Erstellung eines XSLT Style Sheets zur Transformation der XML-Dokumente des Projekts Camena in HTML-Dokumente

Studienarbeit

am

Rechenzentrum der Universität Mannheim

Betreuer:

Prof. Dr. rer. nat. Hans Werner Meuer
Dr. rer. nat. Heinz Kredel

im

Sommersemester 2001

von

Matthias Robert Grünewald
R7, 25
68161 Mannheim
Studiengang Wirtschaftsinformatik

Inhaltsverzeichnis:

Abbildungsverzeichnis:

Listings:

1. Motivation

1.1. Das Projekt CAMENA

Das Projekt CAMENA erfasst ausgewählte Texte der neulateinischen Dichtung aus Deutschland, die zu den historischen Buchbeständen der Universitätsbibliothek Mannheim gehören.
Sinn und Zweck des CAMENA Projektes ist es, die Vielzahl an neulateinischen Texten zu erschließen und in einer repräsentativen Auswahl zu edieren, so dass Forscher der neulateinische Philologie gezielter nach Werken und deren Inhalten im Netz suchen können.
Außerdem soll die Bereitstellung der Werke im Netz die alten Editionen vor Beanspruchung schützen und trotzdem Forschern weltweit die Möglichkeit geben, ohne langwierige Entleihvorgänge mit den Abfassungen zu arbeiten.
Leider können die abfotografierten Originale, selbst in Verbindung mit generierten Inhaltsverzeichnissen, für einen Suchenden im Netz den substantiellen Gehalt der Werke nur unzureichend wiedergeben. Daher wurden alle Texte maschinenlesbar mit XML erfasst, um eine spätere Auswertung der Daten nach inhaltlichen, semantischen Gesichtspunkten vornehmen zu können[1].

Das Ziel der Studienarbeit ist es nun, für die bisher erfassten Volltexte der Autoren Bersmann, Fürstenberg, Gesner und Opitz, die als XML-Dokument im ASCII-Format vorliegen, ein XSLT-Style-Sheet zu erstellen. Dieses XSLT-Style-Sheet soll für alle XML-Dokumente der Autoren des Projekt CAMENA verwendbar sein.

[1] Vgl. Universitätsbibliothek Mannheim / Rechenzentrum der Universität Mannheim 2001, Seite: Camena.html.

Mit Hilfe der XSLT-Definitionen im Style-Sheet soll dann für die XML-Dokumente jeweils eine Version in HTML-Format konvertiert werden.
Die Konvertierung geschieht in Richtung HTML, da HTML-Browser die weiteste Verbreitung haben und somit HTML-Formate von vielen Werkzeugen und praktisch auf allen Rechnerplattformen unterstützt werden[2].

2. Kurzeinführung

2.1. HTML

HTML ist die Abkürzung für Hypertext Markup Language. Die Idee von HTML war es, wissenschaftliche Dokumente online sichtbar zu machen. Dabei sollte es möglich sein, einfache Grafiken einzubinden und Texte zu formatieren[3].
Dazu nötig waren ein neues Dateiformat, nämlich HTML, und ein neues Internet-Protokoll, das Hypertext Transfer Protocol (HTTP) um das HTML-Dateiformat zu transportieren[4].
HTML wurde in den Jahren 1989/1990 am europäischen Hochenergieforschungszentrum CERN bei Genf entwickelt[5]. Die Entwickler stellten damals die wesentlichen Elemente für eine graphische Bedienung des Internet vor, das Hypertextprinzip, bei dem sich der Benutzer durch die Informationen klicken kann[6] und somit nicht mehr auf die Kommandozeileneingabe angewiesen ist. Die Informationen werden dadurch modulhaft, nichtlinear angeordnet, so dass sich eine Art Setzkasten ergibt, der immer wieder neue und auf die Bedürfnisse des Anwenders spezifisch

[2] Vgl. Kobert 1999, Seite: 53.
[3] Vgl. Münz 1998, Seite: tbad.htm.
[4] Vgl. ebenda, Seite: tbad.htm
[5] Vgl. Seeborg-Weichselbaum 2000, Seite 20.
[6] Vgl. ebenda, Seite: 20.

zugeschnittene Dokumentkombinationen zulässt[7]. Abbildung 1 zeigt das Hypertextprinzip: Hyperlinks aus Dokument 2 auf Server 1 können sowohl auf Stellen im Dokument 2 selbst, als auch auf Stellen eines anderen Dokuments (Dokument 1) auf dem selben Server und sogar auf Dokumente auf anderen Servern (Dokument 1 auf Server 2) verweisen. Je nachdem welche Dokumente der Anwender in Zusammenhang setzen will, kann er die Links beliebig setzen[8].

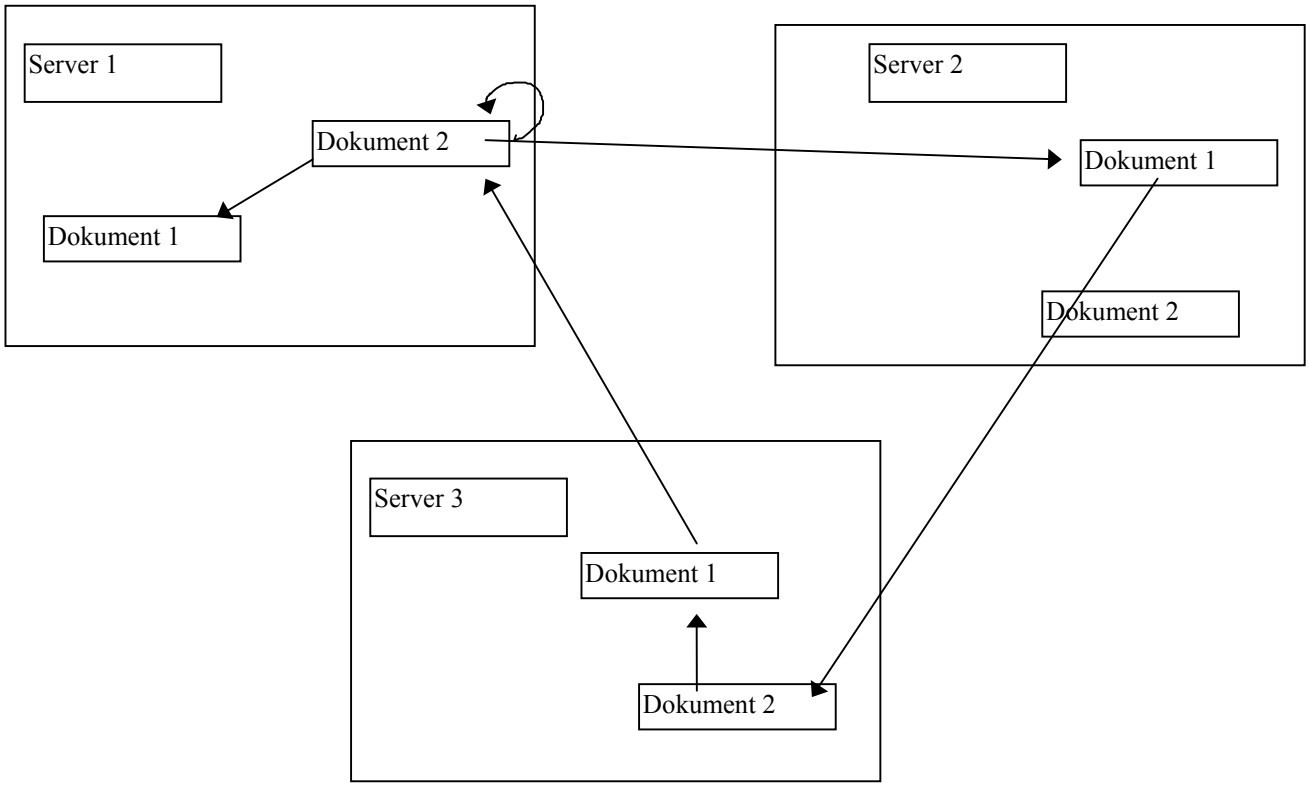

Abbildung 1: Das Hypertextprinzip[9]

HTML ist eine Markup-Sprache auf SGML-Basis und wird deshalb als SGML-Anwendung bezeichnet (auf die Metasprache SGML wird im Kapitel XML noch

[7] Vgl. Bliemel et al 1999, Seite 3f.
[8] Vgl. Jones 1995, Seite 5ff.
[9] Vgl. ebenda, Seite 6.

näher eingegangen). Durch das sogenannte Markup werden einer in einem Dokument vorhandenen Information zusätzlich Zeichen hinzugefügt, die dazu genutzt werden, um diese Information auf eine ganz bestimmte Art und Weise zu verarbeiten bzw. darzustellen[10]. So wird also mit Hilfe dieser Markups eine ganze Reihe von Befehlen und deren Syntax festgelegt, derer man sich zur Entwicklung von Web-Sites bedienen kann[11].

