Haußer, Felix

Research Report

XML-basierte Anreicherung von Texten - Potentiale für Verlage

Erlanger Beiträge zur Medienwirtschaft, No. 03/2014

Provided in Cooperation with:
Friedrich-Alexander University of Erlangen-Nuernberg (FAU), Institute for the Study of the Book, Professorship of E-Publishing and Digital Markets

Suggested Citation: Haußer, Felix (2014) : XML-basierte Anreicherung von Texten - Potentiale für Verlage, Erlanger Beiträge zur Medienwirtschaft, No. 03/2014, Friedrich-Alexander-Universität Erlangen-Nürnberg, Institut für Buchwissenschaft, Professur für Buchwissenschaft, insb. E-Publishing und Digitale Märkte, Erlangen, https://nbn-resolving.de/urn:nbn:de:bvb:29-opus4-48842

This Version is available at:
https://hdl.handle.net/10419/223331

Erlanger Beiträge zur Medienwirtschaft
Hg.: Svenja Hagenhoff

Beitrag Nr. 03/2014
Hrsg.: Svenja Hagenhoff

Felix Haußer

XML-basierte Anreicherung von Texten – Potentiale für Verlage

Erlanger Beiträge zur Medienwirtschaft

Herausgegeben von Svenja Hagenhoff
Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Buchwissenschaft
Professur für E-Publishing und Digitale Märkte
Katholischer Kirchenplatz 9
91054 Erlangen

Erlangen Contributions to Media Management and Media Economics

Edited by Svenja Hagenhoff
Friedrich-Alexander University of Erlangen-Nuernberg
Chair of the Study of the Book
Professorship of E-Publishing and Digital Markets
Katholischer Kirchenplatz 9
91054 Erlangen / Germany

Tel. +49 (0) 9131 / 85-24700
Fax +49 (0) 9131 / 85-24727
www.buchwiss.uni-erlangen.de
buwi-ebm@fau.de

Vorbemerkung

Eine Version der vorliegenden Arbeit wurde im Wintersemester 2013/2014 unter dem Titel »XML-basierte Anreicherung von Texten – Potentiale Für Verlage« als Abschlussarbeit im Studiengang Buchwissenschaft (Master) der Friedrich-Alexander-Universität Erlangen-Nürnberg eingereicht.

Erstgutachten: Prof. Dr. Svenja Hagenhoff, Professur für Buchwissenschaft, insb. E-Publishing und Digitale Märkte.

Zweitgutachten: Prof. Dr. Günther Görz, Department Informatik.

Abstract

The German book market has been changing during the last decade. Electronic publishing has started to play a significant role and continues to become more and more important. New technological advancements lead to new challenges for publishers as well as sellers. The paper *XML-based Text Enrichment. Potentials for Publishers* discusses how the usage of XML-technologies can help to manage these challenges.

The paper focuses on a presentation of applied XML-technologies which are required for semantic text enrichment and cross-platform publishing (XML, XSD, DTD, XSLT, XPATH, XSL-FO, CSS and EPUB). For illustration purposes, XML-technologies are presented using a semantically enriched version of Kafka's *Die Verwandlung*, created by the author of this paper. Furthermore, these technologies will be used to build a content-management-system (CMS), which allows the integration of new text files into a methodical workflow.

The paper shows and emphasizes the potential of XML-based text enrichment and content management, especially for publishers who use or plan to use digital sales channels. In order to minimize the risks that come with the implementation of CMS, innovation and change management must prepare for and accompany these processes.

Keywords and Classification

BKL: 54 Informatik
DDC: 070 Publizistische Medien, Journalismus, Verlagswesen
JEL: O Economic Development, Technological Change, and Growth

Publishing Industry; XML-Technologies; Markup Language; Text Enrichment; Cross Platform Publishing; Content Management.

Inhaltsverzeichnis

Abbildungsverzeichnis.....VII

Verzeichnis der Listings.....VIII

Abkürzungsverzeichnis.....IX

1 Einleitung.....1

2 Theoretische Fundierung I: Content-Management.....3

2.1 Forschungsstand Content-Management.....3

2.2 Daten, Informationen und Wissen.....6

2.3 Die Begriffe Content und Management.....8

2.4 Content-Management in der Anwendung.....11

2.4.1 Content-Management als Prozess.....11

2.4.2 Content-Management-Anwendungen.....13

2.5 Ein Beispiel aus der Praxis: TextGrid.....14

3 Theoretische Fundierung II: eXtensible Markup Language (XML).....17

3.1 Die Bausteine von XML.....18

3.2 Weitere Schlüsseltechnologien.....23

3.2.1 Inhaltsmodelle mit DTD und XSD.....23

3.2.2 XSLT.....25

3.2.3 XPath.....26

3.2.4 XSL-FO.....28

3.2.5 CSS.....30

3.2.6 EPUB.....31

4 Die Umsetzung XML-basierter Textanreicherung – Von XML zum Zielformat 33

4.1 Versuchsaufbau 33

4.2 Programmierschnittstellen und Prozessoren 34

4.3 Anreicherung von Text-Dateien: Das XML-Dokument 35

4.3.1 Der Prolog 36

4.3.2 XML-Daten 36

4.4 Strukturierung: XML-Schema-Definition (XSD) 41

4.4.1 Der Aufbau einer XSD 41

4.4.2 Designvarianten 44

4.4.3 Namensräume in XML 45

4.4.4 Schematische Darstellung der Schema-Definition 47

4.5 Zielformat I: XHTML 49

4.5.1 Der Aufbau eines XHTML- Dokuments 49

4.5.2 Die Bausteine von XSLT 50

4.5.3 Der Aufbau eines XSLT-Stylesheets 52

4.6 Zielformat II: PDF 56

4.6.1 Die Bausteine von XSL-FO 56

4.6.2 Der Aufbau eines XSL-FO-Stylesheets 56

4.7 Zielformat III: EPUB 61

4.7.1 Open Container Format 61

4.7.2 Open Publication Standard 62

4.7.3 Open Publication Format 62

5 Evaluation 65

5.1 Freie oder proprietäre Software 65

5.2 Externe Faktoren: Der Buchmarkt 66

5.3 Interne Faktoren: Der Verlag 67

5.4 Risiken 67

6 Fazit 70

Literaturverzeichnis 72

Anhang 77

1. Bilddateien 77

2. XML-Dokument 79

3. XML-Schema-Definition 86

4. XSLT-Stylesheet 90

5. XSL-FO-Stylesheet 93

6. EPUB 102

Abbildungsverzeichnis

Abbildung 1: Referenzialität von Content in Anlehnung an STEFAN SPÖRRER 2009, S. 5-6 10

Abbildung 2: Content-Management-Anwendungen 13

Abbildung 3: XML Elementsyntax nach VONHOEGEN 2011, S. 55 19

Abbildung 4: Schematische Abbildung eines XML-Dokuments nach ECKSTEIN / ECKSTEIN 2004, S.19 22

Abbildung 5: Einfache und komplexe Datentypen 24

Abbildung 6: XSLT-Transformation 26

Abbildung 7: XPath-Pfade nach BONGERS 2008, S. 56 27

Abbildung 8: XSL-FO Formatierungsprozess 30

Abbildung 9: Schematischer Versuchsaufbau 33

Abbildung 10: Generisches Content-Management 39

Abbildung 11: XML-Namensräume 45

Abbildung 12: Schematische Darstellung der Schema-Definition mit XMLSpy 48

Abbildung 13: XSL-Instruktionen und Literal Result Elements nach BONGERS 2008, S. 33 51

Abbildung 14: Ordnerstruktur EPUB 61

Verzeichnis der Listings

Listing 1: XML Grundaufbau 20
Listing 2: XML-Prolog 36
Listing 3: XML-Daten 37
Listing 4: Namensräume einsetzen 47
Listing 5: Aufbau eines XHTML-Dokuments 50
Listing 6: Das XSLT-Stylesheet 53
Listing 7: Das XSL-FO-Stylesheet 58
Listing 8: Aufbau von kafka.opf 63

Abkürzungsverzeichnis

AIIM	Association for Information and Image Management
B2B	Business-to-Business
B2C	Business-to-Consumer
CM	Content-Management
CMA	Content-Management-Anwendung
CMS	Content-Management-System
CSS	Cascading Style Sheets
DIKWP	Data-information-knowledge-wisdom-pyramid
DOM	Document Object Model
DTD	Dokumententyp-Definition
ECM	Enterprise-Content-Management
EPUB	Electronic Publication
FOP	Formatting Objects Processor
GFK	Gesellschaft für Konsumforschung
HTML	Hypertext Markup Language
IDPF	International Digital Publishing Forum
Infoset	XML Information Set
ISBN	Internationale Standardbuchnummer
LRE	Literal Result Element
MIME	Multipurpose Internet Mail Extension
MSXML	Microsoft XML Core Services
OASIS	Organization for the Advancement of Structured Information Standards
OCF	Open Container Format
OEBPS	Open eBook Publication Structure

OOXML	Office Open XML
OPF	Open Publication Format
OPS	Open Publication Structure
PDF	Portable Document Format
pwc	Pricewaterhouse Coopers
QName	Qualified Name
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
TEI	Text Encoding Initiative
TextGridLab	TextGrid Laboratory
TextGridRep	TextGrid Repository
URL	Uniform Resource Locator
VRE	Virtual Research Environment
W3C	World Wide Web Consortium
WCM	Web-Content-Management
XHTML	Extensible HyperText Markup Language
XML	eXtensible Markup Language
XPath	XML Path Language
XQuery	XML Query Language
XSD	XML-Schema-Definition
XSL	eXtensible Stylesheet Language
XSL-FO	XSL Formatting Objects
XSLT	XSL Transformation

1 Einleitung

Der deutsche Buchmarkt befindet sich in einem ständigen Wandel. Electronic-Publishing spielt in der Wertschöpfungskette der Akteure des deutschen Buchmarkts eine immer wichtigere Rolle. Einer aktuellen Studie des *Börsenvereins des deutschen Buchhandels* zufolge, erscheint bereits die Hälfte aller Novitäten sowohl in gedruckter Form als auch digital als E-Book. Auf der anderen Seite nimmt die Akzeptanz digitaler Formate und elektronischer Endgeräte bei den Endabnehmern der Buchbranche immer weiter zu. Die Zahl der verkauften E-Books vervielfacht sich jährlich (vgl. Börsenverein 2013, S. 10–11). Für den herstellenden Buchhandel hat das weitreichende Folgen. Das *klassische* Buchformat[1] wird erweitert durch eine Reihe medienkonvergenter Einzelformate. Buchverlage vertreiben Content neben gedruckten Büchern über E-Books, Web-Apps und native Applikationen. Viele Verlage haben das Potential von Apps für den deutschen Buchmarkt längst erkannt.[2] Neue Technologien ermöglichen zudem neue Strategien und Erlösmodelle: Digitale, Cloud-basierte Bibliotheken erlauben ihren Benutzern unmittelbaren, flexiblen und internet-basierten Zugang zu einer Vielzahl von Büchern und anderen Medienformaten.[3] Data-Mining-Technologien und Textanreicherungsverfahren ermöglichen die Auswertung großer Datenbestände im Kontext der ständig wachsenden Informationsflut.

Zugleich bringen die neuen technologischen Voraussetzungen eine Reihe von Herausforderungen mit sich. Eine zentrale Herausforderung für Verlage ist die Verwaltung ihrer Inhalte. Diese müssen digital aufbereitet, organisiert und strukturiert werden, um eine möglichst effiziente Nutzung zu gewährleisten. Eine verbreitete Methode, um Inhalte zu strukturieren, ist die Textanreicherung auf Basis der *eXtensible Markup Language* (XML). XML ist universal einsetzbar: Mit der Bezeichnungssprache können Strukturen und Informationszusammenhänge in Textdateien offengelegt werden. Dieser Vorgang vereinfacht einerseits die gezielte Informationsbeschaffung und -verwaltung. Außerdem können mit XML angereicherte Texte plattformunabhängig publiziert werden. Die vorliegende Masterarbeit soll untersuchen, inwiefern sich Potentiale aus einer XML-basierten Textanreicherung für den herstellenden Buchhandel ergeben. Hierzu soll dargelegt werden, wie Textdateien mit XML angereichert werden und wie XML in den Produktionsprozess eines Verlages integriert werden kann.

Der Prozess von der Erstellung, Anreicherung, Gestaltung und Publikation bis zur Archivierung solcher Textdateien wird mit dem Begriff *Content-Management* (CM) umschrieben. CM umfasst alle Vorgänge, die das Ziel verfolgen, unterschiedlich gestaltete Formen und Formate von Content zu verwalten. CM-

[1] Vgl. hierzu: RAUTENBERG 2003, S. 82–83: Das Buch ist ein physisches Objekt oder elektronisches Speichermedium und besteht aus einem Trägermaterial und den darauf abgebildeten Sprach- und Bildzeichen. Bücher sind Produkte eines handwerklichen oder maschinell geprägten Herstellungsprozesses.

[2] Vgl. hierzu: *Carlsen*: http://www.carlsen.de/digitale-produkte/apps [22.03.2014]; *Kiepenheuer & Witsch*: http://www.kiwi-verlag.de/buch/schoener-warten-app/978-3-462-31009-2/[22.03.2014]; *Gräfe und Unzer*: http://www.gu.de/ebooks-und-apps/ [22.03.2014].

[3] Vgl. hierzu: *Safari Books Online* (O'Reilly Media): http://www.safaribooksonline.com/about-us [22.03.2014], *Skoobe* (Bertelsmann / Holtzbrinck): https://www.skoobe.de/help [22.03.2014].

Prozesse können unter anderem auf der Basis von XML ablaufen. So lassen sich insbesondere Strukturierungsvorgaben und Transformationen mit XML-Subsprachen umsetzen, wie sich im Verlauf der Masterarbeit zeigen wird.

