umwandlung in das Pearson-Format wäre diese logische Formatierung leider verloren gegangen, da diese DTD eine Mischung zwischen logischem und visuellem Markup abbildet.

Vor der endgültigen Manuskript-Abgabe muss man sicherstellen, dass das in das Pearson-Format umgewandelte Dokument dem Original im DocBook-Format entspricht. Probeausdrucke und Korrekturfahnen wurden daher mit einem eigenen Stylesheet - basierend auf den DocBook-Styles - aus dem konvertierten Dokument erstellt. Damit ließ sich die Qualität der Umsetzung, sowohl aus dem LaTeX-Format als auch der Schritt zwischen DocBook und Pearson gut prüfen.

Nach dem Druck lieferte die Druckerei die (manuell nachbearbeitete) XML-Datei, die zum Belichten verwendet wurde. Mit Hilfe von Tools wie **wdiff** oder **xmldiff** wurde diese Datei mit dem gelieferten Original verglichen. Das führte noch zu einigen kleinen Änderungen in den Konvertierungsstyles - für die nächste Auflage wird das automatisch korrekt sein und auch dem Setzer Arbeit sparen.

11. DocBook-Erweiterungen

DocBook-Erweiterungen bestehen in der Regel aus zwei Teilen. Ein Customization-Layer für die DTD, so dass neue, zusätzliche Tags und Attribute erlaubt werden oder nur ein Subset von DocBook erlaubt wird. Damit ist es möglich, Dokumente gegen die entstehende DTD zu validieren.

Der zweite Teil sind Stylesheets, die zum Erzeugen der verschiedenen Ausgabeformate dienen. Damit kann aus einem validen Dokument HTML oder ein anderes Format erzeugt werden. Nur in Kombination beider Teile ist eine Erweiterung in der Praxis sinnvoll.

11.1. Website

Viele Webseiten beschäftigen sich mit Computer-Themen. Was also liegt näher, als DocBook um solche

Tags zu ergänzen, die für Webseiten fehlen?

Dabei wird für jede Seite eine eigene XML-Datei erzeugt. Das Layout wird zentral definiert und in alle generierten HTML-Seiten eingefügt. Damit erhält man ein einheitliches Layout und einheitliche Bedienung bei der Verwendung von einfachem, statischem HTML.

Das Layout legt man in einer XML-Datei fest, aus dieser baut der XSL-Prozessor das Menü im linken Teil der HTML-Seite auf. Der Inhalt der rechten Seite wird aus der jeweils prozessierten XML-Datei gelesen. Damit ist es sehr einfach, eine Seite mit einem einheitlichen Layout und einer brauchbaren Navigation zu erstellen. Der XML-Prozessor mit den Stylesheets kümmern sich um die lästigen Details.

Mit einem einfachen `Makefile` kann die Erzeugung der HTML-Seiten der geänderten XML-Dateien automatisiert werden. Wenn man im selben `Makefile` zum Beispiel **rsync** aufruft, so kann man automatisch die publizierte Seite beim Provider ändern.

11.2. Slides

Einige Beispiele finden Sie auf meiner Webseite (http://www.jochen.org/~jochen/vortraege.html). Ein Nachteil ist, dass die Styles nur ein "naktes" Layout verwenden. Auch gibt es keine Features wie Animationen oder Farbverläufe. Diese sind in der DTD nicht vorgesehen.

Dennoch empfinde ich den Einsatz von DocBook/slides als sinnvoll, da ich die Tags kenne und damit recht schnell Präsentationen erstellen kann. Und diese praktisch überall darstellen kann, da nur HTML als Ausgabe verwendet wird.

In Umgebungen, wo viel Wert auf optische Darstellung und Gimmicks gelegt wird, ist man aber mit einem echten Präsentationsprogramm besser bedient, auch wenn ich begonnen habe, Styles für die Konvertierung zu Magicpoint und DFBpoint zu entwickeln.

12. Andere nützliche XML-Anwendungen

Neben dem Einsatz als Computer-Dokumentation kann XML auch noch für andere Zwecke verwendet werden. Hier sei als Beispiel nur XML-Resume genannt. Mit diesem Format können Sie Lebensläufe erfassen und in verschiedenen Formaten darstellen. Auch wäre es möglich, maschinell jeweils nur Teile des Lebenslaufes darzustellen, damit dieser nur die für die aktuelle Bewerbung notwendigen Teile enthält.

13. Ausblick

Für mich persönlich war der Einsatz von XML in einem größeren Projekt sehr nützlich. Die verfügbaren Tools erfüllten ohne Ausnahme alle ihren Zweck, wenn auch die Java-Tools viel Speicher und damit lange Laufzeiten benötigen.

Bisher frei verfügbare Tools für XML-FO reichen für interne Ausdrucke aus, aber nicht für die Abgabe von druckfertigen Manuskripten. Das Ziel, valide Dokumente gemäß der Pearson-DTD zu erzeugen, war jedoch mit nur geringem Aufwand zu erreichen - denn dafür war ein XML-FO-Prozessor nicht notwendig.

Damit konnte der eigentliche Satz durch einen professionellen Setzer erfolgen, der bereits Erfahrung mit der Verlags-DTD hatte. Basierend auf dem LaTeX-Manuskript hätte der Satz durch den Autor erfolgen müssen. Und in den ersten Auflagen wurde klar, dass LaTeX zwar meist gute Ergebnisse liefert, aber manche Dinge nicht einfach zu realisieren sind.

Durch die Verlagerung des Satzes vom Autor zu einem professionellen Setzer hat sich die Satz-Qualität verbessert und der Autor wurde entlastet. Durch die Verwendung einer Standard-DTD mit logischem Markup konnte beim Editieren das LaTeX-Format (mit Makros zum logischen Markup) einfach übernommen werden.

Für zukünftige Auflagen steht als Format wieder ein transparentes, gut wartbares Dokument zur Verfügung. Da Änderungen in einer Versionsverwaltung wie CVS oder Subversion gehalten werden lassen sich auch ältere Änderungen nachvollziehen. Auch ist die Arbeit auf mehreren Rechnern oder von mehreren Personen möglich - insbesondere weil jedes Kapitel in einer eigenen XML-Datei gespeichert ist..

Das hier vorgestelle Projekt erforderte intensive Kenntnisse in den verschiedenen Tools. Für DocBook existieren eine Reihe von Tools, die auch für Laien einsetzbar sind, manche dieser Tools sind aber kommerzielle.

Tausende Legacy-Dokumente?

Literaturverzeichnis

[DocBook TDG] DocBook The Definitive Guide Norm Walsh and Leonard Mueller O'Reilly & Associates, Inc. Oktober 1999 ISBN 156592-580-7
Das Buch ist online unter http://www.docbook.org/tdg/en/html/docbook.html verfügbar.

[DocBook XML] DocBook XSL: The Complete Guide Bob Stayton Sagehill Enterprises September 2003 ISBN 0974152110
Das Buch ist online unter http://www.sagehill.net/docbookxsl/index.html verfügbar.

DocBook Frequently Asked Questions
http://www.dpawson.co.uk/docbook/

DocBook Wiki
http://www.docbook.org/wiki/moin.cgi/FrontPage

DocBook Standard
http://www.oasis-open.org/docbook

DocBook Entwicklung und Dokumentation
http://www.docbook.org

DocBook Projekt bei Sourceforge
http://docbook.sf.net

[lit-docbook-tutorial] DocBook-Tutorial
http://trieloff.net/docbook/tutorial/

DocBook-Filter für OpenOffice
http://xml.openoffice.org/xmerge/docbook/