Trotz ständiger Weiterentwicklung in den letzten Jahren hat HTML auch in der aktuellen Version 4.0 nur einen begrenzte Anzahl von ca. sechzig Befehlen[12]. Einerseits erleichtert dies für Anfänger den Einstieg in diese Sprache, andererseits schränkt dies aber auch die Nutzung für Anwendungen ein[13], so dass besondere Eigenschaften von Dokumenten und deren Anforderungen nur durch zusätzliche Technologien verwirklicht werden können[14].

Eine der größten Schwächen von HTML, nämlich die Vermischung von Dokumentinhalten und Darstellungsinformationen, versucht man, durch den Einsatz von Cascading Style Sheets zu kompensieren. Diese sind eine unmittelbare Ergänzung zu HTML und dienen der Bestimmung der Eigenschaften des Formats eines HTML-Dokuments.

Damit geht die Idee der Style-Sheet-Programmierung für HTML auf das Prinzip des Textverarbeitungsprogramm TeX bzw. LaTeX zurück, nämlich Text und Stilvorlage zu trennen[15].

Die Gestaltungsmöglichkeiten, die einem mit Cascading Style Sheets eröffnen, übersteigen bei weitem die der Attribute von HTML, von denen in absehbarer Zukunft ohnehin einige nicht mehr zum gültigen Standard des W3C gehören werden. Durch die Verwendung von Cascading Style Sheets kann zudem der benötigte Speicherplatz für

[10] Vgl. North/Hermans 1999, Seite kap02.htm.
[11] Vgl. Pott/Wielage 2000, Seite 37.
[12] Vgl. ebenda, Seite 37.
[13] Vgl. ebenda, Seite 37.
[14] Vgl. Michel 1999, Seite27.
[15] Vgl. Harms et al. 2000, Seite 20.

HTML-Dateien klein gehalten und somit die Übertragungszeiten zum Anwender reduziert werden[16].

Trotz der angegebenen Schwächen wird dennoch auf HTML zurückgegriffen. Dies liegt einerseits an den schon oben genannten Vorteilen der Unterstützung auf nahezu allen Rechnerplattformen, andererseits aber auch daran, dass XML noch nicht in genügendem Maße von den Browsern der Nutzer angezeigt werden kann.

2.2. XML

Da sich in der Vergangenheit abzeichnete, dass HTML den Anforderungen an eine moderne, professionelle Webseitengestaltung nicht mehr gerecht wurde[17], begann im Jahre 1996 das W3-Konsortium, an der Extensible Markup Language XML zu arbeiten. Anfang des Jahres 1998 wurde XML zum offiziellen Standard des W3-Konsortiums erhoben[18].

XML ist eine Generalized Markup Sprache, also eine verallgemeinerte Auszeichnungssprache. Dies ermöglicht es dem Entwickler, eigene neue „Web-Sprachen“ definieren zu können, die seinen spezifischen Anforderungen entsprechen[19]. Sprachen, mit denen man Auszeichnungssprachen selbst definieren kann, nennt man auch Metasprachen. Der Zusammenhang zwischen Metasprache und Auszeichnungssprache ist in Abbildung 2 auf der nächsten Seite dargestellt.

Eine Metasprache, z.B. XML oder SGML, definiert eine Auszeichnungssprache, z.B. HTML oder das XML-Pendant XHTML. Auszeichnungssprachen sind somit SGML- bzw. XML-Applikationen. Mit der Auszeichnungssprache können dann die jeweiligen Web-Dokumente erstellt werden.

---

[16] Vgl. Harms et al. 2000, Seite 227.
[17] Vgl. Kobert 1999, Seite 50.
[18] Vgl. Seeboerger-Weichselbaum 2000, Seite 20.
[19] Vgl. Pott/Wielage 2000, Seite 47.

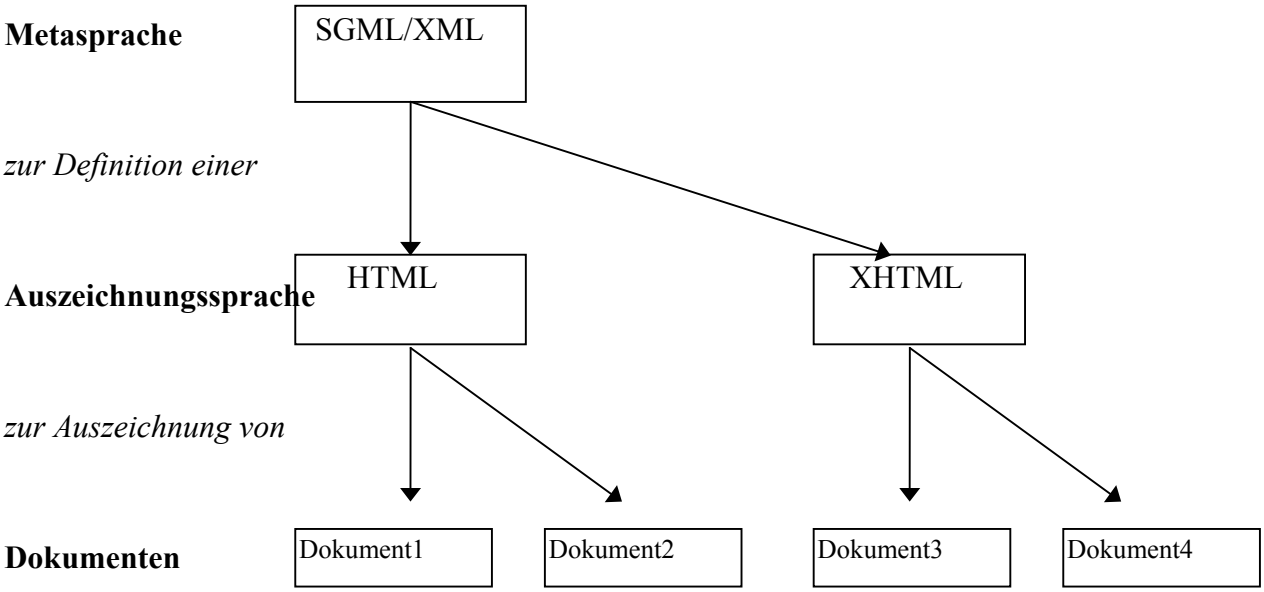

Abbildung 2: Das Prinzip der Metasprachen[20]

Diese Fähigkeit stellt XML auf eine Stufe mit SGML, der Standard Generalized Markup Language. Der 1986 als ISO 8879 verabschiedete Standard SGML ist sozusagen „die Mutter aller Auszeichnungssprachen im Web“[21], denn als Metasprache stellt SGML die Basis für Auszeichnungssprachen wie z.B. HTLM dar, die mit Hilfe von SGML erzeugt bzw. definiert werden. XML ist als Teilmenge von SGML zu verstehen, denn alle Optionen von XML sind bereits in SGML enthalten. Somit ist jedes gültige XML-Dokument auch gleichzeitig ein gültiges SGML-Dokument[22]. Daher spricht man im Zusammenhang mit XML nicht von einer SGML-Anwendung, sondern von einem SGML-Profil[23].

Die wichtigsten Bestandteile eines XML-Dokuments sind wie bei SGML-Dokumenten auch die XML-Auszeichnungen und die Dokumenttyp Definition[24]. Die XML-Ausziechnungen beschreiben welcher Zeichensatz in den Dokumenten auftritt.

[20] Vgl. Michel 1999, Seite 28.
[21] Vgl. Pott/Wielage 2000, Seite 44.
[22] Vgl. Michel, 1999, Seite 30.
[23] Vgl. Pott/Wielage 2000, Seite 47.

Die ausgezeichneten Daten in den Tags und die Dokumenttyp Definition beschreiben also ein Strukturmodell für den Inhalt eines XML-Dokuments. Dieses Modell besagt, welche Sprachelemente zwingend vorhanden sein müssen und welche optional sind. Es beschreibt weiterhin, welche Attribute diese Sprachelemente haben und wie ihre Beziehungen untereinander strukturiert werden können[25].