Die Masterarbeit ist aufgebaut wie folgt: Der erste Teil dient der theoretischen Fundierung der Thematik. In Kapitel 2 sollen vor allem die Begriffe *Content* und *Management* geschärft werden. Da Content ein komplexer Begriff ist, der je nach Verwendung unterschiedliche Begriffsinhalte haben kann, soll er im Kontext der *data-information-knowledge-wisdom-pyramid* (DIKWP) von RUSSEL ACKOFF erläutert werden. Außerdem gilt es, CM als Prozess darzustellen, um einen ersten Einblick zu ermöglichen, wie CM in Verlagen eingesetzt werden kann. Das zweite Kapitel schließt mit der Vorstellung von *TextGrid* ab. TextGrid ist ein CM-System aus der humanwissenschaftlichen Praxis. Kapitel 3 dient ebenfalls der theoretischen Fundierung. In diesem Kapitel sollen die Grundlagen von XML erläutert werden. Das Kapitel beginnt mit einem kurzen Abriss zur Geschichte von XML und soll anschließend die Bausteine und Schlüsseltechnologien, die für den praktischen Teil der vorliegenden Masterarbeit notwendig sind, skizzieren.

Im zweiten, praktischen Teil der Masterarbeit wird die Umsetzung eines möglichen CMS auf der Basis von XML vorgestellt. Kapitel 3 folgt einem Workflow in vier Schritten. Schritt eins sieht die Erstellung eines XML- Dokuments vor, aus dem im zweiten Schritt ein Inhaltsmodell abgeleitet wird. Dieses Inhaltsmodell ist die Basis des eigentlichen Content-Managements, da es Vorgaben über die Gestaltung zukünftiger Textdateien festlegt. In Schritt drei erfolgt die Transformation der Ausgangsdateien in die Zielformate XHTML, PDF und EPUB. In einem vierten Schritt sollen die produzierten Zielformate schließlich publiziert werden. Die für die Zielformate verwendeten Bilddateien stammen von der Gestalterin FRIEDERIKE WOLF.[4] Die für den praktischen Teil programmierten Stylesheets verfolgen das Ziel, alle Anfangsdateien, die dem Inhaltsmodell von Schritt zwei entsprechen, automatisch in das gewünschte Zielformat zu transformieren. Der praktische Teil der Masterarbeit soll in Kapitel 5 evaluiert werden. Dieses Kapitel soll Aufschluss über Potentiale und Risiken eines XML-basierten Content-Managements geben.

Die vorliegende Masterarbeit bildet eine Schnittstelle zwischen der Buchwissenschaft und der Informatik, insbesondere der Teilbereiche *Digital Humanities* und Wirtschaftsinformatik. Unter Digital Humanities werden computergestützte Verfahren zusammengefasst, die in geisteswissenschaftlichen Arbeitsbereichen Anwendung finden. Die Verwendung der Bezeichnungssprache XML sowie verwandter Subsprachen wie XSLT und XSL-FO fallen in den Fachbereich der Informatik. Content-Management-Systeme als Softwarelösungen für die Verwaltung von Informationseinheiten in Organisationen sind Gegenstand der Wirtschaftsinformatik. Der buchwissenschaftliche Anspruch der Masterarbeit zeigt sich in der Problemstellung. Diese soll klären, inwiefern sich Potentiale und Anwendungsmöglichkeiten aus einer XML-basierten Textanreicherung für Verlage ergeben.

[4] Vgl. hierzu: http://www.friederikewolf.com/ [07.07.2014]

2 Theoretische Fundierung I: Content-Management

Unter Content-Management (CM) werden gemeinhin Vorgänge verstanden, die das Ziel verfolgen, unterschiedlich gestaltete Formen von Content zu verwalten. Der Begriff Content unterliegt allerdings im Kontext seines Managements vielfältigen Bedeutungen. In diesem Kapitel soll daher geklärt werden, wie die Begriffe Content und Management gefasst werden können und wie sich Daten, Content, Informationen und daraus resultierendes Wissen unterscheiden. Außerdem sollen die Bestandteile und Prozesse eines CM am Modell erläutert werden. Das zweite Kapitel wird mit der Vorstellung von *TextGrid*, einem Beispiel aus der wissenschaftlichen Praxis, abgeschlossen. Zunächst gilt es aber, den Forschungsstand um das Thema CM zu beleuchten.

2.1 Forschungsstand Content-Management

Content-Management entwickelte sich im Kontext der rasanten technologischen Entwicklungen, die bereits Mitte der 1990er-Jahre durch den Bedeutungszuwachs des Internets hervorgerufen wurden. Durch neue, elektronische und netzbasierte Medien sanken die Kosten für die Vervielfachung und Distribution von Content auf ein Minimum (vgl. Hagenhoff et al. 2007, S. 130). Durch diesen Umstand begründet entstanden neue Finanzierungs- und Verwaltungsmodelle für Content. Mit Hilfe von Webpublishing-Strategien versuchten Unternehmen schon frühzeitig, eigene Websites als dynamische Informationsquellen und digitale Repräsentationen zu gestalten. Vor allem die Forderung nach Aktualität aber auch der Bedarf, die Vielfalt unterschiedlicher Formate möglichst sinnvoll und einfach zu verwalten, führten zu der Entwicklung komplexer Content-Management-Systeme (CMS) (vgl. Spörrer 2009, S. 1–2). In der Folgezeit entstand eine Fülle proprietärer und offener CMS. Das Portal cmscritic.com5 listet aktuell insgesamt 161 CMS auf, wovon 30 sogenannte Enterprise Management Systeme sind (vgl. Cmscritic, 2014).

CM wird in der Literatur vielfach diskutiert und entsprechend unterschiedlich definiert (vgl. Koop et al. 2001, S. 14–16; Bullinger et al. 2001, S. 8–10; Hansen / Neumann 2001, S. 452; Christ 2001, S. 22–23; Rothfuss / Ried 2003, S. 14–17; Ehlers 2003, S. 105–108; Christ 2003, S. 15–16; Bodrow / Bergmann 2003, S. 20–22; Boiko 2005, S. 63–198; Wilhelm 2006, S. 56–57; Bodendorf 2006, S. 95–96; Fuchs 2007, S. 9–12; Fröschle / Behrendt 2007, S. 8–9; Riggert 2009, S. 1–9; Spörrer 2009, S. 5–8). Der Forschungsstand lässt sich den Anwendungsgebieten von CM entsprechend aufteilen. Nachfolgend sollen die wichtigsten Tendenzen nachvollzogen werden.

Einen Einstieg in das Thema Content-Management gibt STEFAN SPÖRRERS *Content Management Systeme – Begriffsstruktur und Praxisbeispiele*. Das Buch zeigt insbesondere einen Überblick über die Definitionen und Abgrenzungen unterschiedlicher Systeme im Bereich des CM. SPÖRRER differenziert dabei zwi-

[5] Vgl. hierzu: http://www.cmscritic.com/ [11.03.2014].

schen CM und Dokumenten- sowie Wissensmanagement, Web-Content-Management und Enterprise-Content-Management. Außerdem führt er eine Marktübersicht relevanter Content-Management-Systeme auf (vgl. Spörrer 2009). CMS sind Softwarelösungen, die der Implementierung von CM in Organisationen dienen. Die Anwendungsbereiche und technischen Merkmale von CMS beschreibt LARS H. EHLERS in seinem Buch *Content Management Anwendungen. Spezifikationen von Internet-Anwendungen auf Basis von Content Management Systemen* (vgl. Ehlers 2003). Der Großteil von CMS zielt dabei auf die Publikation von Contentobjekten im Internet ab. In der Literatur wird diese Form von CM daher Web-Content-Management (WCM) genannt. Unter WCM – oft auch Online-CM genannt (vgl. Bullinger et al. 2001, S. 10) – werden alle Prozesse zusammengefasst, die zur Verwaltung von Contentobjekten auf Websites genutzt werden.

DANIEL FUCHS behandelt in seinem Buch *Web Content Management Systeme – Evaluation anhand eines Praxisbeispiels* die Implementierung von WCM in Unternehmen. Der Fokus des Buchs liegt dabei auf dem sogenannten *Content-Life-Cycle*, der alle Lebensphasen von Contentobjekten von ihrer Produktion bis hin zur Publikation und Archivierung umfasst (vgl. Fuchs 2007). Die momentan am häufigsten genutzten CMS sind WordPress, Joomla! und Drupal (vgl. W3techs, 2014).

Die wachsende Bedeutung von CM für die Unternehmensführung im Informationszeitalter wird bei KOOP ET AL. erläutert. Neben der Definition von Content und CM widmet sich ihr Buch *Erfolgsfaktor Content Management – Vom Webcontent bis zum Knowledge Management* den Prozessen, Abläufen und Anforderungen von CM in Unternehmen (vgl. Koop et al. 2001). CM in Unternehmen wird häufig auch als Enterprise-Content-Management (ECM) bezeichnet. ECM wird in großen Unternehmen eingesetzt, um sämtliche unternehmensinternen Contentquellen zu erschließen. Der Branchenverband AIIM[6] definiert ECM folgendermaßen: »Enterprise Content Management (ECM) is the strategies, methods and tools used to capture, manage, store, preserve, and deliver content and documents related to organizational processes« (AIIM, 2013). Mit ECM befasst sich WOLFGANG RIGGERT. In *ECM – Enterprise Content Management* beschreibt RIGGERT Eigenschaften, Techniken und Bestandteile von ECM. Auch er unterscheidet zwischen fünf Hauptaufgaben, die ECM leisten muss: die Beschaffung, Verwaltung, Bereitstellung, Speicherung und Sicherung von unternehmensrelevanten Informationseinheiten (vgl. Riggert 2009).

Dokumenten- und Wissensmanagement sind jeweils Verfahren, die häufig im Kontext von CM genannt und diskutiert werden. Zu den Hauptaufgaben von Wissensmanagement zählen die Selektion, Bewertung, Aufarbeitung und Verbreitung von Informationen sowie die Anwendung, Bewahrung und Weiterentwicklung von Wissen (vgl. Lehner 2009, S. 32). In ihrem Buch *Wissen managen – Wie Unternehmen ihre wertvollste Ressource optimal nutzen* zeigen PROBST ET AL. die Bausteine von Wissensmanagement und die damit verbunden Aufgaben und Prozesse in Organisationen auf (vgl. Probst et al. 2012). BODROW und BERGMANN

[6] Vgl. hierzu: http://www.aiim.org/ [11.03.2014].

zufolge umfasst Wissensmanagement »sämtliche Instrumente, Verfahren und Maßnahmen, um das in einem Unternehmen vorhandene Wissen transparent zu machen […]« (Bodrow / Bergmann 2003, S. 43). Wissensmanagement hat im Vergleich zu CM einen geringeren Bezug zu Medien und deren Publikation. Wissen wird aus Informationen gewonnen, die auf einer Vielzahl von Daten beruhen. CM bezieht sich – im Gegensatz zu Wissensmanagement – insbesondere auf die Daten- und Informationsebene (vgl. Kapitel 2.2).

Ein weiteres Redaktionssystem, das es vom CM abzugrenzen gilt, ist das Dokumenten-Management. Dokumenten-Management ist die datenbankgestütze Verwaltung von innerbetrieblichen Dokumenten. HARALD KLINGELHÖLLER erläutert in *Dokumentenmanagementsysteme. Handbuch zur Einführung* den gesamten Lebenszyklus von Dokumenten, also deren Erstellung, Verwaltung und Verteilung sowie die Integration in nachgelagerte Prozesse. Ähnlich zum Wissensmanagement verfolgt Dokumenten-Management das Ziel, den Zugriff auf alle relevanten Informationen für Mitarbeiter innerhalb einer Organisation zu gewährleisten (vgl. Klingelhöller 2001). Dokumente werden dabei als Träger von semantisch kohärenten Informationsbeständen in physischer oder digitaler Form betrachtet (vgl. Götzer et al. 2008, S. 2). Während Dokumenten-Management auf die Verwaltung von Dokumenten abzielt, hat CM einen umfassenderen Anspruch indem es die Produktion, Verwaltung und Publikation von Contentobjekten unterstützt (vgl. Hansen / Neumann 2001, S. 452).

Die konkrete Umsetzung von XML-basiertem CM und die damit verbundenen technischen Problemstellungen werden von GUNTHER ROTHFUSS und CHRISTIAN RIED erläutert. In *Content Management mit XML* beschreiben sie umfassend die dafür notwendigen Technologien, Anwendungen und Datenbanksysteme. Neben XML und zugehörigen Substandards liegt der Fokus ihres Buchs auf der Transformationssprache XSLT und der Schema-Erstellung mit XML-Schema (vgl. Rothfuss / Ried 2003). Eine wichtige Grundlage für CM ist die XML-gestützte Modellierung von Daten. Diese wird bei RAINER UND SILKE ECKSTEIN in *XML und Datenmodellierung – XML-Schema und RDF zur Modellierung von Daten und Metadaten einsetzen* detailliert beschrieben. Noch ausführlicher als ROTHFUSS und RIED erläutern sie die konkrete Umsetzung einer XML-basierten Datenmodellierung und alle dafür notwendigen Technologien (vgl. Eckstein / Eckstein 2004).

Aktuelle Literatur zum Thema CM bezieht sich vor allem auf einzelne CMS wie WordPress [7] (vgl. Sauer 2013), TYPO3 [8] (vgl. Hontheim / Herzog-Kienast 2012), Drupal [9] (vgl. Tal 2013), Joomla! [10] (vgl. Marriott 2013) oder Contao [11] (vgl. Weitzel 2014). Ein weiterer, aktueller Trend im Kontext von CM sind semantische Technologien. WOLFGANG MAAS und TOBIAS KOWATSCH erläutern in *Semantic Technologies in Content Management Systems – Trends, Applications and Evaluations* dafür notwendige Verfahren und Anwendungen. Entsprechende

[7] Vgl. hierzu: http://de.wordpress.org/ [12.03.2014].

[8] Vgl. hierzu: http://typo3.org/ [12.03.2014].

[9] Vgl. hierzu: https://drupal.org/ [12.03.2014].

[10] Vgl. hierzu: http://www.joomla.de/ [12.03.2014].

[11] Vgl. hierzu: https://contao.org/[12.03.2014].

Technologien basieren insbesondere auf der Erschließung strukturierter und unstrukturierter Daten im sogenannten *Deep Web* mithilfe semantischer Metadaten und der Nutzung kontextsensitiver Anwendungen bei der Produktion von Contentobjekten (vgl Maass / Kowatsch 2012).

2.2 Daten, Informationen und Wissen

Content ist ein komplexer Begriff. Er kann – je nach Verwendung – verschiedene Bestandteile beinhalten, stellt aber grundsätzlich eine in Form von Daten vorliegende Ansammlung von Informationen dar.

In seiner Arbeit *Management of unstructured Information using semantic Metadata* widmet sich HINNERK BRÜGMANN der Differenzierung von Daten und Informationen. Hierfür verwendet er die die sogenannte *data-information-knowledge-wisdom-pyramid* (DIKWP), die erstmals 1988 von R. L. ACKOFF beschrieben wurde. Die vieldiskutierte[12] DIKWP gilt als grundlegend und kanonisch für die Wissensforschung und wurde in vielfacher Weise in entsprechenden Werken rezipiert (Bernstein 2009, S. 68–69). Die DIKWP besteht aus vier Stufen. Die unterste Stufe beinhaltet Daten. Darauf aufbauend befinden sich auf der zweiten Stufe Informationen, aus denen in der dritten Stufe Wissen gewonnen werden kann. Die letzte Stufe von ACKOFFS Pyramide beschreibt das Konstrukt Weisheit. Weisheit entsteht durch ein tieferes Verständnis, basierend auf der Fähigkeit, über gegebene Fakten urteilen zu können (vgl. Brügmann 2011, S. 6). Weisheit wird im Verlauf der vorliegenden Arbeit nicht im Fokus stehen und soll daher nur der Vollständigkeit halber erwähnt sein.

Daten auf der untersten Ebene der DIKWP sind die Basis für alle darauf aufbauenden Stufen. Sie werden von BRÜGMANN als – in der Regel – digitale, übertragbare, formalisierbare und wieder-interpretierbare Symbole oder Zeichen verstanden, die Informationen repräsentieren. Da Daten in Form von Zeichenketten kodiert sind, gewährleisten sie im Gegensatz zu Informationen keinen direkten Nutzen. Daten müssen somit interpretiert und weiterverarbeitet werden (vgl. Brügmann 2011, S. 4-5). Für die vorliegende Arbeit von großer Bedeutung ist, dass Daten und Informationen strukturiert, semistrukturiert oder unstrukturiert vorliegen können. Durch die Anreicherung mit Metadaten, Markups und anderen Strukturelementen können Informationen strukturiert abgespeichert werden. Insbesondere natürlichsprachige Dokumente können mittels solcher Markierungen nahezu beliebig fein strukturiert werden (vgl. Rothfuss/Ried 2003, S. 61).

Die Einordnung von Textdokumenten als strukturierte oder unstrukturierte Daten fällt allenfalls schwer, da der Strukturierungsgrad stark variiert. Texte weisen fast ausnahmslos eine inhaltliche Struktur auf, die auch in der Form ihren Ausdruck findet. Während die Struktur des Inhalts den Aufbau eines Textes beschreibt, bezeichnet die Struktur des Ausdrucks Gestaltungsmerkmale, wie etwa

[12] Für Kritik an der DIKWP siehe auch: BERNSTEIN 2009, S. 68–70; FRICKÉ 2008, S. 1–6. Beide Autoren werfen dem Modell eine utilitaristische und positivistische Konzeption vor. BERNSTEIN unterstellt Ackoff eine eher betriebswirtschaftliche als akademische Herangehensweise (vgl. Bernstein 2009, S. 68). Frické kritisiert die Implikation einer objektiven Wahrheit in ACKOFFS Datenmodell (vgl. Frické 2008, S. 4).

Hervorhebungen, Einzüge und ähnliche typografischen Elemente. Da Computerprogramme solche textinhärente Strukturmerkmale nicht ohne weiteres auslesen können, müssen diese gekennzeichnet werden. Im Zuge der Digitalisierung wurden daher sogenannte *Markup Languages* entwickelt, mit welchen die Struktur von Texten formal beschrieben werden kann (vgl. Lobin 2010, S. 19; Rothfuss / Ried 2003, S. 11–12). Derart angereicherte Texte werden in der vorliegenden Arbeit daher als semistrukturierte Daten betrachtet, da alle relevanten Strukturinformationen immanent vorhanden sind, der Strukturierungsgrad einer Datenbank jedoch nicht erreicht wird. Der zweite Teil der vorliegenden Masterarbeit widmet sich der der Umsetzung einer solchen Textstrukturierung (vgl. Kapitel 3).

Die Grenze zwischen Daten und Informationen ist fließend. Zwischen beiden Stufen befinden sich die Metadaten. Metadaten – sprich ‚Daten über Daten' – bilden die Grundlage zur Beschreibung von Semantik in Datenstrukturen. Sie können in extrinsischer und intrinsischer Form vorliegen. Extrinsische Metadaten nutzen explizite Attribute um Daten zu beschreiben. Daten können aber auch durch die gezielte Auswahl repräsentativer Wörter beschrieben werden. Solche Wörter werden Deskriptoren genannt. Deskriptoren sind intrinsische Metadaten. Die Nutzung von Metadateninfrastrukturen für die semantische Erschließung von Datenstrukturen setzt standardisierte Regelwerke voraus (vgl. Dengel 2012, 13–15). Beispiele für solche Standards sind der *Dublin Core*-Metadatenstandard,[13] ein aus dem Bibliothekswesen bekanntes Datenformat zur Beschreibung bibliografischer Angaben, oder die von der *Text Encoding Initiative* (TEI) entwickelten Standards zur Kodierung und Edition von Büchern (vgl. Kapitel 2.5, Kapitel 4.4).

Auf den Daten aufbauend, befinden sich Informationen auf der zweiten Stufe der Pyramide. Informationen unterscheiden sich dadurch signifikant von den ihnen zugrundeliegenden Daten, dass sie einen spezifischen Nutzen in einer spezifischen Situation gewährleisten. Informationen haben – im Gegensatz zu Daten – somit eine direkte Problemlösungskompetenz. Sie werden daher oft auch als ‚Daten mit Bedeutung' bezeichnet (vgl. Brügmann 2011, S. 5). ACKOFFS DIKWP impliziert einen Verarbeitungsprozess: »[…] the transformation of data to information is often referred as a process of distillation« (Brügmann 2011, S. 5). Informationen werden also durch einen solchen Prozess in einem entsprechenden Kontext aus Daten gewonnen. Sie entstehen durch den kontextabhängigen Bedarf ihres Nutzers. Die Kommunikationswissenschaftlerin URSULA HASLER ROUMOIS folgert daraus: »Information ist also eine immaterielle und dynamischen Qualität von Daten« (Hasler Roumois 2007, S. 34). In anderen Worten: Informationen liegen in Daten vor, ihre Materialität gewinnen sie erst in Abhängigkeit von Kontext, Aufgabe und verarbeitendem Subjekt. Weder das Bereitstellen noch die Absicht des Senders bestimmen außerdem, welche Informationen aus welchen Daten gewonnen werden, sondern ausschließlich die Verarbeitung des Rezipienten (vgl. Hasler Roumois 2007, S. 34). Dass diese Verarbeitung individuell erfolgt, erklärt die Dynamik möglicher Informationen. Der Wahrnehmungspfad des Subjektes, beginnend bei den wahrnehmbaren Daten bis hin zur Schlussfolgerung und Handlung, ist immer individuell und different (vgl. Güldenberg 1998, S. 358–361).

[13] Vgl. hierzu: http://dublincore.org/metadata-basics/ [10.03.2014].

Für das digitale Informationszeitalter charakteristisch ist, dass Daten und somit auch potentielle Informationen in immer größeren und exponentiell wachsenden Mengen vorliegen, wohingegen die gewinnbringende Nutzung von Daten beständig komplexere Ansätze und Prozesse erfordert (vgl. Bodrow/Bergmann 2003, S. 15–16; Lehner 2009, S. 6–7). Dieser Umstand wird als Informationsflut bezeichnet (vgl. Mehler/Wolf 2005, S. 1). Ansätze, mit dieser Informationsflut umzugehen, beruhen auf der Offenlegung von Informationen mithilfe von Markups. Texte haben daher eine herausragende Bedeutung, da sie Schätzungen zufolge rund 85 % aller relevanten Informationen beinhalten (vgl. Hotho et al. 2005, S. 19).

Die dritte Stufe in ACKOFFS Pyramide beinhaltet Wissen. Wissen resultiert aus dem Informationspotential von Daten durch einen aktiven Denk- und Lernprozess und stellt eine Weiterentwicklung von Informationen dar: »Knowledge, the next layer, further refines information by making ‚possible the transformation of information into instructions« (Bernstein 2009, S. 68). Das verarbeitende Subjekt beurteilt die ihm zugängigen, relevanten Informationen, integriert diese in bereits vorhandenes Wissen und kann dieses Wissen schließlich in Form von Anweisungen weitergeben. Die Verwaltung von Wissen in Organisationen fällt unter den Aufgabenbereich des Wissensmanagements. Wissensmanagement meint dabei die Gesamtheit aller Unternehmensstrategien zur Gestaltung einer organisationalen Wissensbasis. Eine solche Wissensbasis dient als Grundlage zur Lösung unternehmerischer Aufgaben (vgl. Probst et al. 2012, S. 24). Wissen wird im Kontext von Unternehmen zu einer immateriellen und personenabhängigen Ressource und subsumiert das intellektuelle Vermögen von Individuen in Form von Fähigkeiten, Kenntnissen, Einstellungen und Verhalten (vgl. Hagenhoff 2008, S. 88–90). Durch ein tieferes Verständnis kann aus Wissen auf der letzten Stufe der DIKWP schließlich Weisheit resultieren.

ACKOFFS Pyramide zeigt den Unterschied zwischen Daten, Informationen und Wissen auf. Daten, Informationen und Wissen sind differente und aufeinander basierende Einheiten. Im Gegensatz zu Daten entstehen Informationen und Wissen erst durch das verarbeitende Subjekt. Alle drei Stufen stehen außerdem in einer reziproken und prozesshaften Beziehung: Wissen wird aus Daten und Informationen gewonnen und produziert zugleich neue Daten. Für CM spielen insbesondere Daten und Informationen eine wichtige Rolle, da Content in Form von Daten vorliegt und die genuin informative Natur von Content durch den Publikationsprozess offengelegt werden kann.

2.3 Die Begriffe Content und Management

In diesem Kapitel gilt es, die Begriffe Content und Management zu erläutern. Content soll außerdem in das vorgestellte Konzept von Daten, Informationen und Wissen (vgl. Kapitel 2.2) eingeordnet werden.

Die Verwaltung von Content wird mit dem Begriff Management umschrieben. HUNGENBERG bezeichnet Management als ›Einflusshandeln‹ (Hungenberg 2012, S. 20) auf materielle und immaterielle Güter in Unternehmen, mit dem Ziel, unternehmerische Leistung zu erbringen. Die drei wesentlichen Managementfunkti-

onen sind HUNGENBERG zufolge die Planung, Steuerung und Kontrolle unternehmerischer Prozesse (vgl. Hungenberg 2012, S. 20–21). Grundsätzlich werden in der Literatur zwei Ansätze von Management unterschieden. Der *institutionelle* Managementansatz untersucht die mit Weisungsbefugnissen ausgestatteten Personen und Gremien innerhalb einer Organisation. Der *funktionale* Ansatz betrachtet Management als Gesamtheit der Aufgaben und Handlungen, die zur Steuerung eines Leistungsprozesses innerhalb einer Organisation nötig sind. Dabei wird zwischen einem *strategischen* und einem *operativen* Management unterschieden. Das strategische Management analysiert die Stärken und Schwächen des Unternehmens sowie die Chancen und Risiken, die sich aus der jeweiligen Branche und deren Umwelt ergeben, und entwickelt entsprechende Strategien, um die unternehmerische Leistung auf dem Markt zu sichern und zu erweitern. Das nachgelagerte operative Management ist für die Umsetzung dieser Strategien im Tagesgeschäft verantwortlich (vgl. Hagenhoff 2008, S. 20–21). Um CM zu definieren, wird in der vorliegenden Masterarbeit der funktionale Managementbegriff verwendet. CM bezeichnet somit alle Aufgaben und Handlungen, die eingesetzt werden, um Content in einen unternehmerischen Leistungsprozess zu integrieren.

In der Literatur finden sich eine Vielzahl entsprechender Definitionen für Content (vgl. Bullinger et al. 2001, S. 6–8; Koop et al. 2001, S. 8–12; Rothfuss / Ried 2003, S. 10–17; Ehlers 2003, S. 25–29; Kretzschmar / Dreyer 2004, S. 18; Boiko 2005, S. 3–63; Wilhelm 2006, S. 50–51; Fuchs 2007, S. 6–7; Spörrer 2009, S. 5–6).

STEFAN SPÖRRER erklärt Content als ein maschinell erfassbares, drei Ebenen umfassendes Produkt, bestehend aus Inhalt, Layout und Struktur. Content besteht SPÖRRER zufolge aus einer Reihe von Einzelinformationen und deren Erweiterung mit möglichen Metainformationen. Der Inhalt, so SPÖRRER, sei der ‚eigentliche' Content und existiere in Dateien jeglicher Art. Er wird durch ein Layout ergänzt, das aus Formatierungen und Gestaltungselementen besteht. Die Struktur des Contents wiederum unterteilt dessen Inhalt in seine logischen Bestandteile (vgl. Spörrer 2009, S. 117). Die Funktion von Content ist dabei die eines Behälters, der unterschiedliche Komponenten erfassen kann. Zwischen Behälter und Inhalt besteht eine Referenz, so SPÖRRER. Der jeweilige Inhalt muss somit nicht unmittelbar im dazugehörigen Behälter abgespeichert sein, sondern lediglich durch eine Referenz verknüpft sein. Durch diese Referenztechnik kann der Inhalt selbst mehrfach und in unterschiedlichen Kombinationen verwendet werden (vgl. Spörrer 2009, S. 5–6; Thygs 2001, S. 62–63). In anderen Worten: Inhalt muss nicht für jede Verwendung neu produziert werden, sondern kann spezifisch mit neuen Struktur- und Layoutelementen kombiniert werden. Abbildung 1 stellt die Referenzialität von Content schematisch dar.