2.2.1. Struktur der XML-Auszeichnungen

XML verwendet seine Anfangs- und Ende-Tags als eine Art Container. Ein einzelnes Element setzt sich zusammen aus dem Anfangs-Tag, dem Inhalt und dem Ende-Tag. Der Inhalt eines solchen Elementes kann sich wieder aus einem oder mehreren anderen Elementen zusammensetzen, so dass Elemente rekursiv ineinander verschachtelt sein können[26]. Elemente sind also die Bausteine, aus denen sich ein XML-Dokument zusammensetzt. Dabei muss jedes XML-Dokument ein Wurzelelement haben, welches das gesamte Dokument einnimmt. Alle anderen Elemente müssen vollständig in dieses Wurzelelement verschachtelt sein.
Das bedeutet, dass kein anderes Element vor dem Wurzelelement beginnen oder nach dem Wurzelelement enden darf. Dieses Verschachtelungsprinzip gilt, wie schon anhand des Containerprinzips erläutert, auch für alle anderen Elemente innerhalb des Wurzelelements. Wenn ein Element andere Elemente enthält, so müssen diese anderen Elemente vollständig von diesem Element eingeschlossen sein. Abbildung 3 erläutert das Container-Prinzip.

---

[24] Vgl. Michel 1999, Seite 29.
[25] Vgl. ebenda, Seite 29.
[26] Vgl. Pott/Wielage 2000, Seite 112.

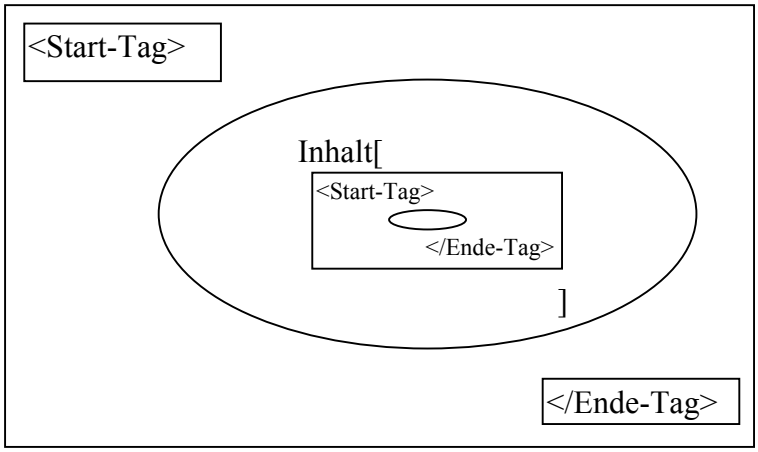

Abbildung 3: Das Container-Prinzip[27]

In Abbildung 4 ist die Struktur eines XML-Dokuments in der Baumnotation anhand einiger Elemente im XML-Dokument Bersmann.xml skizziert. Dabei kann man die Verschachtelungsregeln anhand eines konkreten Beispiels nachvollziehen.

Wie man sehen kann, sind die Tags des Dokuments baumähnlich strukturiert, wobei sich das Wurzelelement <text> ganz oben im Baum befindet. Da ein XML-Dokument nur ein einziges Root-Element haben darf, befindet sich kein weiteres Element in der gleichen Hierarchiestufe wie das <text>-Element.

Alle Elemente innerhalb dieses Elements, also <front>, <body> und <back>, sind vollständig in das <text>-Element verschachtelt. Da dieses Verschachtelungsprinzip auch für alle weiteren Hierarchiestufen gilt, gibt es keine Elemente unterer Hierarchiestufen, die teilweise vor oder hinter einem Element liegen, von dem sie eigentlich eingeschlossen sein sollten.

[27] Vgl. Pott/Wielage 2000, Seite 112.

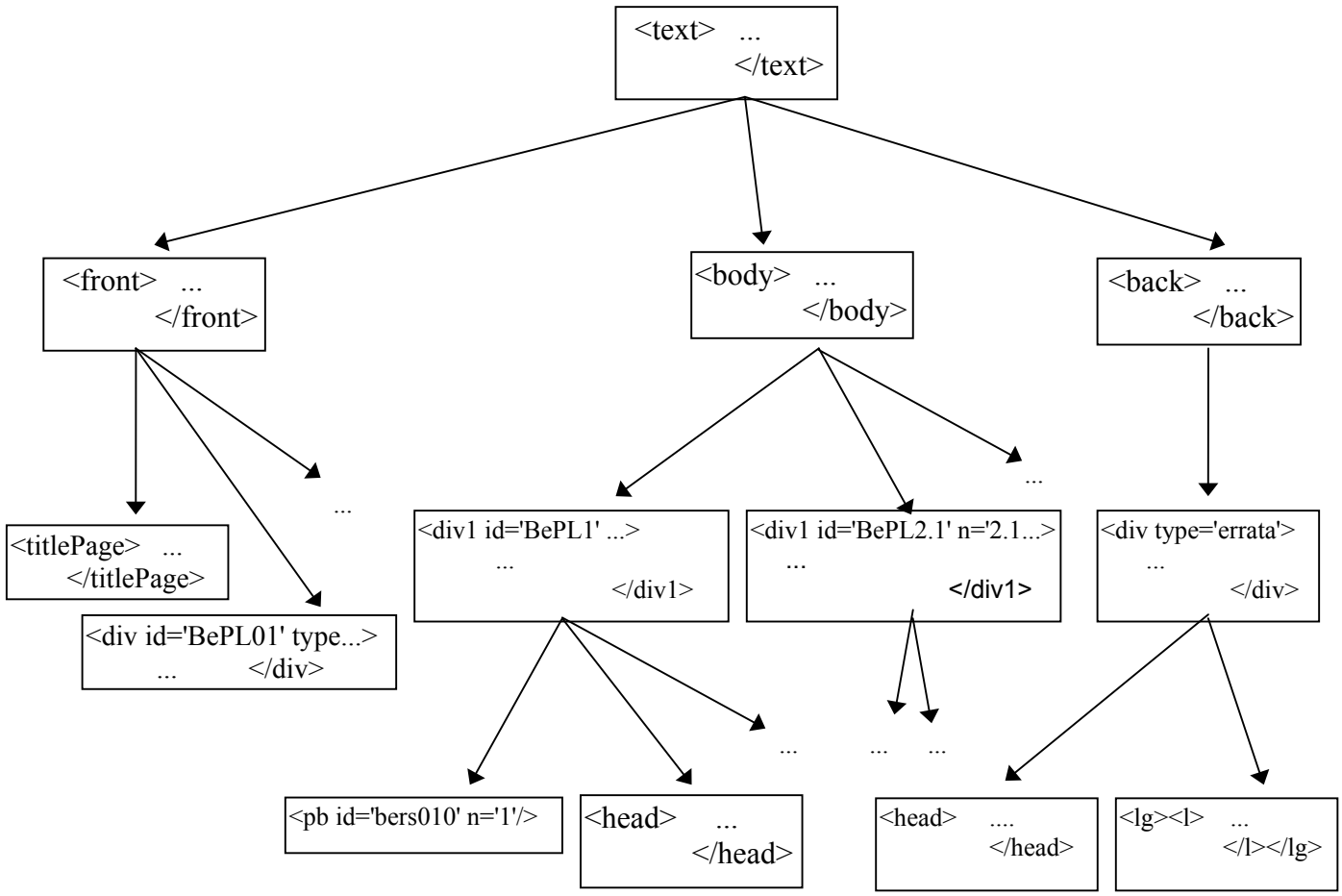

Abbildung 4: Baumstruktur anhand einiger Elemente im XML-Dokument Bersmann.xml

2.2.2. Dokumenttyp Definition

Die Dokumenttyp Definition (DTD) gibt die Regeln vor, wie ein Dokument in XML korrekt auszuzeichnen ist. Die Struktur der Dokumenttyp Definition ist ebenso geschachtelt wie die XML Auszeichnungen.

Elemente müssen vollständig ineinander verschachtelt sein und dürfen nicht verzahnt werden. Mithilfe der DTD läßt sich somit in XML festlegen, welche Inhaltstypen es in

Dokumenten eines bestimmten Typs geben kann oder muss und in welcher Beziehung diese Inhaltstypen stehen können oder müssen.

Solch eine Struktur kann ebenfalls als Baumstruktur dargestellt werden. Abbildung 4 zeigt die Darstellung in Baumnotation anhand der Definition des <text>-Elements in der TEIXLITE.DTD, welche in Listing 1 zu sehen ist.