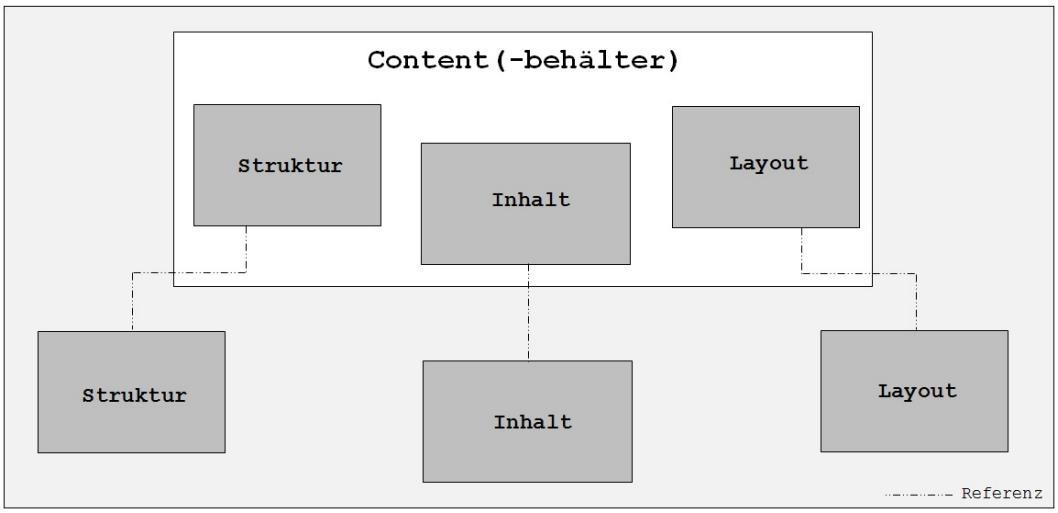

Abbildung 1: Referenzialität von Content in Anlehnung an STEFAN SPÖRRER 2009, S. 5-6

Problematisch an SPÖRRERS Definition von Content ist die unklare Abgrenzung von Content und Inhalt. Deutlicher wird dieser Unterschied bei Gunther Rothfuss und Christian Ried. Sie ordnen Content in das schon bekannte Schema von Daten und Informationen ein. Content ist ROTHFUSS und RIED zufolge eine besondere Form von Daten und Informationen. Seine Definition erfolgt zweckmäßig und wird mit dem Begriff Publikation gleichgesetzt (vgl. Rothfuss / Ried 2003, S. 10–13). Content ist demnach ein Informationsbestand, der an eine Mehrzahl von Adressaten publiziert werden kann. Die verwendeten Daten müssen dabei prinzipiell persistent sein und somit eine längerfristige Verwendung gewährleisten können (vgl. Rothfuss / Ried 2003, S. 19–20). Unter Inhalt verstehen die Autoren dagegen nur eine Form von Content. Um Mehrdeutigkeiten zu verhindern, führen ROTHFUSS und RIED den Begriff *Asset* ein. Assets sind im CM die eigentlichen Medieninhalte. Dabei kann es sich ROTHFUSS und RIED zufolge um Bilddateien, Tonaufnahmen, Graphiken und Videodateien handeln. Auch Textdateien sind demnach als Assets zu bezeichnen. Insbesondere die automatische Weiterverarbeitung und Weiterverwendung solcher Assets erfordert eine Zergliederung und separate Datenhaltung persistenter Bestandteile (vgl. Bullinger et al. 2001, S. 6).

KOOP ET AL. integrieren den Begriff Content in eine Daten-Informations-Wissens-Hierarchie, indem sie Content als Vermittlungsinstanz zwischen Informationen und Wissen beschreiben. Unter Content verstehen die Autoren ein Informationspaket, das mittels eines Mediums weitergegeben werden kann. Content ist somit gleichsam eine Form von Inhalt mit einer spezifischen Publikations- und Vermittlungsabsicht. Content lässt sich gemäß KOOP ET AL. in seine logischen Bestandteile wie etwa Inhalt, Struktur, Medienformat und Layout aufteilen. Unter dem Inhalt verstehen die Autoren die eigentlichen Basisinformationen, wie beispielsweise einen unformatierten Text (vgl. Koop et al. 2001, S. 11–12). Wie auch schon bei ROTHFUSS und RIED zeichnet sich bei KOOP ET AL. ab, dass Content eine vieldimensionale Kombination von Einzelinformationen ist, der mit einer Publikationsintention produziert wird.

LARS H. EHLERS unterscheidet zwischen drei solcher Dimensionen: der syntaktischen, semantischen und pragmatischen Dimension. Die syntaktische Ebene repräsentiert die Kombination und Anordnung von Einzelinformationen. Auf einer semantischen Ebene werden diese Einzelinformationen mit Metadaten versehen, um deren Bedeutung und Struktur offen zu legen. Den Einzelinformationen wird somit eine allgemeine Bedeutung zugewiesen, die sie für Informationssysteme auslesbar machen. Eine solche Interpretation findet schließlich auf einer pragmatischen Ebene statt. Informationssysteme können somit die strukturierten Inhaltswerte kontextabhängig auswerten und nutzen (vgl. Ehlers 2003, S. 26–28).

Content wird in der vorliegenden Masterarbeit als ein aus Daten bestehendes, mehrdimensionales Gebilde von Einzelinformationen gefasst, das mit einer Publikationsabsicht produziert wird. Die einzelnen, medialen Bestandteile werden als Assets bezeichnet, die alle über spezifische Inhalte verfügen. Zentral für Content ist seine Eigenschaft, diese unterschiedlichen und separat abgespeicherten Assets als Container zusammenzufassen, wobei die Verbindung von Speicherort und Container referenzieller Natur ist. In materieller Form liegt Content als Contentobjekt vor. Schließlich stellt Content für seine Rezipienten eine kontext- und plattformabhängige Form von Informationen dar, aus der wiederum Wissen resultieren kann.

2.4 Content-Management in der Anwendung

Wie eingangs erwähnt, fasst Content-Management (CM) die Prozesse zusammen, die für die Verwaltung von Content benötigt werden. Eine solche Verwaltung geschieht über spezifische CM-Anwendungen (CMA), die Teil eines Content-Management-Systems (CMS) sind. CMS können aus einer Reihe von differenten Anwendungen bestehen oder im Fall von integrierenden CMS, alle notwendigen Anwendungen in einer Software vereinen. Beispiele für integrierende CMS sind *WordPress*, *Joomla!* und *Drupal* (vgl. Kapitel 2.1). CMS zeichnen sich grundsätzlich durch eine mehrstufige Architektur aus, die aus einzelnen Modulen besteht. Die modularisierte Architektur von CMS bringt den Vorteil mit sich, dass diese flexibel mit unterschiedlichen Projekttypen und Anforderungen umgehen können (vgl. Christ 2003, S. 29). Idealtypisch existiert ein Datenhaltungssystem – auch *Content Repository* genannt – eine Interaktionsschicht in Form von Benutzeroberflächen und eine vermittelnde Schicht, die den Zugriff auf das Content Repository authentifiziert (vgl. Rothfuss / Ried 2003, S. 111). Auf der Interaktionsschicht befinden sich die verwendeten CM-Anwendungen.

2.4.1 Content-Management als Prozess

CMA erfüllen eine Reihe von Aufgaben, die im Prozess eines Content-Managements auftreten, und unterstützen den gesamten Content-Life-Cycle von Contentobjekten. Dieser umfasst generell deren Produktion, Kontrolle, Freigabe, Publikation und Archivierung (vgl. Christ 2003, S. 104). Die Anwendung von CM erfolgt unter zwei Grundannahmen: Erstens behandeln CMA die Struktur und Gestaltung von Content als getrennte Bestandteile einer Publikation. Struktur- und

Layoutinformationen müssen daher sowohl während des Produktionsprozesses als auch bei der anschließenden Archivierung streng getrennt werden. Zweites erfolgt die Produktion und Verwaltung von Content prozesshaft (vgl. Rothfuss / Ried 2003, S. 89). Der Prozess eines CMs und alle darin anfallenden Aufgaben lassen sich demgemäß in eine Reihe von Schritten aufteilen.

Der erste Schritt eines CMs sieht die Definition eines Konzepts vor. Das Konzept bestimmt über die Bestandteile und den Aufbau der zu produzierenden Contentobjekte. Diese Vorgaben betreffen die Auswahl der Assets und deren Strukturierung. Strukturierungsvorgaben können in Form von Inhaltsmodellen vorliegen. XML unterstützt dafür Dokumententyp-Definitionen und XML-Schema-Definitionen (vgl. Kapitel 4.4). Mithilfe solcher Definitionen können insbesondere Textdokumente beliebig fein strukturiert werden. Außerdem muss definiert werden, wie der eigentliche Produktionsprozess ablaufen soll, welche Ausgangs- und Zielformate vorgesehen sind, welche Personen oder Abteilungen an der Produktion beteiligt sind und mit welchen Berechtigungen sie ausgestattet werden. Vorgaben dieser Art werden zumeist über Workflows organisiert (vgl. Ehlers 2003, S. 106; Rothfuss / Ried 2003, S. 89).

Nachdem ein Konzept definiert wurde, folgt die eigentliche Erstellung des Contents. Das bedeutet, dass Inhaltswerte und Metadaten – den Inhaltsmodellen des vorangegangen Schritts entsprechend – von Redakteuren und Webdesignern produziert oder bezogen werden. Ergebnis einer solchen Erstellung kann beispielsweise ein XML-Dokument sein (vgl. Kapitel 4.3). Nach Abschluss der Erstellung wird der entstandene Content zentral gespeichert, damit er im Verlauf des Prozesses effektiv genutzt werden kann (vgl. Ehlers 2003, S. 107).

In einem dritten Schritt besteht die Möglichkeit, dass unterschiedlicher Content bedarfsorientiert kombiniert wird. Zur Auswahl stehen alle archivierten Contentobjekte und Assets. Beispielsweise können XML-Dokumente und Bilddateien zu neuen Contentobjekten kombiniert werden. Dabei können Endabnehmer miteinbezogen werden. Diese bestimmen in einem solchen Fall selbst, welche Informationen sie erhalten möchten. Benutzerintegrative Strategien bürgen für eine dynamische und kundenorientierte Contentproduktion (vgl. Ehlers 2003, S. 107, S. 111).

Nachdem infolge der Erstellung und Kombination von Content ein vorläufiges Contentobjekt entstanden ist, kann nun eine medienspezifische Formatierung vorgenommen werden. Die Formatierung sieht die Konvertierung in eine für die Publikation benötigte Datenstruktur vor (vgl. Ehlers 2003, S. 108). Dieser Schritt hängt stark vom Ausgangs- und Zielformat ab. XML bietet ein plattformneutrales Ausgangsformat, das mit entsprechenden XML-Stylesheets in die gewünschten Zielformate transformiert werden kann (vgl. Kapitel 4.5, Kapitel 4.6). Struktur, Inhalt und Formatierungsanweisungen werden dabei immer getrennt abgespeichert und erst in der Produktionsphase zusammengeführt. Der dadurch entstehende referenzielle Bezug von Inhalt, Layout und Struktur ermöglicht, dass einzelne Bestandteile von Content zu verschiedenen Contentobjekten zusammengesetzt werden können. Content kann daher mehrfach in verschiedenen Medienformaten publiziert werden (vgl. Kapitel 2.3). Die Publikation ist der fünfte Schritt eines

CM-Prozesses. Die formatierten Contentobjekte werden den Endabnehmern über verschiedene Absatzkanäle zur Verfügung gestellt. Die Publikation kann in gedruckter und digitaler Form erfolgen (vgl. Ehlers 2003, S. 108). CM besitzt somit nach LARS H. EHLERS fünf grundlegende Schritte: die Definition eines Konzepts, die Erstellung und die Kombination des Contents, die Formatierung der Contentobjekte und schließlich deren Publikation.

EHLERS Modell soll für die vorliegende Masterarbeit noch um einen weiteren, aber für CM grundlegenden Schritt erweitert werden. Nach der Publikation muss entschieden werden, ob die im Laufe des Produktionsprozesses entstandenen Contentobjekte archiviert oder gelöscht werden sollen. Im Falle einer Archivierung müssen entsprechende Datenbanken und Datenbankmanagementsysteme ausgewählt und implementiert werden. Das am weitesten verbreite Datenbankmanagementsystem ist die Open-Source-Software *MySQL*.[14] Ein Beispiel für eine XML-basierte Datenbank ist *eXist*.[15] Sie liegt ebenfalls als freie Software vor. Die Abfrage auf einzelne Elemente der Datenbank erfolgt über die XML-Abfragesprache XQuery.[16]

2.4.2 Content-Management-Anwendungen

CM-Anwendungen erfüllen die spezifischen Anforderungen, die im Zuge der erläuterten sechs Schritte eines Content-Managements auftreten. So können für die Vorgabe und Organisation von Workflows entsprechende Softwarelösungen genutzt werden. Die Eingabe von Inhaltswerten kann mithilfe von XML-Editoren geschehen. Für die Formatierung und die Publikationen existiert ebenfalls eine Vielzahl freier und proprietärer Anwendungen. Produzierte Contentobjekte können mithilfe gängiger Datenbankmanagementsysteme archiviert werden. Neben Einzelanwendungen existiert auch eine Reihe von integrierenden CMS, die alle erforderlichen CM-Anwendungen in sich vereinen.

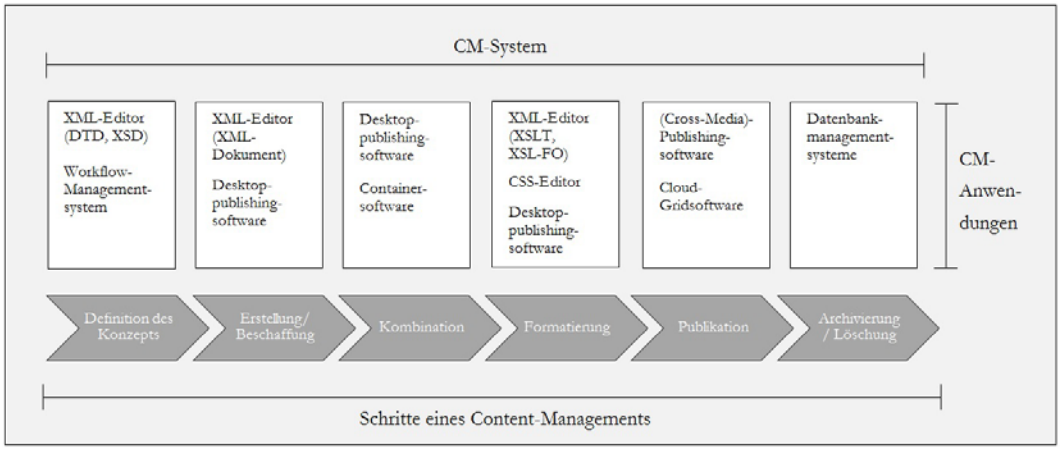

Abbildung 2: Content-Management-Anwendungen

[14] Vgl. hierzu: http://www.mysql.com/ [15.03.2014].

[15] Vgl. hierzu: http://exist-db.org/exist/apps/homepage/index.html [15.03.2014].

[16] Vgl. hierzu: http://www.w3.org/standards/xml/query [15.03.2014].

Abbildung 2 zeigt die erforderlichen Schritte eines möglichen Content-Managements. Die Abbildung stellt insofern einen Idealtypus dar, da die konkrete Umsetzung von CM sehr von ihrem Kontext – sprich von der ausführenden Organisation selbst, den angestrebten Zielen, den zu bearbeitenden Contentobjekten und schließlich von den verwendeten CM-Anwendungen – abhängt. Den jeweiligen Schritten wurden mögliche CM-Anwendungen hinzugefügt. Alle Anwendungen ergeben in ihrer Gesamtheit ein aufeinander abgestimmtes CMS.

2.5 Ein Beispiel aus der Praxis: TextGrid

TextGrid [17] ist eine vom *Bundesministerium für Bildung und Forschung* geförderte, virtuelle Forschungsumgebung, die 2006 ins Leben gerufen wurde. Im Mai 2012 wurde die Version 2.0. veröffentlicht, die laufend weiterentwickelt wird. TextGrid richtet sich an Fachwissenschaftler, Entwickler und Forschungsprojekte im Bereich der Geistes- und Humanwissenschaften. Das Projekt verfolgt das Ziel, ein auf Open-Source-Technologien basierendes, digitales Ökosystem für die Bearbeitung wissenschaftlicher Datenbestände zu ermöglichen: »The primary mission of TextGrid is to provide a virtual research environment (VRE) for humanities scholars in which various tools and services are available for the creation, analysis, editing, and publication of texts and images« (TextGrid 2012, S. 9). Unter digitalen Ökosystemen versteht man in der Informatik virtuelle und serverbasierte, zumeist offene sowie selbstorganisierte Umgebungen, die aus einer Vielzahl selbständiger Nutzer bestehen (vgl. Küster et al. 2009, S. 185).

Der Anspruch von TextGrid besteht darin, die heterogenen und aus unterschiedlichen Contentobjekten bestehenden Datenbestände aller beteiligten Organisationen in einem virtuellen und homogenen Textkorpus zu vereinen und den Zugriff darauf zu vereinheitlichen. In erster Linie handelt es sich bei diesen Contentobjekten um Textdateien und Digitalisate. Mittels aktueller Informationstechniken kann der daraus entstandene Textkorpus analysiert und editiert werden. Dabei werden sehr große Datenmengen mit Metadaten angereichert und archiviert (vgl. Küster et al. 2009, S. 185).

Alle verwendeten Technologien basieren auf offenen Standards, um den Austausch zwischen unterschiedlichen Entwicklungsumgebungen zu gewährleisten. Die von TextGrid zur Verfügung gestellte Plattform hat den Anspruch, den gesamten Workflow – von der Generierung der Primärdaten bis zu deren Publikation – zu vereinheitlichen. Um diesem Anspruch gerecht zu werden, verwendet TextGrid das XML-basierte Datenbanksystem *eXist*. XML ist eine Auszeichnungssprache, die zur Beschreibung logischer Strukturen und Hierarchien in Textdateien genutzt wird (vgl. Kapitel 3). Die Verwendung von XML ermöglicht einen zentralen Anspruch von TextGrid: die Verwendung des von der *Text Encoding Initiative* (TEI) festgelegten Markup-Sets für die texttechnologische Erschließung von Informationsstrukturen in Texten (vgl. TextGrid 2014a, Das Projekt).

[17] Vgl. hierzu: http://www.textgrid.de/home/ [14.03.2014].

Die TEI ist eine 1987 gegründete Organisation, mit dem Ziel, Grundlagen und Standards für die computergestützte Kodierung vornehmlich geisteswissenschaftlicher Werke und textkritischer Editionen zu entwickeln. Die Standards liegen in Form von komplexen Dokumententyp-Definitionen und XML-Schema-Definitionen vor. Diese Inhaltsmodelle legen fest, welche Bestandteile in welchem Kontext für die Anreicherung wissenschaftlicher Texte verwendet werden können. Für alle erlaubten Bestandteile sind von der TEI entsprechende XML-Tags vorgesehen. Diese Tags werden in Form von Modulen organisiert, die je nach Bedarf kombiniert werden können (vgl. TEI Consortium 2014, TEI History).

TextGrid besteht aus einer sogenannten *Grid*-Konstruktion. Grid bezeichnet eine Infrastruktur, die aus einem Verbund verschiedener, lose-gekoppelter Recheneinheiten besteht, deren Kapazitäten über eine vermittelnde Software gebündelt werden. Das Ziel des Zusammenschlusses ist die gemeinsame Nutzung von Rechenleistung, Instrumenten und Datenbanken (vgl. Fey 2010, S. 5). Die Architektur von TextGrid teilt sich in drei Schichten auf.

Auf der obersten Ebene der Architektur befindet sich die Benutzeroberfläche. Die über die Benutzeroberfläche erreichbaren Services sind auf der mittleren Schicht angesiedelt. Die Trennung dieser Ebenen ermöglicht die individuelle Konfiguration der Benutzeroberfläche. Die grundlegenden Services beinhalten unter anderem einen XML-Editor für die Produktion von XML-Dokumenten, einen Tokenizer, der für die Aufteilung von Textdokumenten in ihre einzelnen Bestandteile verantwortlich ist sowie einen Lemmatizer für die Analyse dieser Bestandteile. Ein wichtiger Service ist zudem die Workflow-Engine, die mehrere Dienste zu einem komplexen Arbeitsablauf kombinieren kann. Die unterste Ebene wird als *Middleware* bezeichnet. Auf dieser Ebene werden die Archive und Datenbanken von TextGrid abgespeichert. Die Middleware ermöglicht die Einbindung der Services in den Grid. Die Kommunikation zwischen den Ebenen erfolgt über die Netzwerkprotokolle *SOAP*[18] und *WSDL*.[19] (Gietz et al. 2006, S. 137–139). Beide Protokolle basieren auf XML und wurden vom *World Wide Web Consortium* (W3C) entwickelt.

Für die Nutzer von TextGrid sind insbesondere zwei Hauptkomponenten wichtig: das *TextGrid Laboratory*[20] (TextGridLab) und das *TextGrid Repository*[21] (TextGridRep). TextGridRep ist eine digitale Bibliothek von Texten aus der Frühzeit des Buchdrucks bis ins 20. Jahrhundert. Sie enthält alle wichtigen Texte des deutschen Literaturkanons mit gemeinfreier Lizenz. Das besondere an dieser Bibliothek ist, dass die Textdokumente mit TEI-konformen XML-Tags angereichert sind. Darüber hinaus wurden alle bibliografischen Angaben in Form von Metadaten hinzugefügt. Die einzelnen Texte des im Zuge des Projekts entstandenen Textkorpus stehen für die Endnutzer als XML-Dateien, als Zip-Archive sowie als E-Books im EPUB-Format zur Nutzung bereit (vgl. TextGrid 2014b, Die Digitale Bibliothek bei TextGrid).

[18] Vgl. hierzu: http://www.w3.org/TR/soap/ [15.03.2014].

[19] Vgl. hierzu: http://www.w3.org/TR/wsdl20/ [15.03.2014].

[20] Vgl. hierzu: https://www.textgrid.de/registrierungdownload/download-und-installation/ [15.03.2014].

[21] Vgl. hierzu: http://www.textgridrep.de/repository.html [15.03.2014].

Die zweite Hauptkomponente von TextGrid ist die Clientsoftware TextGridLab. TextGridLab ist eine *Eclipse*[22]-basierte Benutzeroberfläche, die den dezentralen Zugang zur virtuellen Forschungsumgebung TextGrid ermöglicht. Sie ist auf der obersten der drei Ebenen der TextGrid-Architektur angesiedelt und kombiniert die gewünschten Dienste. TextGridLab ermöglicht es, Projekte offline zu bearbeiten und die Daten später mit dem Grid zu synchronisieren (vgl. Küster et al. 2009, S. 187). Die Benutzeroberfläche vereint alle für die Produktion, Bearbeitung und Archivierung von Contentobjekten benötigten Werkzeuge. Hauptbestandteil der Benutzeroberfläche ist der für die Bearbeitung und Produktion von TEI-konformen Textdokumenten erforderliche XML-Editor. Mithilfe dieses Editors können XML-Dokumente erstellt werden und mit unterschiedlichen TEI-konformen Dokumententyp-Definitionen und XML-Schemata verknüpft werden. Zusätzlich verfügt das Programm über einen eigenen Metadaten-Editor. Außerdem bietet TextGridLab eine Projekt- und Workflow-Verwaltung an, über die Arbeitsabläufe, die Rechte aller beteiligten Personen sowie die Contentobjekte selbst verwaltet werden können (vgl. Küster et al. 2009, S. 187–188).

TextGridLab entspricht der Definition eines XML-basierten Content-Management-Systems. Mit dieser Software können unterschiedliche Assets wie beispielsweise Textdokumente und Digitalisate zu Contentobjekten verknüpft und zentral abgespeichert und verwaltet werden. Die Verwendung von TEI-konformem Markup gewährleistet, dass die produzierten Contentobjekte dem de-facto-Standard geistes- und humanwissenschaftlicher Textkodierung entsprechen. Insbesondere unterstützt TextGridLab eine am gesamten Content-Life-Cycle orientierte Verarbeitung der in der virtuellen Umgebung von TextGrid produzierten Contentobjekte. So kann Content mit TextGridLab produziert, bearbeitet, publiziert und archiviert werden. Schließlich können alle an einem Projekt beteiligten Personen über die Clientsoftware dezentral auf die zu bearbeitenden Contentobjekte zugreifen.

[22] Vgl. hierzu: http://www.eclipse.org/org/ [15.03.2014].

3 Theoretische Fundierung II: eXtensible Markup Language (XML)

Im zweiten Theorieteil sollen die für den praktischen Teil der Masterarbeit benötigten Grundlagen der *eXtensible Markup Language* (XML) erläutert werden. Hierzu werden die Bausteine und Eigenheiten von XML vorgestellt. Anschließend sollen weitere XML-Subsprachen und die für ein XML-basiertes Content-Management benötigten Schlüsseltechnologien dargelegt werden. Zur Veranschaulichung dienen dabei insbesondere repräsentative Programmausdrücke, sogenannte Listings. Anhand dieser Listings können die einzelnen Elemente von XML erläutert werden. Im Fließtext werden Programmausdrücke außerdem entsprechend markiert.

XML bezeichnet eine Auszeichnungssprache zur Beschreibung logischer Strukturen und Hierarchien in Texten. XML wird daher auch als Metasprache bezeichnet (vgl. Erlenkötter 2012, S. 11). XML basiert auf der *Standard Generalized Markup Language* (SGML), die als internationaler Standard im Jahr 1986 verabschiedet wurde, sich allerdings für den Alltagsgebrauch als zu komplex erwies (vgl. Vonhoegen 2011, S. 28). XML wurde vom *World Wide Web Consortium* (W3C) entwickelt und in der Erstfassung am 10. Februar 1998 veröffentlicht. Die aktuelle Empfehlung für XML stammt vom 26. November 2008.[23]

Zentral für Bezeichnungssprachen ist, dass Texte als Zeichen und ihrem Aufbau entsprechend als semistrukturierte Daten behandelt werden (vgl. Kapitel 2.2). Mittels präziser und algorithmisierbarer Verfahren werden Struktur und Aufbau des Inhalts von Texten erfasst und zur Weiterverarbeitung genutzt. Dabei zeichnet sich XML – im Vergleich zur *Hypertext Markup Language* (HTML) – insbesondere durch ihren generischer Charakter aus, der es ermöglicht, die Struktur des Textes formal zu beschreiben, wohingegen HTML über eine Reihe fest definierter Elemente verfügt (vgl. Lobin 2010, S. 17–18).

Die Logik von XML ist simpel: XML definiert eine Syntax, die es erlaubt, semistrukturierte Daten mit einfachen Bezeichnungen – sogenannte Tags – anzureichern. Sowohl das dabei verwendete Vokabular als auch die Grammatik sind erweiterbar: Benutzer können je nach Bedarf die Vorgaben zum Aufbau einer Textdatei entwickeln und die entsprechenden Bezeichnungen definieren oder aber auf gängige und bereits definierte Vokabulare zurückgreifen. Die große Stärke von XML liegt darin, dass die erfassten Strukturen beliebig tief verschachtelt werden können, sodass – in Kombination mit der Erweiterbarkeit von XML – komplexeste Hierarchien verarbeitet werden können.

XML bietet sich insbesondere für CM an. Diesen Umstand verdankt XML der Tatsache, dass Stylesheets aus der XML-Familie die Möglichkeit bieten, Struktur, Inhalt und Darstellung eines Dokuments streng getrennt zu verwalten. Diese Teilung des Contents in seine Bestandteile erweist sich als basale Eigenschaft von CM (vgl. Kapitel 2.3). Zudem erlaubt XML eine beliebig fein strukturierte Speicherung von Content und darüber hinaus die Verwaltung aller nötigen Meta-Informationen. Schließlich reicht zur Darstellung von XML-Dateien grundsätzlich

[23] Vgl. hierzu: W3C Recommendation XML 1.0: http://www.w3.org/TR/REC-xml/ [03.02.2014].

ein einfacher Texteditor aus, da die angereicherten Daten einfach in Textform abgelegt werden. In der Praxis greift man allerdings auf komfortablere XML-Editoren[24] zurück (vgl. Vonhoegen 2011, S. 42–43).

Die Zahl der Programme, die zur Erzeugung, Bearbeitung oder automatischen Weiterverarbeitung von XML-Dokumenten genutzt werden, nimmt in allen Softwarebereichen zu. »Zwar wurde XML zunächst als eine ‚digitale Repräsentation von Dokumenten entworfen', […] inzwischen aber hat XML längst die Aufmerksamkeit von Datenbankexperten, B2B- oder E-Commerce-Entwicklern auf sich gezogen« (Vonhoegen 2011, S. 30–31). XML hat sich dadurch zur globalen Sprache für den Datenaustausch etabliert. Große Anbieter wie Microsoft,[25] IBM[26] und Oracle[27] unterstützen und nutzen XML umfassend für ihre Anwendungen (vgl. Vonhoegen 2011, S. 28–32).