Die Syntax erlaubt hier fünf Operatoren: die beiden Konnektoren ‘ , ‘ und ‘ | ‘ und die Häufigkeitsindikatoren ‘ ? ‘ , ‘ + ‘ und ‘ * ‘. Der ,-Operator wird für die Konkatenationsoperation verwendet. Wenn zum Beispiel a und b Elemente sind, dann muss bei einer Operation a,b das Element b auf Element a folgen. Das Operatorzeichen ‘ | ‘ entspricht einer Oder-Verknüpfung. Soll beispielsweise klar werden, dass entweder das Element a oder das Element b in der Instanz stehen soll, dann wird dies durch den Ausdruck a|b dargestellt. Die Häufigkeitsindikatoren sind Postfix-Operatoren mit jeweils nur einem Operanden. Dabei repräsentiert der ?-Operator das optionale Element, welches nur einmal oder gar nicht vorkommen kann, der +-Operator steht für ein erforderliches und wiederholbares Element, welches mindestens einmal vorkommen muss und der *-Operator zeigt an, dass ein optionales und wiederholbares Element vorliegt, also ein Element, das beliebig oft oder gar nicht vorkommen kann[28].

```
<!ELEMENT text      (
      (index | interp | interpGrp | lb | milestone | pb | gap | anchor)* ,
      (front, (index | interp | interpGrp | lb | milestone | pb | gap | anchor)*)?,
      (body | group),
      (index | interp | interpGrp | lb | milestone | pb | gap | anchor)*,
      (back, (index | interp | interpGrp | lb | milestone | pb | gap | anchor)*)?) >
```

Listing 1: Definition des Elements <text> in der teixlite.dtd

[28] Vgl. Michel 1999, Seite 57f.

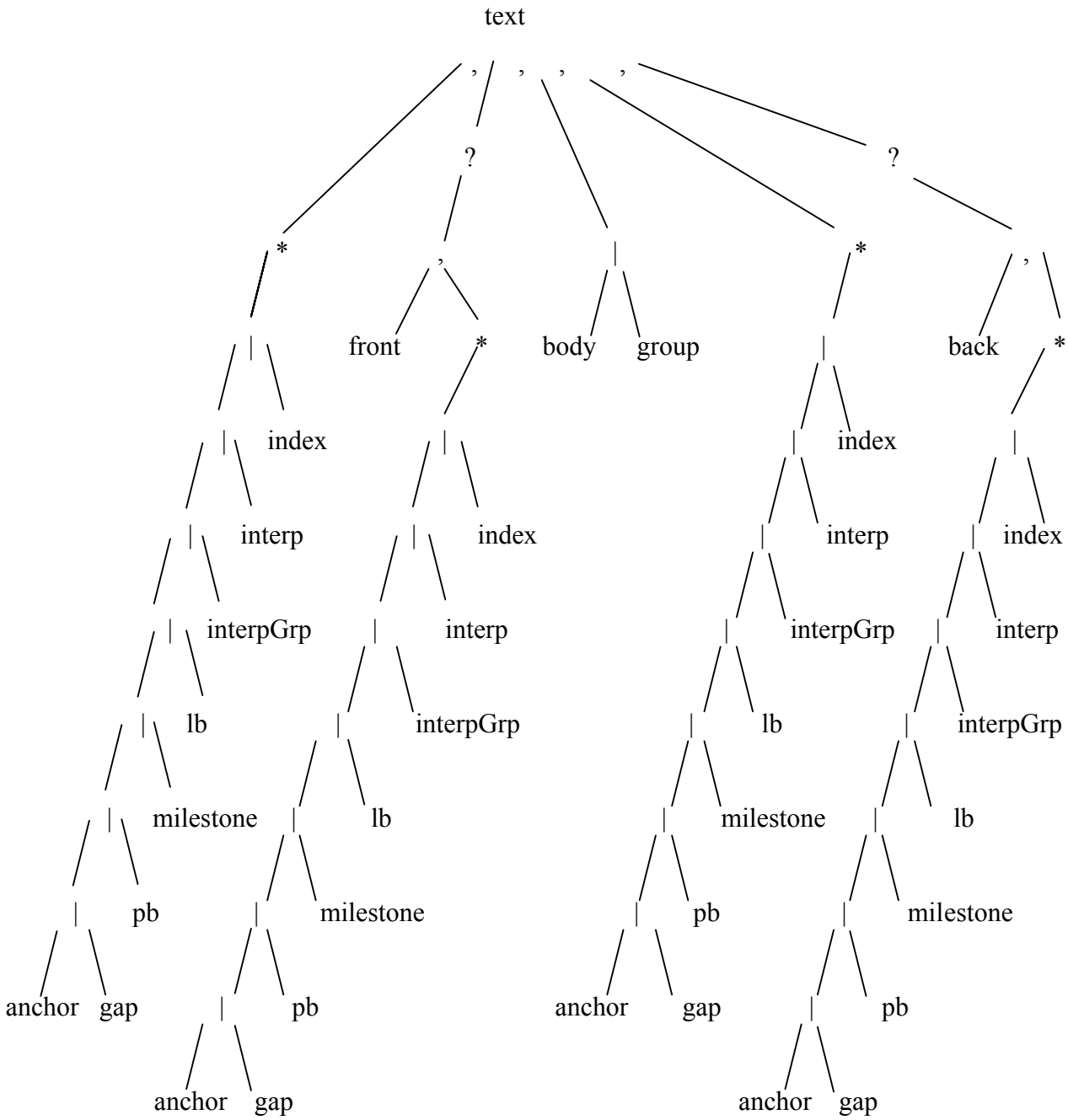

Abbildung 5: Baumstruktur anhand des Elements <text> in der teixlite.dtd

2.2.3. Der Zusammenhang zwischen XML-Dokument und DTD

Man kann bei XML (wie auch bei SGML) zwei Typen von Regeln ausmachen: die Deklarationsregeln der DTD zur Deklaration von Auszeichnungssprachen und die Auszeichnungsregeln zur jeweils konkreten Auszeichnung eines XML-Dokuments anhand der vorliegenden Auszeichnungssprache[29].

Ist ein XML-Dokument gemäß den Regeln in seiner DTD korrekt ausgezeichnet, so spricht man von einem gültigen bzw. von einem validen Dokument[30]. Ein valides XML-Dokument wird auch als Instanz des jeweiligen Dokumenttyps bezeichnet[31]. Existiert keine DTD, so muss die Instanz lediglich entsprechend der Auszeichnungsregeln von XML wohlgeformt sein, was eine korrekte Klammerung bezüglich der Verschachtelung der einzelnen Elemente voraussetzt[32], nur ein einziges Wurzelelement erlaubt und zusätzlich zur Beendigung jedes Tags noch Kontextsensitivität verlangt[33].

Ein XML-Dokument kann man so als ein Element einer Klasse typengleicher Dokumente auffassen, die alle anhand einer DTD validiert werden können[34].

Das Zusammmenspiel von DTD und Dokument-Instanz ist in Abbildung 6 graphisch dargestellt.

Gemäß den Deklarationsregeln wird zuerst eine DTD deklariert und dann anhand der Auszeichnungsregeln ein XML-Dokument erstellt. Ein bezüglich der DTD valides XML-Dokument wird so, in einem weiteren Schritt, zu einer Instanz des von der DTD deklarierten Dokumenttyps.

---

[29] Vgl. Michel 1999, Seite 29.
[30] Vgl. ebenda, Seite 36.
[31] Vgl. ebenda, Seite 36.
[32] Vgl. Tolksdorf 2000, Seite 178.
[33] Vgl. Eberhart/Fischer 2000,Seite 259.
[34] Vgl. Michel 1999, Seite 36.

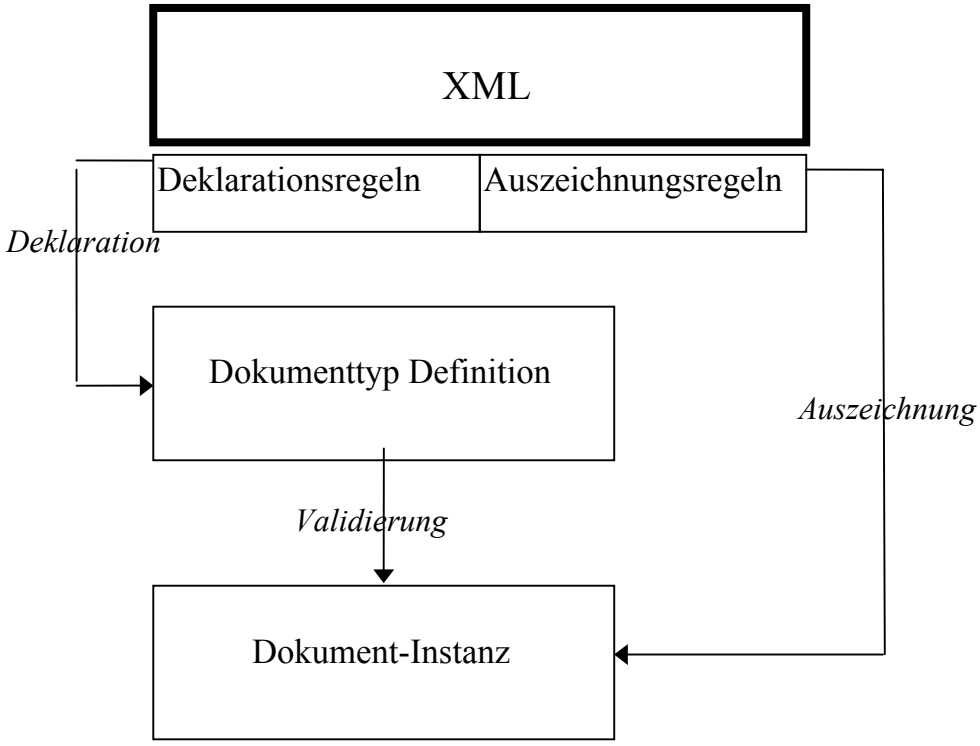

Abbildung 6: Vorgang der Validierung[35]