3.1 Die Bausteine von XML

Mit XML werden laut den W3C-Empfehlungen Datenobjekte beschrieben, die XML-Dokumente genannt werden. Grundvoraussetzung für ein XML-Dokument ist dessen Wohlgeformtheit im Sinne der XML-Syntax. Um wohlgeformt (engl. *well-formed*) zu sein, muss ein XML-Dokument daher einer Reihe von Regeln entsprechen, die in diesem Abschnitt vorgestellt werden sollen.

Wie alle Daten, bestehen auch XML-Dokumente zunächst aus einer Zeichenkette: »Physikalisch bestehen XML-Dokumente aus Speichereinheiten. [...] Ein XML-Prozessor startet seine Arbeit mit dem ersten Zeichen und arbeitet sich bis zum letzten Zeichen durch« (Vonhoegen 2001, S. 48). Ein wichtiges Modul innerhalb solcher Prozessoren sind sogenannte XML-Parser (von engl. *to parse*, ›analysieren‹). XML-Parser analysieren XML-Dokumente und zerlegen sie in separate Textstücke, sogenannte Entitäten. Abgesehen von der Dokument-Entität, die eine Art übergeordnete und alle weiteren Entitäten umfassende Meta-Entität darstellt, enthalten alle Entitäten Inhalt und lassen sich mit einem eindeutigen Namen identifizieren. Zudem lassen sich Entitäten in zwei Gruppen einteilen: *parsed* und *unparsed data*. Alle Zeichendaten und Markups, darunter auch Tags, werden vom Parser eingelesen und analysiert. Sie stellen daher *parsed data* dar. Unter Markup wird eine Reihe von Anweisungen verstanden, die sich dadurch auszeichnen, dass sie in spitzen Klammern stehen oder in Einzelfällen mit einem Ampersand-Zeichen beginnen. *Unparsed data* hingegen werden genutzt, um Nicht-Text-Objekte wie etwa Bilder oder Video-Dateien, die vom Parser nicht analysiert werden können, oder Textstücke, die nicht analysiert werden sollen, in

[24] Vgl. hierzu: Altova XML Spy: http://www.altova.com/de/xmlspy.html [03.02.2014]; Microsoft XML Notepad: http://www.microsoft.com/en-us/download/details.aspx?id=7973 [03.02.2014]; Liquid Technologies: http://www.liquid-technologies.com/xml-studio.aspx [03.02.2014].

[25] Vgl. hierzu: Microsoft MSXML 4.0: http://www.microsoft.com/de-de/download/details.aspx?id=19662 [03.02.2014].

[26] Vgl. hierzu: IBM XML downloads and products: http://www.ibm.com/developerworks/xml/find/downloads/[03.02.2014].

[27] Vgl. hierzu: Oracle XML Database: http://www.oracle.com/technetwork/database/database-technologies/xmldb/overview/index.html [03.02.2014].

ein XML-Dokument einzubinden. Die entsprechende Syntax sieht folgendermaßen aus: `<![CDATA[`"Dieser Text wird nicht geparst"`]]>`.

Neben der – durch erwähnte Entitäten bestimmten – physikalischen Struktur besitzt jedes XML-Dokument eine logische Struktur. Diese besteht aus einem Baum von Informationsträgern, der den Namen *XML Information Set* (Infoset) trägt. Die W3C-Empfehlung definiert insgesamt elf Typen von Informationsträgern. Die wichtigsten Träger stellen die Elemente dar. Jedes XML-Dokument besteht aus einer logischen Verschachtelung von Elementen, die sich in einem Baumdiagramm anordnen lassen und in Beziehung zueinander stehen. Daneben enthält das Dokument weitere Informationsträger, beispielsweise Attribute, Kommentare, Verarbeitungsanweisungen (von engl. *processing instructions*) und Zeichenreferenzen (vgl. Vonhoegen 2011, S. 47–49, S. 59–61; Erlenkötter 2012, S. 20–25).

Die wichtigsten Bausteine eines XML- Dokuments sind dessen Elemente und Attribute. Das folgende Beispiel zeigt ein Element `verfasser`, das die Zeichendaten ‚Franz Kafka' enthält: `<verfasser>`Franz Kafka`</verfasser>`. Jedes Element beginnt mit einem Start-Tag und endet mit einem End-Tag, das mittels eines Schrägstrichs gekennzeichnet wird. Beide Tags werden auch als Markup bezeichnet. Sowohl die Tags als auch die Zeichendaten werden von einem XML-Parser eingelesen und sind daher *parsed data*. Es ist üblich, dass der Inhalt eines Elements durch dessen Namen beschrieben wird. Element-Tags werden daher auch als Metainformationen bezeichnet. Elemente können leer sein. In diesem Fall lautet die Syntax: `<keininhalt/>`. Der Schrägstrich aus dem End-Tag wird also in den Start-Tag vorgezogen.

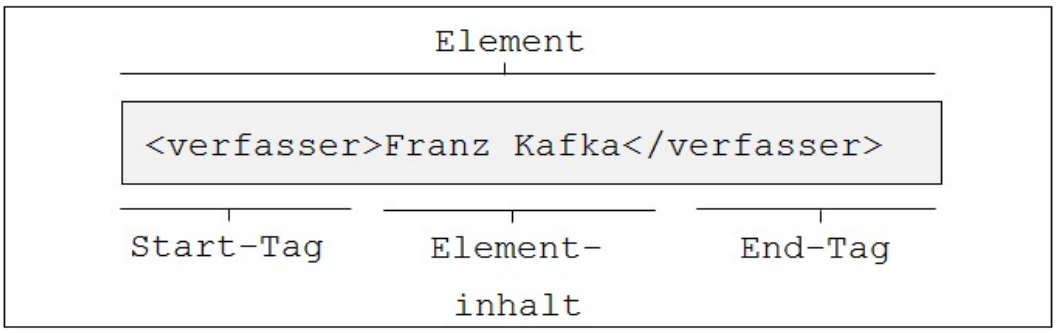

Abbildung 3: XML Elementsyntax nach VONHOEGEN 2011, S. 55

Alle Elemente in einem XML-Dokument können mit beliebig vielen Attributen erweitert werden. Sie dienen dazu, bestimmte Eigenschaften eines Elementes zu beschreiben oder es mit einem Identifizierungsmerkmal zu versehen. Attribute werden immer innerhalb des Start-Tags eines Elementes platziert und besitzen einen Attributwert. Die Syntax für Attribute sieht somit aus wie folgt: `<elementname attributname="attributwert"/>`.

In einem wohlgeformten XML-Dokument müssen zudem alle Elemente korrekt verschachtelt sein. Das bedeutet, dass jedes Element – abgesehen vom Wurzelelement – innerhalb eines Elternelemtents geöffnet und geschlossen werden muss. Eine überkreuzte Verschachtelung wie folgt entspricht nicht der XML-

Syntax: `<element1><element2></element1></element2>`. Damit ein XML-Dokument nicht nur wohlgeformt, sondern auch gültig (engl. *valid*) ist, müssen außerdem alle im XML-Dokument vorkommenden Elemente und Attribute in einer Dokumententyp-Definition (DTD) oder einer XML-Schema-Definition (XSD) definiert werden.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>      <!-- 1 -->

<?xml-stylesheet href="kafka_transform.xsl" type="text/xsl" ?>     <!-- 2 -->

<!DOCTYPE programm [...]>                          <!-- 3 -->

<!-- kafka verwandlung -->                         <!-- 4 -->

<programm>                                  <!-- 5 -->

  . . .