2.3. XSLT

2.3.1. Darstellung von XML

Die oben genannten Vorteile von XML als Metasprache, nämlich die frei definierbaren Auszeichnungselemente verkehren sich bei der Darstellung von XML-Dokumenten in einen Nachteil. Während im Falle von HTML alle heutzutage eingesetzten Browser in der Lage sind, sämtliche Tags zu erkennen und entsprechend darzustellen, sind

[35] Vgl. Michel 1999, Seite 37.

XML-Dokumente für solche Browser meist nicht darstellbar[36]. Die Anzahl der HTML-Tags und deren Bezeichnungen sind allen Browsern bekannt. Somit wird das Tag <p> immer als Absatz, das Tag <i> immer als Kursivschrift und das Tag <h1> immer als größte Überschrift angezeigt. Der semantischen Auszeichnung wegen ist dies aber bei XML anders geregelt.

Jeder Anwender kann sich seine eigene Markupsprache per DTD selbst definieren. Dadurch ist es dem Browser unmöglich zu wissen, wie beispielsweise ein Element <Werk> darzustellen ist. Im Zusammenhang mit dem Camena-Projekt ist zwar klar, dass es sich um eine der neulateinischen Dichtungen handelt; anhand der DTD ist entsprechend der Syntax ersichtlich, dass das Element <Werk> aus den Elementen Name, First, Last, URL und Type besteht und dass ein Element <Werk> mindestens einmal und beliebig oft innerhalb des <Camena> Elements vorkommen kann (siehe hierzu Listing 2). Es ist damit jedoch noch nichts über die physische Auszeichnung, also die Art der Darstellung des Elements, gesagt.

```
<!ELEMENT Camena (Werk)+>
<!ELEMENT Werk (Name, Erstes_Bild, Letztes_Bild, Url, Typ)>
```

Listing 2: Deklaration der Elements <Werk>

Eine Möglichkeit zur Lösung dieses Problems ist es, dem Browser mitzuteilen, wie er die einzelnen Elemente darzustellen hat. Dazu dienen die XSL-Style-Sheet Definitionen. XSL-Style-Sheets sollen die Inhalte von XML-Dokumenten sichtbar machen; damit erfüllen sie für XML denselben Zweck wie die Cascading Style Sheets für HTML-Dateien.

XSL ist eine auf XML abgestimmte Gestaltungssprache, die in zwei voneinander unabhängige Aufgabenkomplexe, XSLT und XSL:FO (Formatting Objects), unterteilt werden kann[37]. Diese beiden Komponenten werden im folgenden kurz erläutert:

[36] Vgl. Knobloch/Kopp 2001, Seite 49.
[37] Vgl. Born 2001, Seite 238.

- XSLT ist die Transformationssprache für XML, mit deren Hilfe XML-Dokumente in beliebige Ausgangsdokumente überführt werden können. Es ist eine Markup-Sprache, die auf XML basiert. Somit liegen XSLT-Style-Sheets als wohlgeformte Dokumente vor, die der XSLT-Prozessor benutzt, um XML-Quelldokumente zu konvertieren. Eine wichtige Subkomponente von XSLT ist XPATH; dies ist eine Notation zur Beschreibung von Pfaden innerhalb des Quelldokuments, die zu bestimmten Elementen oder auch Dokumentfragmenten führen.
- Formatting Objects sind Formatdefinitionen für Dokumentauszeichnungen, die einen Formatierungsprozess steuern sollen.

Da die XML-Dokumente des Camena Projekts in HTML-Dokumente transformiert werden sollen, ist die XSLT-Komponente der relevante Teil für das Ziel dieser Arbeit. Genau genommen findet eine Transformation in XHTML statt, denn HTML wurde zu Beginn des Jahres 2000 als XML-Anwendung XHTML definiert. Da XHTML aber lediglich HTML 4.0 in XML-konformer Notation ist, werden HTML und XHTML im folgenden als gleichbedeutende Ausdrücke verwendet.

Nicht nur weil HTML-Browser die am weitesten verbreiteten Browser in der „Web-Welt“ sind, geschieht eine solche Konvertierung in das HTML-Format, obwohl auch PDF und RTF möglich wären[38], sondern hauptsächlich weil eine weitere umständliche Verarbeitung durch XSL:FO und Formatierer, die ohnehin kaum zur Verfügung stehen, eingespart werden kann.

Eine Transformation, bei der XSL:FO-Steueranweisungen benutzt werden, ist deshalb nicht notwendig, weil das XHTML-Kovertierungsergebnis von gängigen Browsern direkt angezeigt werden kann.

Somit erhält ein HTML-Dokument bei einem Konvertierungsprozess, wie er in Abbildung 7 veranschaulicht wird, seine Inhalte aus einem XML-Dokument und die Anweisungen über seine Darstellung aus einem XSLT-Style-Sheet.

[38] Vgl. Pott/Wielage 2000, Seite 233.

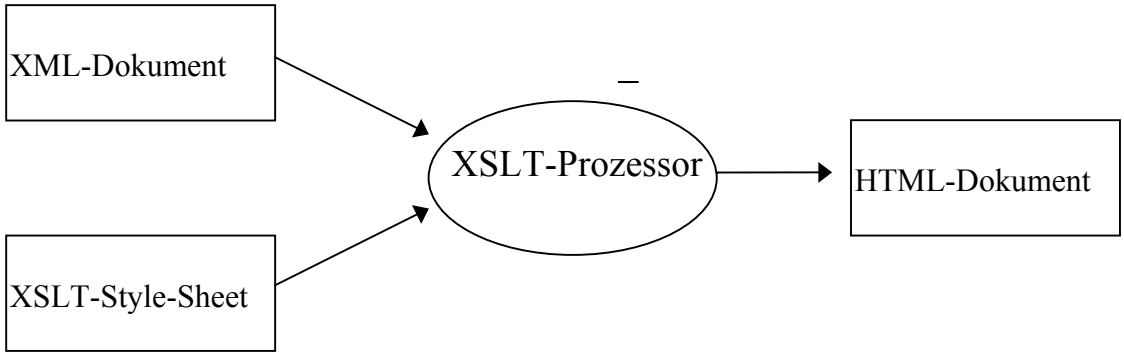

Abbildung 7: Konvertierungsprozess

Diese Trennung von semantischer und physischer Auszeichnung in XML-Dokument und XSLT-Style-Sheet ist eine Grundintention von XML[39]. Das Style-Sheet enthält eine Art Gestaltungsplan für das XML-Dokument, wobei die XSLT-Anweisungen in Blöcken, sogenannten Schablonen (bzw. Templates), zusammengefasst werden. Diese enthalten die Regeln, die den einzelnen Tags des XML-Dokuments die jeweiligen Verarbeitungsmuster zuordnen[40]. Bei der Erstellung des Style-Sheets ist also die Kenntnis der Struktur des zu transformierenden XML-Dokuments unbedingt erforderlich.

Die eben schon erwähnten Schablonen oder auch Templates sind ein zentrales Verarbeitungskonzept von XSLT und werden daher kurz erläutert, um das Verständnis der nachfolgenden Abschnitte zu erleichtern.

Eine Template-Anweisung hat folgendes Aussehen: <xsl:template match="/"> . Das Attribut "match" einer Template-Anweisung legt fest, für welchen Knoten bzw. welche Knotenmenge des XML-Dokuments die Verarbeitungsanweisungen, die sich innerhalb des Schablonenrumpfes befinden, angewandt werden sollen[41].

Im obigen Beispiel wird das Wurzelelement, also der Root-Knoten "/", verarbeitet. Die Anweisung <xsl:apply-templates/> innerhalb des Schablonenrumpfes stößt eine Verarbeitung aller Knoten an, die sich in der Hierarchieebene des XML-Dokuments

[39] Vgl. Eberhart/Fischer 2000, Seite 254.
[40] Vgl. Knobloch/Kopp 2001, Seite 105.
[41] Vgl. ebenda, Seite 111.

unterhalb des aktuellen Knotens befinden und für die ebenfalls ein Template angelegt wurde[42]. Besitzen diese Knoten ebenfalls eine <xsl:apply-templates/>-Anweisung, so stoßen auch sie eine Verarbeitung ihrer Kindknoten an.
Ist die Verarbeitung innerhalb eines Templates beendet, so springt der Prozess wieder zurück zur aufrufenden Stelle, an der die Rekursion begann, und fährt mit der seriellen Verarbeitung fort[43].
So kann man sich die Verarbeitungsabfolge innerhalb eines XSLT-Style-Sheets als eine Kette rekursiver Aufrufe vorstellen, die sich in den Hierarchieebenen des XML-Dokuments gemäß der oben beschriebenen Baumstruktur vom Wurzelelement auf einer Seite zu den Blattknoten durcharbeitet und auf der anderen Seite wieder zum Wurzelelement zurückkehrt.

3. Implementierung

3.1. Verwendete Werkzeuge

Die XSLT-Dateien wurden mit dem Mircosoft Editor für Windows 95/NT erstellt. Da die XSLT-Funktionalitäten Aufgabe des XSLT-Prozessors sind, sind die Anforderungen an den verwendeten Editor gering, so dass jeder beliebige Texteditor oder XML-Editor verwendet werden kann.
Die Transformation der betreffenden XML-Dokumente anhand der XSLT-Style Sheets wurde von einem XSLT-Prozessor vorgenommen. Dazu wurde der Xalan-J der Apache Software Foundation unter SuSE Linux 7.1 verwendet, der die W3C-Recommendations für XSL Transformations (XSLT) und XML Path Language (XPATH) implementiert[44]. Dieser in Java geschriebene XSLT-Prozessor kann auf

[42] Vgl. Knobloch/Kopp 2001, Seite 113.
[43] Vgl. ebenda, Seite 113.
[44] Vgl. The Apache Software Foundation 2001, Seite index.html.

Kommandozeilenebene als Applet oder Servlet oder auch als Modul eines anderen Programms verwendet werden[45]. Die für diese Studienarbeit verwendete Version wird auf Befehlszeile mit dem Schlüsselwort „transform“ aufgerufen und verlangt als Eingabe für eine Transformation zuerst das betreffende XML-Dokument und dann, von Leerzeichen getrennt, das XSLT-Style Sheet. Der Xalan-J gibt das Ergebnis im Fenster der Befehlszeileneingabe aus. Daher muss eine Ausgabeumleitung in eine Datei vorgenommen werden, um sich die Ergebnisdatei bzw. ihre Darstellung im Browser ansehen zu können. Dies geschieht, indem von einem weiteren Leerzeichen getrennt die Ergebnisdatei angegeben wird.
Als Browser zur Darstellung der HTML-Dateien wurde schließlich der Netscape Communicator in der Version 4.51 verwendet.

3.2. Die Vorgehensweise

Wie schon weiter oben beschrieben, sollen die als XML-Dokumente im ASCII-Format vorliegenden literarischen Werke der Autoren Bersmann, Fürstenberg, Gesner und Opitz mit Hilfe von XSLT-Transformationsskripten in Dokumente im HTML-Format konvertiert werden. Da hier ein Style-Sheet für mehrere XML-Dokumente zum Einsatz kommt und anzunehmen ist, dass im Laufe des Camena Projekts die Anzahl der XML-Dokumente noch stark anwachsen wird, soll die Konvertierung der Quelldaten in HTML-Format in zwei Arbeitsschritte aufgeteilt werden, um so die Konfigurierbarkeit des Programms zu verbessern.
Im ersten Schritt soll eine Art Datenbank erstellt werden, in der alle Autoren des Camena-Projekts, deren Werke bereits als Volltexte in einer XML-Version vorliegen, aufgenommen werden sollen. Diese kleine Datenbank ist die CamenaDB.

[45] Vgl. The Apache Software Foundation 2001, Seite index.html.

In einem zweiten Schritt wird ein XSLT-Transformationsskript erstellt, welches die CamenaDB ausliest und ein Zwischendokument generiert. Dieses Zwischendokument ist dann das eigentliche Style-Sheet, welches das jeweilige XML-Dokument in ein HTML-Format konvertiert. In Abbildung 8 wird der gesamte Transformationsprozess noch einmal anschaulich dargestellt.

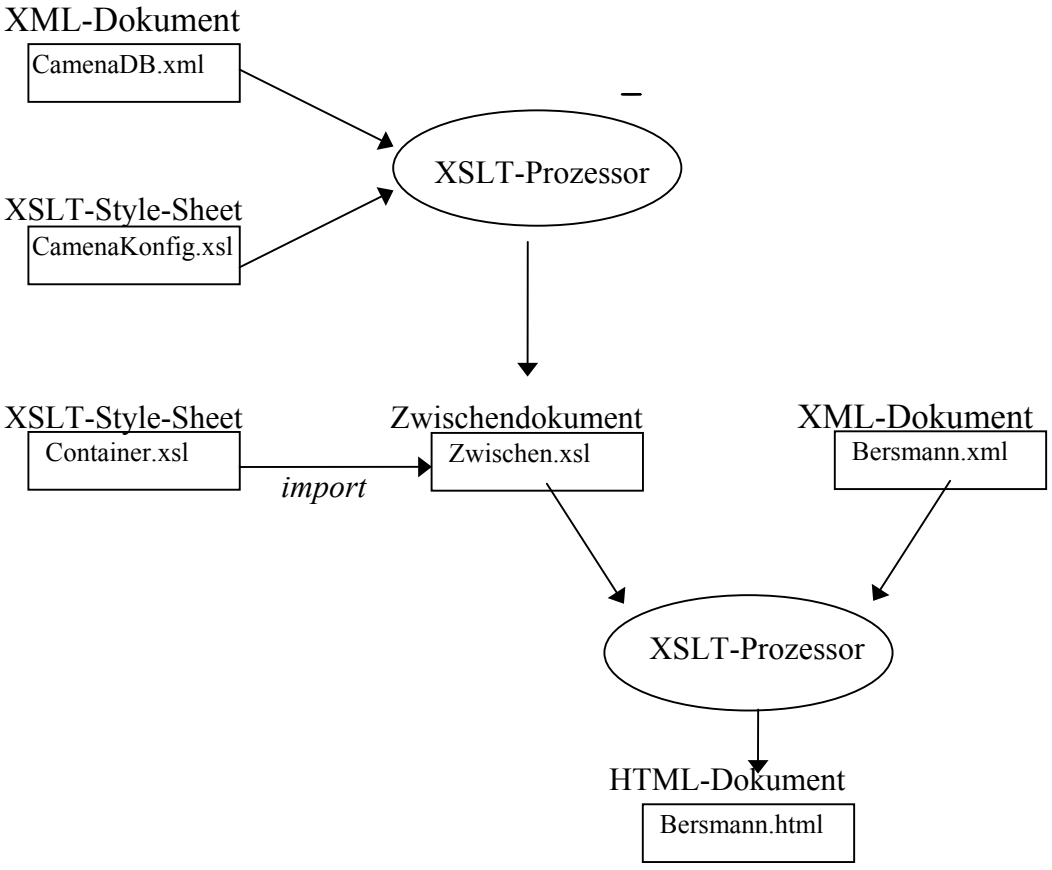

Abbildung 8: Transformationsprozess mit allen vorhandenen Dokumenten

Je nachdem, welche Datensätze in der CamenaDB markiert sind, wird über das Konfigurationsstylesheet CamenaKonfig.xsl ein anderes Zwischendokument erstellt. So

ist das Zwischendokument an beliebige XML-Dokumente anpassbar. In Abbildung 8 wurde ein Zwischendokument speziell für das Dokument Bersmann.xml generiert.

3.3. Die CamenaDB

CamenaDB ist ein typisches XML-Dokument. Sie besteht aus einem Auszeichnungsteil und einer DTD.

In der DTD sind alle Tags definiert, die für einen Datensatz des Camena-Projektes notwendig sind. Diese sind das Root-Tag <Camena>, das Tag <Werk>, also ein literarisches Werk eines Autors des Camena-Projekts, welches einen kompletten Datensatz in der CamenaDB repräsentiert, und die Tags <Name>, <First>, <Last>, <URL> und <Type>, die in dem Tag <Werk> enthalten sind.

Das Tag <Name> gibt den Namen des Autors an, dessen Werk sowohl als XML-Dokument vorliegt, als auch in abfotografierter, digitaler Version zur Verfügung steht.

Die Tags <First> und <Last> sind der erste und der letzte Bezeichner der abfotografierten Graphiken der originalen lateinischen Texte. Diese Bezeichner bestehen aus einem Kürzel für den Autor und einer laufenden Nummer für die Seitenzahlen im Original. Somit hat beispielsweise das erste Bild des Autors Bersmann den Bezeichner: ‘bers001’ und das letzte Bild den Bezeichner: ‘bers176’.

Im Tag <URL> ist die eindeutige URL-Adresse (Uniform Ressource Locator) angegeben, an der die Bilder im Netz zu finden sind.

Das Tag <Type> ist dazu da, den MIME-Typ des jeweiligen Bildes zu spezifizieren. Verwendet wurden bisher Gif-Grafiken (.gif) oder JPEG-Grafiken (.jpg).

Im Anschluss an die Dokumenttyp Definition befinden sich die konkreten Auszeichnungen des Dokuments. Hier ist das Prinzip der Verschachtelung der Elemente gut zu erkennen. Das Root-Element Camena umschließt alle Werk-Elemente mit den kompletten Datensätzen; diese wiederum umschließen all die Tag-Elemente, die das jeweilige Werk genauer spezifizieren.

In Listing 3 ist der Source-Code der CamenaDB.xml zu sehen. Damit das Listing nicht zu viel Platz einnimmt, sind nur die Datensätze der Autoren Bersmann und Gesner mit in das Listing übernommen worden.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Camena[

<!ELEMENT Camena (Werk)+>
<!ELEMENT Werk (Name, First, Last, URL, Type)>
<!ATTLIST Werk print (true|false) #REQUIRED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT First (#PCDATA)>
<!ELEMENT Last (#PCDATA)>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
]>

<Camena>
  <Werk print="true">
    <Name>Bersmann</Name>
    <First>bers000</First>
    <Last>bers177</Last>
    <URL>http://www.uni-mannheim.de/mateo/camena/bers1/jpg/</URL>
    <Type>.jpg</Type>
  </Werk>
  <Werk print="false">
    <Name>Gesner</Name>
    <First>gesner000</First>
    <Last>gesner999</Last>
    <URL>http://www.uni-mannheim.de/mateo/camena/gesner1/</URL>
    <Type>.gif</Type>
  </Werk>
</Camena>
```

Listing 3: Source-Code der CamenaDB

3.4. Die XML-Quelltexte

Zum besseren Verständnis der Funktionsweise des nachfolgenden Camena-Konfigurationsstylesheets muss auch noch kurz die Struktur der XML-Quelldateien erläutert werden.

Alle XML-Quelldateien des Camena-Projekts sind nach den Richtlinien der Text Encoding Initiative (TEI) ausgezeichnet. Dafür wurde auf Grundlage der TEILITE (TEI für LIterarische TExte) eine „Tag-Library“ von 50 Markierungen erstellt. So wird die Syntax alle XML-Quelldateien durch die TEIXLITE.DTD, die dem WWW-Standard für XML folgt, bestimmt.

Das Root-Element einer solchen XML-Quelldatei ist das Element <text>. Darin enthalten sind die Elemente <front> für einen Vorspann, <body> für den Hauptteil und <back> für einen abschließenden Teil. Diese können sich wiederum unterteilen in Abteilungen , Überschriften <head>, evtl. einen Vorspann <opener> oder metrischen Text von mehr als einem Vers Umfang bzw. ein Gedicht oder ein episches Buch <lg>.

Wesentlich dabei ist, dass die als XML-Dokument vorliegenden Texte Abschriften von ursprünglich mittelalterlichen Werken sind. Um nun Forschern die Recherche zu erleichtern, soll zwischen den späteren HTML-Dokumenten und den abfotografierten Originalseiten hin und her gewechselt werden können. Dazu ist im XML-Dokument das Element Pagebreak <pb> eingesetzt, welches jede neue Seite markiert. Je nachdem, ob die Seite des Originaltextes einseitig oder doppelseitig abfotografiert ist, ist dann in der XML-Datei jedem oder jedem zweiten Pagebreak-Element ein eindeutiges id-Attribut zugeordnet.

So kann anhand des Pagebreak-Elements eine Verknüpfung zwischen XML-Dokument und fotografierter Grafik hergestellt werden.

3.5. Das Camena Konfigurationsstylesheet

Eine der Aufgaben des Camena Konfigurationsstylesheet ist das Auslesen der markierten Datensätze aus der CamenaDB.

Dabei werden mehrere Fälle entsprechend der Verschachtelungstiefe des Pagebreak-Elements <pb> im XML-Quelltext unterschieden, wobei vornehmlich auf die besondere Behandlung von Verszeilen innerhalb eines <lg>-Elements geachtet werden

muss, damit das Versmaß aus den originalen Manuskripten auch im HTML-Dokument erhalten bleibt.

Damit nicht die gesamte CamenaDB ausgelesen und verarbeitet wird, muss in einem ersten Schritt überprüft werden, welche Datensätze überhaupt verarbeitet werden sollen. Hierfür können die Datensätze in der CamenaDB mit Hilfe des Attributs „print“ des Elements <Werk> markiert werden. Soll ein Datensatz verarbeitet werden, so ist das Attribut auf „true“ gesetzt, andernfalls auf „false“. Somit können beliebig viele Datensätze verarbeitet werden, bzw. ein Style-Sheet erstellt werden, welches beliebig viele XML-Dokumente verarbeiten kann.

Ausgelesen werden dann die Inhalte der Elemente <First>, <Last>, <URL> und <Type>. Welche Werte durch diese Tags repräsentiert werden, wurde schon im Abschnitt 3.2. erläutert.

Wenn klar ist, welche Datensätze verarbeitet werden sollen, kann die Hauptaufgabe des Camena Konfigurationsstylesheet, die Erstellung eines Zwischendokuments, beginnen. Hierfür wird der Prolog des neuen Zwischen-Style-Sheets gleich im Root-Template <xsl:template match="/"> angelegt. Im nachfolgenden Camena-Template wird dann die Pagebreak-Behandlung für die Links zu den Bildern definiert, wie sie später im Zwischendokument vonstatten gehen soll. Um, wie oben schon beschrieben, sämtliche Fälle von Verschachtelungstiefen zu berücksichtigen, und um zu vermeiden, daß die Verarbeitung rekursiv abläuft, wird das Camena-Template dreimal hintereinander in verschiedenen Modi aufgerufen (Siehe hierzu Listing 4). Die Linearität wird erzwungen, um im Zwischen-Style-Sheet eine serielle Bearbeitungsreihenfolge zu erhalten.

```
<xsl:apply-templates select="Camena" mode="ebene0" />  ...

<xsl:apply-templates select="Camena" mode="ebene1" />  ...

<xsl:apply-templates select="Camena" mode="ebene2" />  ...
```

Listing 4: Erzwingen einer linearen Verarbeitung

Wird das Camena Konfigurationsstylesheet auf die CamenaDB angewandt, so resultiert aus dem Transformationsprozess das Zwischendokument.

3.6. Das Zwischendokument

Das Zwischendokument ist ein Style-Sheet, welches nur kurzzeitig existiert. Aufgabe dieses Zwischen-Transformationsskriptes ist es, die XML-Quelltexte von ausgewählten Autoren des Camena-Projekts in HTML-Format zu konvertieren. Nach seiner Verwendung wird es wieder gelöscht.
Zur Konstruktion des HTML-Dokuments ist zunächst eine HTML-Vorlage definiert, die die Daten der XML-Quelldatei in geeigneter Weise aufnimmt.
Hierfür ist im Root-Tamplate <xsl:template match="/">, wie in Listing 5 zu sehen ist, die gesamte Dokumentstruktur der zu erstellenden HTML-Datei samt Styledefinition angelegt.

```
<xsl:template match="/">
  <HTML>
    <HEAD>
      <TITLE> Camena Style-Sheet </TITLE>
        <Style type="TEXT/CSS">
            .fig_design1{ ... }   ...
            BODY {margin:1cm}
        </Style>
    </HEAD>
    <BODY bgcolor="beige">
      <xsl:apply-templates />
    </BODY>
  </HTML>
</xsl:template>
```

Listing 5: Template zur Definition der HTML-Dokumentstruktur

Weitere drei Schablonen existieren für die Pagebreak-Behandlung mit Funktionalitäten, wie sie bei der Erstellung des Camena Konfigurationsstylesheet aufgrund der unterschiedlichen Verschachtelungstiefen schon angedacht waren (siehe hierzu auch die verschiedenen Aufrufe der Templates mit den jeweilige Bedingungen für das "match"-Attribut in Listing 6).

```
<xsl:template match="pb|pb[@id]"> ...

<xsl:template match="p/pb|p/pb[@id]">  ...

<xsl:template match="lg/l/pb[@id]|lg/l/pb">...
```

Listing 6: Definition von Schablonen zur Pagebreak-Behandlung

In der obersten Schablone werden Pagebreak-Elemente auf oberster Ebene behandelt, in der zweiten Schablone werden die Pagebreak-Elemente bearbeitet, die innerhalb eines Absatzes <p> stehen und die dritte Schablone bearbeitet Pagebreak-Elemente, die in metrische Texte, Gedichte oder ein episches Buch <lg> verschachtelt sind. Diese differenzierte Betrachtungsweise ist notwendig, da ein Pagebreak-Elemente später im HTML-Dokument durch eine „Logcal Division“ repäsentiert wird, welche ein „Horizontal-Rule“-Element zur Seitenabgrenzung, ein „Anchor“-Element zur Verknüpfung mit den Grafiken der Originalausgaben und ein „Break“-Element zur Ausgabe eines Zeilenumbruchs beinhaltet (siehe hierzu Listing 7).
Innerhalb des HTML-Quelltextes wird so durch das Vorhandensein eines „Logical Division“- Elements das jeweilige Tag, welches den äußeren Rahmen bildet, ausgeschaltet und muss deshalb nach Beendigung der -Anweisung wieder eingeschaltet werden.

```
<div align="center">
    <hr/>
    <a>
       <xsl:attribute name="href">
              http://www.uni-mannheim.de/mateo/camena/fuerst1/jpg/
               <xsl:value-of select='@id'/>.jpg
       </xsl:attribute>
       <xsl:attribute name="target">Bild</xsl:attribute>
            Grafik: <xsl:value-of select='@id'/>
    </a>
    <br/>
</div>
```

Listing 7: HTML Quelltext eines „Logical Division“- Elements im Zwischendokument

Die weiteren Verarbeitungsschritte innerhalb der Schablonen sind weitgehend identisch und werden im folgenden zusammengefasst.

Zuerst wird innerhalb einer <xsl:choose>-Anweisung abgefragt, welche Autoren überhaupt verarbeitet werden sollen, bzw. für welche Autoren-XML-Dokumente eine HTML-Datei erstellt werden soll. Der Zugriff erfolgt dann über die Attributwerte der Pagebreak-Elemente innerhalb des jeweiligen XML-Dokuments. Dazu geben die Abfragewerte die Intervallgrenzen an, innerhalb deren sich die Attributwerte des Pagebreak-Elements des ausgewählten Camena-Werkes befinden sollen. Hier findet dann auch die Verknüpfung mit den abfotografierten Graphiken der lateinischen Originaltexte statt, wenn deren Bezeichner und das id-Attribut des Pagebreak-Elements identisch sind.

Die weiteren Funktionalitäten, die das Transformationsskript erfüllen muss, sind aus Gründen der Übersichtlichkeit und einfacheren Wartung in einem weiteren Style-Sheet, dem ContainerXSL, definiert und werden bei Bedarf importiert.

3.6. Das Style-Sheet ContainerXSL

Die Templates im ContainerXSL-Style-Sheet dienen in erster Linie der Weiterleitung rekursiver Aufrufe an XML-Dokumentknoten, die weiter unten in Hierarchieebenen liegen; der Verarbeitung von Elementen, die keine besonders umfangreichen Funktionalitäten beinhalten; und der Behandlung des <name>-Elements und des „Reference String“-Elements.
<name>-Elemente sind Eigennamen, die als Element markiert werden. <rs>-Elemente werden verwendet, wenn ein Eigenname gleichzeitig auch eine Gattungsbezeichnug darstellt. Beide Elemente besitzen ein Attribut „type“, welches das Bezeichnete genauer klassifiziert.
Deshalb werden in einer <xsl:choose>-Anweisung die verschiedenen Ausprägungen dieses Attributs abgefragt. Die weiteren Funktionalitäten dieses Templates werden erst dann implementiert, wenn für die auszuwählenden Attributausprägungen Inhalte erstellt sind und die Möglichkeit besteht, auf diese Inhalte zuzugreifen.

3.7. Das Ergebnis im Browser

Das Ergebnis der Transformation ist ein HTML-Dokument. Es besteht aus einer Textdatei im ASCII-Format, die eingebettete HTML-Tags enthält. Der Aufbau dieser HTML-Seite ist relativ einfach strukturiert. So besteht der Dokumentrumpf hauptsächlich aus Paragraphen, die die Textpassagen der Online-Editionen enthalten und „Logical Division“- Elementen, die die Verweise zu den abfotografierten Originalen beinhalten.

4. Fazit

Die Speicherung der Daten des Camena Projekts als XML-Dokumente und deren Konvertierung mit Hilfe von XSLT-Transformationsskripten in das XHTML-Format ist eine neue Art der Präsentation von Inhalten im Web. Für HTML galt bislang die Tatsache, dass die dargestellten Daten und Informationen in Web-Seiten nicht mehr maschinenlesbar abgebildet werden können. Es kam also zu einem Informationsverlust. Eine automatische Weiterverarbeitung der Daten war nicht möglich. Mit Hilfe von XML kann diese Beschränkung aufgehoben werden. Die Informationen in den XML-Dokumenten des Projekts Camena sind ja gerade zur Recherche und Weiterverarbeitung gedacht.

Die XSLT-Style-Sheets sollen hierbei helfen, Handlungsabläufe bei der Darstellung der XML-Dokumente zu vereinfachen und somit das Camena-Projekt und die Forschung im Bereich der neulateinischen Philologie zu unterstützen.

Literaturangaben:

Apache Software Foundation;
Xalan-Java version 2.2.D6;
http://xml.apache.org/xalan-j/index.html, 2001;

Bliemel, Friedhelm / Fassott, Georg / Theobald, Axel;
Electronic Commerce. Herausforderungen - Anwendungen - Perspektiven;
Gabler Verlag, Wiesbaden, 1999;

Born, Günther;
Jetzt lerne ich XML;
Markt+Technik Verlag, München, 2001;

Eberhard, Andreas / Fischer, Stefan;
Java-Bausteine für E-Commerce Anwendunngen,
Verteilte Anwendungen mit Servlets, CORBA und XML;
Carl Hanser Verlag, München / Wien, 2000;

Harms, Florian / Koch, Daniel / Kürten, Oliver;
Das große Buch HTML & XML;
DATA BECKER GmbH & Co. KG, Düsseldorf, 2000;

Jones, Russ / Nye Adrian;
HTML und das World Wide Web: Selbst publizieren im WWW;
O´Reilly / International Thomson Verlag GmbH & Co. KG, Bonn, 1995;

Knobloch, Manfred / Kopp, Matthias;
Web-Design mit XML;
dpunkt.verlag, Heidelberg, 2001;

Kobert, Thomas;
XML, Das bhv Taschenbuch;
Bürohandels- und Verlagsgesellschaft mbH, Kaarst, 1999;

Michel, Thomas;
XML kompakt - Eine praktische Einführung;
Carl Hanser Verlag; München/Wien, 1999;

Münz, Stefan;
SELFHTML: Einführung / Internet, WWW und HTML /
Entstehung des World Wide Web;
http://www.netzwelt.com/selfhtml/tbad.htm, 1998;

North, Simon / Hermans, Paul;
XML in 21 Tagen(CD-ROM);
Markt+Technik Verlag, München, 1999;

Pott Oliver / Wielage,Gunter;
XML – new technology, Praxis und Referenz;
Markt+Technik Verlag, München, 2000;

Seeboerger-Weichselbaum, Michael;
XML, das Einsteigerseminar;
Bürohandels- und Verlagsgesellschaft mbH, Kaarst, 2000;

Tolksdorf, Robert;
Die Sprachen des Web: HTML und XHTML
Informationen aufbereiten und präsentieren im Internet;
dpunkt.verlag, Heidelberg, 2000;

Universitätsbibliothek Mannheim / Rechenzentrum der Universität Mannheim;
MATEO - editio theodoro-palatina online;
http://www.uni-mannheim.de/mateo/camena.html, 2001;

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Studienarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mannheim, den 17.07.2001 Matthias Robert Grünewald