   <buch>                                 <!-- 6 -->

      <verfasser>Franz Kafka</verfasser>

      <titel>Die Verwandlung</titel>

      <kapitel NR="1">Als Gregor Samsa...</kapitel>        <!-- 7 -->

   </buch>

</programm>                                   <!-- 8 -->
```

Listing 1: XML Grundaufbau

Listing 1 zeigt einen Ausschnitt aus einem XML-Dokument. Anhand des Beispiels lassen sich sowohl Aufbau als auch die Bausteile eines XML-Dokuments erläutern.

(1) Jedes XML-Dokument beginnt in der Regel mit einem Prolog. Ein Prolog verfügt über eine sogenannte XML-Deklaration. Diese beginnt zwingend mit `<?xml` und endet mit `?>`. Innerhalb der Deklaration wird die verwendete Version benannt. Zum Zeitpunkt des Verfassens der vorliegenden Arbeit handelt es sich dabei um die Version `"1.0"`. Das zweite Attribut trägt den Namen `encoding` und gibt die Anweisung darüber, mit welcher Zeichenkodierung das XML-Dokument abgespeichert werden soll. `"ISO-8859-1"` wird für westeuropäische Sprachen genutzt, da mit dieser Kodierung auch Zeichen, die oberhalb des ASCII-Zeichensatzes liegen – wie etwa Umlaute – vom Parser interpretiert werden können. Das letzte Attribut `standalone` gibt an, ob sich das XML-Dokument auf eine externe DTD bezieht oder nicht. Der Wert `"yes"` bedeutet, dass das XML-Dokument alle Informationen in sich trägt, die für eine Validierung notwendig sind. Die Attribute `encoding` und `standalone` sind optional. Wenn sie genannt werden, müssen sie allerdings in der gezeigten Reihenfolge aufgeführt werden.

(2) Sollen für die weitere Verarbeitung zusätzliche Anweisungen herangezogen werden, ist das über sogenannte Verarbeitungsanweisungen möglich. Eine verarbeitende Software erhält somit beispielsweise Informationen über die Darstellung oder die Transformation des XML- Dokuments. Eine solche Instruktion muss immer der XML-Deklaration folgen. Sie beginnt mit `<?` direkt gefolgt vom Namen, der in diesem Fall `xml-stylesheet` lautet, und endet mit `?>`. Das Attribut `href` gibt den URI der Zieldatei an, in welcher sich die Anweisungen befinden. Das Attribut `type` wiederum trägt einen anwendungsspezifischen Wert – in diesem Fall `"text/xsl"` – und teilt dem XML-Prozessor mit, um was für eine Anwendung es sich handelt. Im aufgeführten Beispiel wird auf ein XSL-Stylesheet verwiesen (vgl. Kapitel 4.5).

(3) Da das Attribut `standalone` in der XML-Deklaration den Wert `"yes"` hat, besteht der dritte und letzte Teil des Prologs aus einer internen DTD. Diese beginnt immer mit `<!DOCTYPE`, gefolgt vom Wurzelelement, das in diesem Fall den Namen `programm` trägt. In den darauf folgenden eckigen Klammern werden alle Regeln für das XML-Dokument definiert, die ein Prozessor benötigt um dessen Gültigkeit zu überprüfen. Eine weitere Möglichkeit besteht darin, eine XML-Schema-Definition (XSD) einzubinden. Wird eine XSD genutzt, entfällt das Attribut `standalone` sowie die Regeldefinition mittels einer DTD (vgl. Kapitel 4.4).

(4) Kommentare werden genutzt, um XML-Dokumente mit Informationen anzureichern, die in der formatierten Ausgabe nicht sichtbar sind. Sie richten sich daher an XML-Designer und dienen oftmals der Explikation der Struktur des Dokuments. Kommentare beginnen mit `<!--` und enden mit `-->`.

(5) Hinter dem Prolog beginnen die eigentlichen XML-Daten in Form von Elementen und Attributen. Das erste Element ist immer das Wurzelelement, oft auch Dokumentelement genannt. Dieses trägt alle darauf folgenden Elemente in sich. Das Wurzelelement beginnt mit dem Start-Tag `<programm>`. Jedes darauf folgende Element besitzt daher genau ein Elternelement. Das Wurzelelement kann wiederum für jedes seiner Kindelemente weitere Nachkommen zum Inhalt haben. Die Abfolge aller Elemente wird Dokumentreihenfolge genannt und spielt insbesondere für die Pfadbezeichnungssprache XPath eine wichtige Rolle (vgl. Kapitel 3.2.3).

(6) Das Element `buch` hat in diesem Beispiel genau drei Kindelemente: `verfasser`, `titel`, `kapitel`. Diese Elemente sind Kindelemente des Elements `buch` und Nachkommen des Elements `programm`.

(7) Das Element `kapitel` besitzt zwischen dem Start- und End-Tag einfachen Text in Form von Zeichendaten. Vor allem aber hebt es sich von den anderen Elementen dadurch ab, dass es das Attribut `NR` besitzt. In diesem Fall dient das Attribut als Identifizierungsmerkmal, um das Element `kapitel` von seinen Geschwisterelementen zu unterscheiden.

(8) Um wohlgeformt zu sein, muss jedes nicht leere Element über einen Start- und einen End-Tag verfügen. Das Wurzelelement `programm` umschließt alle Elemente und muss daher an dieser Stelle geschlossen werden.

Listing 1 stellt den Aufbau eines XML- Dokuments und dessen wichtigste Bestandteile vor. Demnach beginnt ein XML-Dokument mit einem Prolog, bestehend aus der XML-Deklaration sowie optionalen Verarbeitungsanweisungen oder einer optionalen DTD. Dem Prolog folgen die eigentlichen XML-Daten in Form von korrekt verschachtelten Elementen. Die wichtigsten Bestandteile eines XML-Dokuments sind die Elemente, gefolgt von deren Attributen. Abbildung 4 zeigt den schematischen Aufbau eines XML- Dokuments.

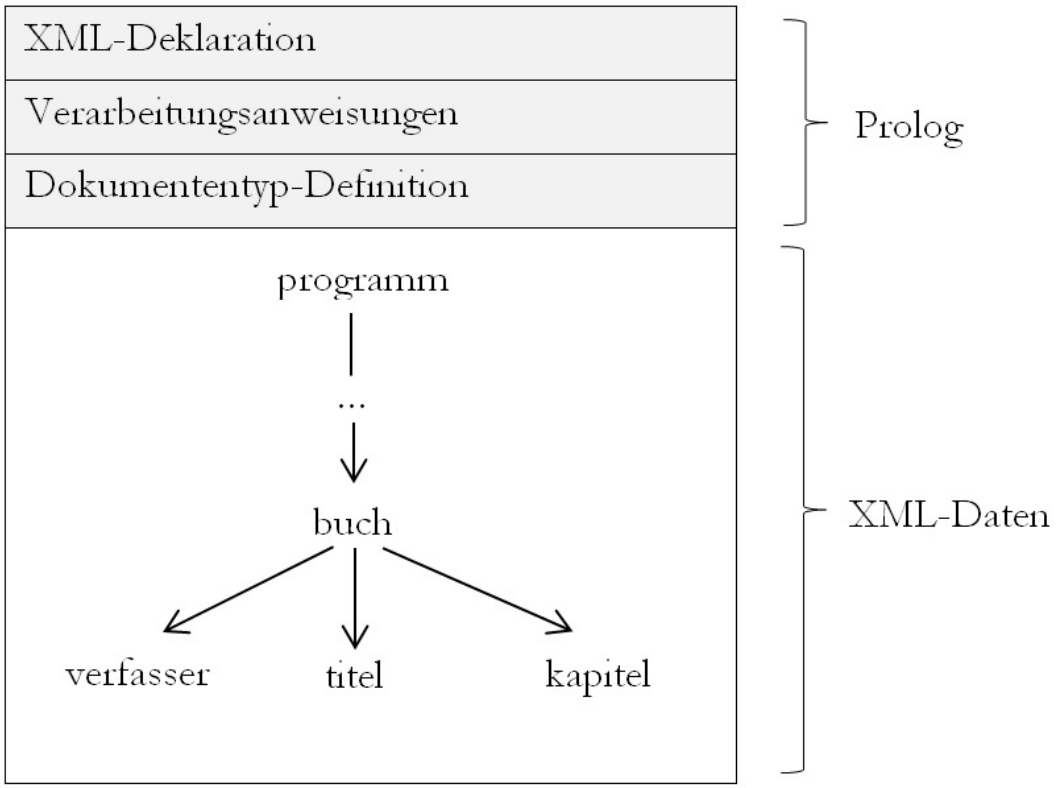

Abbildung 4: Schematische Abbildung eines XML-Dokuments nach ECKSTEIN / ECKSTEIN 2004, S.19

Damit ein XML-Dokument wohlgeformt ist, müssen die in diesem Kapitel vorgestellten Regeln erfüllt sein. Die wichtigsten Regeln lauten zusammenfassend: Jedes wohlgeformte XML-Dokument benötigt einen Prolog mit einer XML-Deklaration. Außerdem muss ein XML-Dokument im Bereich der XML-Daten über mindestens ein Element verfügen. Im Normalfall besitzt ein XML-Dokument eine Vielzahl von Elementen. In diesem Fall müssen diese korrekt verschachtelt sein, das heißt, dass jedes Element nur innerhalb seines Elternelementes existieren darf. Schließlich muss genau ein Wurzelelement existieren. Zudem verfügt jedes gültige XML-Dokument zusätzlich über eine DTD oder XSD, die alle Bestandteile und die Struktur des Dokuments definiert. DTD und XSD fungieren daher als eine Art generische und spezifische Grammatik für ein valides XML-Dokument.

3.2 Weitere Schlüsseltechnologien

XML ist eine lebendige, sich stets in Entwicklung befindliche Sprache. VONHOEGEN zeichnet eine passende Analogie:

> Werden Technologien, die sich unter dem Namen XML gruppieren lassen, mit einem Gebäudekomplex verglichen, so hat es sich dabei einige Jahre lang eher um eine Baustelle gehandelt [...]. Inzwischen aber sind die Hauptgebäude hochgezogen, der Kern des Komplexes steht [...] (Vonhoegen 2011, S. 19).

XML befindet sich jedoch nicht nur in einer ständigen Entwicklung. XML stellt den Kernbereich einer ganzen Sprachfamilie dar, deren Bestandteile im Verlauf der vorliegenden Masterarbeit eine wichtige Rolle spielen und daher kurz vorgestellt werden sollen. Die genaue Funktion dieser Subsprachen wird in den folgenden Kapiteln näher erläutert (vgl. Kapitel 4).

Grundlegend für die XML-Familie ist die bereits erwähnte Kernspezifikation für XML 1.0. Das W3C verabschiedete zusätzlich im Jahr 2004 die erweiterte Version 1.1,[28] die allerdings mit den gängigen XML-Prozessoren nicht kompatibel ist und daher in der Praxis kaum eine Rolle spielt (vgl. Vonhoegen 2011, S. 34). Zur Kernspezifikation von XML gehören das bereits erwähnte Infoset (vgl. Kapitel 3.1) sowie die XML-Namensräume (engl. *namespaces*). Namensräume spielen eine wichtige Rolle in XML. Sie werden von der Empfehlung *Namespaces in XML*[29] geregelt. Die aktuelle, dritte Fassung liegt seit Dezember 2009 vor. Die Wichtigkeit von Namensräumen liegt in der Erweiterbarkeit des XML-Vokabulars begründet. Dadurch, dass nahezu beliebige Elementbezeichnungen deklariert werden können, besteht bei der Auswertung verschiedener XML-Dokumente die Gefahr von Mehrdeutigkeiten und Überschneidungen. Um diese zu vermeiden, können Elemente und Attribute bestimmten Namensräumen zugeordnet werden. Elemente und Attribute, die einem Namensraum zugeordnet werden können, haben qualifizierte Namen, sogenannte *QNames*. In Kapitel 4.4.3 wird das Konzept der Namensräume näher erläutert.

3.2.1 Inhaltsmodelle mit DTD und XSD

Namensräume und deren Bestandteile müssen in einem Regelwerk definiert werden. Das geschieht mithilfe einer XML-Schema-Definition (XSD). Eine XSD definiert jedoch nicht nur die Namensräume, die in einem XML-Dokument verwendet werden können. Mithilfe einer XSD lassen sich auch die Inhalte eines XML- Dokuments und deren Strukturierung festlegen. Für die aktuelle Version 1.1 liegen seit April 2012 zwei Empfehlungen vor.[30] Im Gegensatz zur Dokumententyp-Definition (DTD), die nicht Teil der XML-Sprachfamilie ist, sondern einer

[28] Vgl. hierzu: http://www.w3.org/TR/xml11/ [19.02.2014].

[29] Vgl. hierzu: http://www.w3.org/TR/REC-xml-names/ [21.02.2014].

[30] Vgl. hierzu: http://www.w3.org/TR/xmlschema11-1/ [21.02.2014] und http://www.w3.org/TR/xmlschema11-2/ [21.02.2014].

eigenen Syntax folgt, sind XSD in XML formuliert und können entsprechend auch auf ihre Wohlgeformtheit geprüft werden.

Zentral für eine XSD ist die Unterscheidung zwischen einfachen und komplexen Datentypen. Datentypen – auch XML-Schema-Instanzen genannt – sind abstrakte XSD-Bausteine, die konkrete Elemente und Inhalte eines XML- Dokuments repräsentieren. Einfache Typen sind unstrukturiert. Sie besitzen nur einen Wert und dürfen weder Attribute noch Kindelemente besitzen. Die meisten einfachen Datentypen werden vom XML-Schema bereits vorgegeben. Ein häufig verwendeter, einfacher Datentyp ist beispielsweise `xsd:string`. Mithilfe des String-Datentyps, lassen sich alle bei XML erlaubten Zeichen beschreiben. Einfacher, nicht weiter verschachtelter Text in einem XML-Dokument wird daher in einer XSD mit `xsd:string` bestimmt (vgl. Vonhoegen 2011, S. 124–125).

Komplexe Datentypen werden dagegen immer vom Schema-Designer konstruiert. Datentypen sind dann komplex, wenn sie Elemente beschreiben, die über Attribute oder Kindelemente verfügen. Komplexe Datentypen setzen sich aus Partikeln zusammen. Unter Partikeln werden lokale Elementdeklarationen oder verweise auf globale Elementdeklarationen verstanden (vgl. Vonhoegen 2011, S. 122–123). Abbildung 5 zeigt den Zusammenhang zwischen einfachen und komplexen Datentypen in einer XSD:

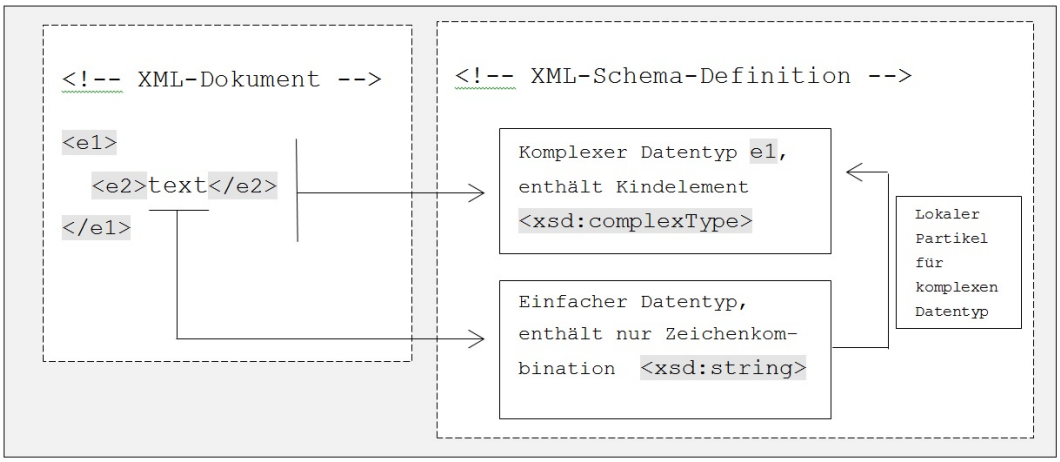

Abbildung 5: Einfache und komplexe Datentypen

Eine weitere Möglichkeit, die Struktur von XML-Dokumenten zu bestimmen, ist die Dokumententyp-Definition (DTD). Mit einer DTD kann ebenfalls der logische Aufbau und die Form des Inhalts vorgegeben werden. Im Gegensatz zu XSD sind DTD allerdings älter und unflexibler, dafür aber leichter zu verstehen und zu erstellen. DTD spielen in der Praxis, vor allem im Bereich der Textdigitalisierung, noch immer eine wichtige Rolle. So stützt die *Text Encoding Initiative* (TEI) sich auf äußerst umfangreiche Regelwerke in Form von DTD (vgl. Vonhoegen 2011, S. 42). Ein weiterer offener Standard, der maßgeblich auf DTD basiert, ist *DocBook*. DocBook wird in der Praxis ebenfalls genutzt, um Buchformate mit XML-Markup anzureichern. Die aktuelle Version DocBook v5.0 liegt seit November

2009 vor.[31] Veröffentlicht wird DocBook von der *Organization for the Advancement of Structured Information Standards* (OASIS), die hierfür ein Komitee mit dem Namen *DocBook Publishers SC* ins Leben rief (vgl. OASIS Consortium 2014, DocBook Publishers SC).

XML-Dokumente, die mit einer XSD oder einer DTD verknüpft sind, zeichnen sich dadurch aus, dass sie nicht nur *wohlgeformt*, sondern auch *gültig* sind. Wenn sie dem Inhaltsmodell einer XSD oder DTD folgen, werden XML-Dokumente von einem XML-Prozessor dem Inhaltsmodell entsprechend validiert. Sie werden im Fall einer erfolgreichen Validierung auch als Instanzen des Regelwerks bezeichnet (vgl. Vonhoegen 2011, S. 76).

3.2.2 XSLT

Die Bedeutung, die XML in der Praxis zukommt, verdankt die Bezeichnungssprache im besonderen Maß der Möglichkeit, Voraussetzungen für einen plattformneutralen Informationsaustausch zu schaffen. Während ein XML-Dokument lediglich die Struktur des Inhalts beschreibt oder den Inhalt mit Metadaten anreichert, spielen für den B2B- und B2C-Bereich vor allem der Austausch und die Publikation von Inhalten eine zentrale Rolle. Dafür müssen XML-Dokumente in neue und anders strukturierte Zielformate umgewandelt werden. Für eine solche Umwandlung entwickelte das W3C die *eXtensible Stylesheet Language* (XSL). XSL umfasst zwei Teilbereiche: *XSL Transformation* (XSLT) und *XSL Formatting Objects* (XSL-FO).

XSLT ist eine Programmiersprache, die zur Transformation von XML-Dokumenten in gewünschte Zielformate dient. Die aktuelle Version lautet XSLT 2.0[32] und liegt seit Januar 2007 vor. XSLT ist eine XML-Subsprache und dementsprechend in XML formuliert. Mit XSLT lässt sich schrittweise die Umwandlung eines XML- Dokuments in das gewünschte Ausgabeformat bestimmen. Das Zieldokument kann dabei beliebig weit von der Quelldatei abweichen. Elemente können umstrukturiert, ausgelassen oder komplett neu geschaffen werden.

Bei einem Transformationsprozess mit XSLT sind drei Dokumente beteiligt. Zunächst greift der XSLT-Prozessor auf das Quelldokument zu. Dieses besitzt die zu verarbeitenden Informationen. Das XSLT-Dokument, dass die Transformation bestimmt, wird XSLT-Stylesheet genannt. Da XSLT in XML formuliert ist, müssen sowohl die Quelldatei als auch das Stylesheet der XML-Syntax entsprechend wohlgeformt sein. Das Stylesheet dient dem XSLT-Prozessor als Regelwerk. Nach Abschluss der Transformation entsteht ein Ergebnisdokument. Struktur und Inhalt dieses Dokuments ergeben sich aus dem Quelldokument und den Verarbeitungsanweisungen des Stylesheets (vgl. Bongers 2008, S. 27–28). Abbildung 6 verdeutlicht den Ablauf einer Transformation mit XSLT.

[31] Vgl. hierzu: https://www.oasis-open.org/standards#dbv5.0 [06.03.2014].

[32] Vgl. hierzu: http://www.w3.org/TR/xslt20/#xslt-mime-definition [22.02.2014].

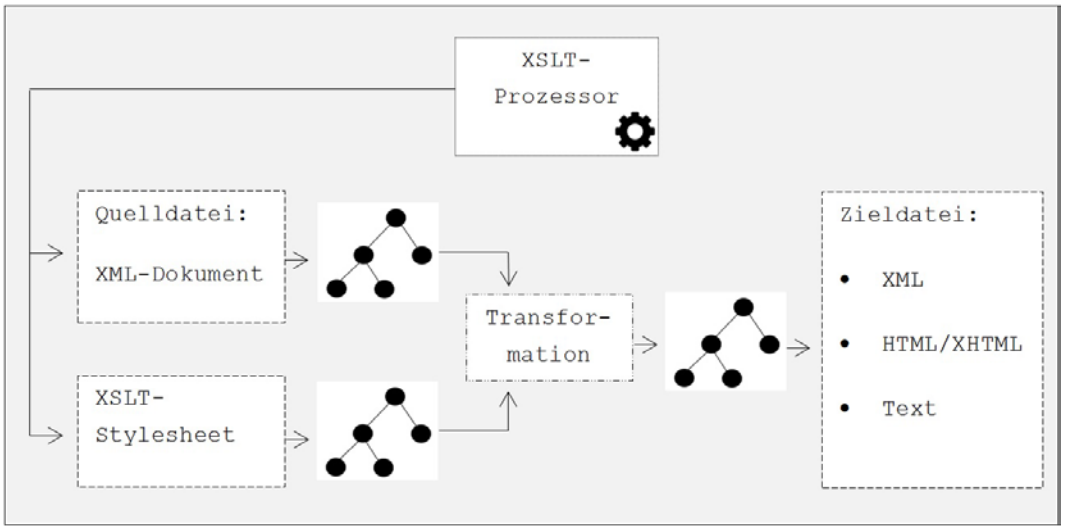

Abbildung 6: XSLT-Transformation

Der XSLT-Prozessor parst zunächst das Quelldokument und das Stylesheet. Anschließend werden beide Dokumente in eine Baumstruktur umgewandelt und gespeichert. Die Elemente des XML- Dokuments werden im Quelldokumentbaum als Knoten repräsentiert. Die Struktur dieses Baumes steuert indirekt die Verarbeitung, indem es sukzessive abgearbeitet wird. Findet der Prozessor bei der Abarbeitung des Quelldokumentenbaumes eine Anweisung für den aktuellen Knoten (von engl. *current node*) in der Baumstruktur des Stylesheets, so wird diese auf den aktuellen Knoten angewandt. Mit fortschreitender Verarbeitung wächst der Ergebnisbaum. Die Transformation ist beendet, wenn das Quelldokument vollständig abgearbeitet ist und in einem letzten Schritt serialisiert wurde. Unter Serialisierung versteht man die Umwandlung des Ergebnisbaums in ein wohlgeformtes Zielformat (vgl. Bongers 2008, S. 62–62; Vonhoegen 2011, S. 255–256).

3.2.3 XPath

XSLT nutzt die *XML Path Language* (XPath), um auf gewünschte Elemente zugreifen zu können: »XPath provides directions from one element on the XML tree to another element on the XML tree. Thus, you determine a starting element and an ending element, XPath is the path between them« (Whitmore 2013, S. 11). XPath ist demnach eine Pfadbeschreibungssprache, um Elemente im XML-Dokument zu adressieren. XPath wurde vom W3C entwickelt. Die aktuelle Version lautet 2.0. Die letzte W3C-Empfehlung stammt von Dezember 2010.[33] XPath selbst ist keine XML-Anwendung, sondern ist in einer eigenen Syntax formuliert. XPath arbeitet auf der Basis der schon genannten Baummodelle. Ein Baummodell enthält alle Bestandteile des XML-Dokuments, die in der Infoset-Empfehlung vorgegeben sind. Das betrifft demnach alle Elemente, Attribute, Kommentare, Verarbeitungsanweisungen und Zeichenreferenzen, die im XML-Dokument existieren.

[33] Vgl. hierzu: http://www.w3.org/TR/xpath20/ [22.02.2014].

Alle im Infoset erwähnten Informationsträger werden in entsprechende Knotentypen umgewandelt, auf die Prozessoren mithilfe von XPath zugreifen können. Der oberste Knoten ist der sogenannte Dokumentknoten. Der Dokumentknoten hat als einziger Knoten kein Pendant im XML-Dokument. Stattdessen trägt der Dokumentknoten das gesamte XML-Dokument in sich. Der Dokumentknoten ist also ein Konstrukt, von dem aus alle Transformationsprozesse beginnen (vgl. Bongers 2008, S. 55–56; Vonhoegen 2011, S. 185–186). Die Adressierung mit XPath geschieht über sogenannte XPath-Ausdrücke, die eine eigene Syntax haben:

```
achsenbezeichner::knotentest( )
```

Der Achsenbezeichner legt die Bewegungsrichtung und die Reichweite der Abfrage fest. XPath unterscheidet dabei zwischen dreizehn Achsen, die alle eine unterschiedliche Reichweite und Richtung aufweisen. Ausgangspunkt für einen XPath-Ausdruck ist immer der Kontextknoten, also der aktuelle Knoten, der bei der Transformation von einem XSLT-Prozessor eingelesen wird. Die wichtigste Achse ist die sogenannte *Child-Achse*. Diese Achse beschreibt den Schritt von Kontextknoten zu dessen Kindelementen. Die *Parent-Achse* beschreibt den umgekehrten Weg. Die *Self-Achse* gibt dagegen immer den aktuellen Kontextknoten als Ergebnis heraus.

Der Knotentest wählt den Knoten innerhalb der gesamten Knotenmenge aus, der am Ende der Achse auffindbar sein muss. Die zwei Funktionsklammern `( )` verdeutlichen, dass es sich dabei um eine aktive Filterung handelt. Handelt es sich bei dem anzusteuernden Knoten um einen Elementknoten, so spricht man von einem *NameTest*. Für den Knotentest wird in diesem Fall der im XML-Dokument verwendete Name des anzusteuernden Elements gewählt. Es könnten aber auch andere Knotengattungen angesprochen werden. Mit `text( )` können beispielsweise alle Textknoten angesprochen werden und mit `comment( )` werden Kommentarknoten ausgewählt. Der allgemeinste Knotentest lautet `node( )` und spricht alle Knoten aus der Knotenmenge an. Die zusammengesetzten Achsenbezeichner und Knotentests ergeben den gesamten Pfadausdruck. Beginnt dieser beim Dokumentknoten, spricht man von einem absoluten Pfadausdruck, andernfalls ist der Ausdruck relativ. Die folgende Abbildung zeigt eine Baumstruktur, an der die vorgestellten XPath-Ausdrücke besser veranschaulicht werden können.

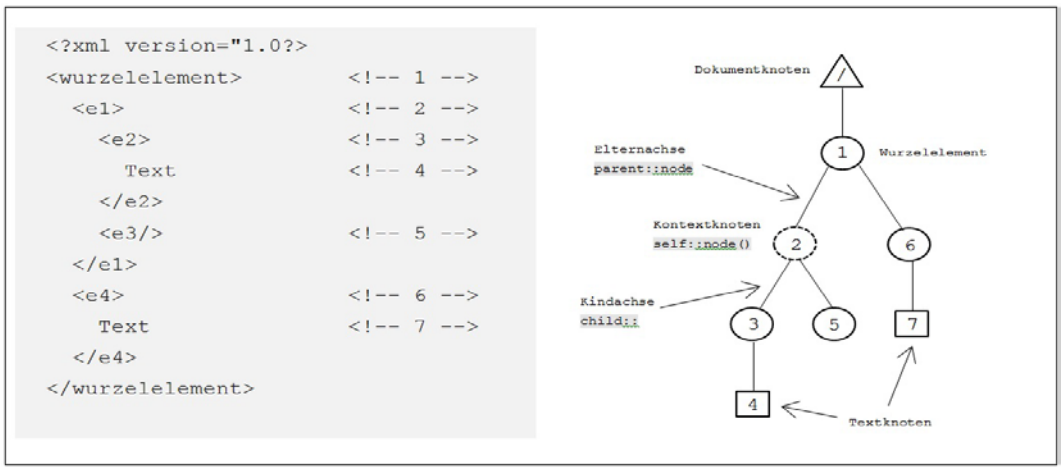

Abbildung 7: XPath-Pfade nach BONGERS 2008, S. 56

Der Quelldokumentbaum auf der rechten Seite der Abbildung beginnt mit dem Dokumentknoten, für den es im Quelldokument rechts kein Pendant gibt. Die Reihenfolge aller nachfolgenden Knoten ergibt sich aus der sogenannten Dokumentreihenfolge und entspricht der Leserichtung des XML- Dokuments. Element `<e2>` und Element `<e4>` haben Textzeichen als Inhalt. Diese Textzeichen werden bei XPath in Textknoten umgewandelt und können demnach auch angesteuert werden. In der Abbildung wird deutlich, wie ein Quelldokument sukzessive eingelesen wird. Der Kontextknoten ist Element `<e1>`. Das bedeutet, dass dieses Element vom Prozessor im Moment der Darstellung eingelesen wird. Der Kontextknoten kann mit `self::node( )` angesprochen werden. Relative Pfadausdrücke können vom Kontextknoten aus jedes Element ansprechen. Die zwei wichtigsten Achsen sind in der Abbildung dargestellt: die Kindachse `child::` und die Elternachse `parent::node( )`. Möchte man von Kontextknoten aus dessen Kindelement ansteuern, sähe das wie folgt aus: `e1/child::e2`. Umgekehrt wird über die Elternachse `e1/parent::wurzelelement` das Elternelement angesprochen. Im Beispiel nicht erwähnt, aber dennoch sehr wichtig ist außerdem die Attributachse. Deren Ausdruck lautet: `attribute::` (vgl. Bongers 2008, S. 70–81).

Durch Funktionen und sogenannten *Predicates* – also Filterbedingungen beim Knotentest – lassen sich mit XPath deutlich komplexere Pfade formulieren. Deren Erläuterung würde allerdings den Rahmen der vorliegenden Arbeit sprengen. Wichtig ist vorerst: XPath wird von XSLT und anderen XML-Subsprachen genutzt, um bei der Verarbeitung des Quelldokuments alle gewünschten Informationsträger ansteuern zu können. Dabei benutzt XPath Pfadausdrücke, die sich aus einer Achse – also der Bewegungsrichtung und Reichweite – und einem Knotentest – der eigentlichen Auswahl des Knotens – zusammensetzt.

3.2.4 XSL-FO

Der zweite Teilbereich von XSL ist XSL-FO. XSL-FO wird genutzt, um XML-Dokumente zu formatieren. Mit XSL-FO können Texte, Bilder und grafische Elemente auf Seiten angeordnet werden. Für Verlage ist XSL-FO insofern sehr interessant, da mit dieser Sprache hochwertige Druckerzeugnisse produziert werden können. XSL-FO liegt in Version 1.1 vor. Die letzte Spezifikation stammt vom Dezember 2006.[34] Die möglichen Ausgabeformate hängen vom XSLT-FO-Prozessor ab. Das zentrale Ausgabeformat ist das Portable-Document-Format (PDF).

Eine Transformation mit XSL-FO greift auf XSLT- und XPath-Technologien zurück. Die Transformation geschieht mithilfe von XSLT und der Zugriff auf die Informationseinheiten wird mittels XPath realisiert. Lediglich die Formatierung der Informationseinheiten wird auf der Basis von speziellen XSL-FO-Elementen umgesetzt. Das bedeutet: Nach der Transformation mit XSLT entsteht, wie bereits erläutert, ein Ergebnisbaum. In einem nächsten Schritt greift ein XSL-FO-Prozessor – auch *XSL Formatter* genannt – auf diesen Baum zu und formatiert ihn dem XSL-FO-Stylesheet entsprechend. Dieses Stylesheet gleicht daher einem

[34] Vgl. hierzu: http://www.w3.org/TR/xsl11/ [23.02.2014].

XSLT-Stylesheet, verfügt aber zusätzlich über sogenannte Formatierungsobjekte, die wie gewohnt in Form von Elementen dargestellt werden. VONHOEGEN beschreibt Formatierungsobjekte folgendermaßen: »Die Formatierungsobjekte sind jeweils Instanzen von abstrakten Klassen, die typografische Dinge, wie Seite, Absatz, Tabelle etc. repräsentieren« (Vonhoegen 2011, S. 334). In anderen Worten: XSL-FO stellt für alle typografisch relevanten Gestaltungsoptionen Kategorien zur Verfügung, die in einem Stylesheet in Form von XML-Elementen vorliegen.

XSL-FO formatiert den Inhalt eines XML-Dokuments in Form von rechteckigen Bereichen. Jeder Bereich trägt drei wichtige Informationen in sich: seine Position auf einer Seite, die anzuzeigenden Informationseinheiten sowie die Formatierungsanweisungen, auch *Traits* genannt. Jeder Bereich stellt insofern eine Art Container für die Inhalte des XML- Dokuments dar. Grundsätzlich verfügt XSL-FO über zwei Bereiche: Block-Bereiche und Inline-Bereiche. Mit Block-Bereichen lassen sich Inhalte auf einer Seite anordnen. Der XSL-FO-Ausdruck für Block-Bereiche lautet: `<fo:block>`. Mit Blockobjekten können einzelne Absätze, Überschriften oder vergleichbare gesonderte Bereiche formatieren werden. Inline-Bereiche werden dagegen genutzt, um einzelne Zeichenfolgen innerhalb eines Block-Bereichs separat zu formatieren, beispielsweise die Initiale am Anfang eines Kapitels. Der entsprechende Ausdruck lautet `<fo:inline>`. Inline-Objekte sind daher in aller Regel Kindelemente von Block-Objekten (vgl. Vonhoegen 2011, S. 333–336, 342–347).

Abbildung 8 zeigt, dass ein Formatierungsprozess mit XSL-FO ähnlich abläuft, wie ein Formatierungsprozess mit XSLT. Allerdings wird bei einer XSL-FO-Formatierung kein eigenes XSLT-Stylesheet benötigt, da alle erforderlichen XSLT-Anweisungen ebenfalls im XSL-FO-Stylesheet vorliegen. Der XSLT-Prozessor greift wie gewohnt auf die Quelldatei und das XSL-FO-Stylesheet zu, wandelt diese in Baumstrukturen um und transformiert beide in einen Ergebnisbaum. An dieser Stelle beginnt der XSL-FO-Formatter mit seiner Arbeit. Auch er findet alle erforderlichen Anweisungen um XSL-FO-Stylesheet und wandelt den Ergebnisbaum den Anweisungen entsprechend in die Zieldatei um, die in diesem Fall ein PDF besitzt.

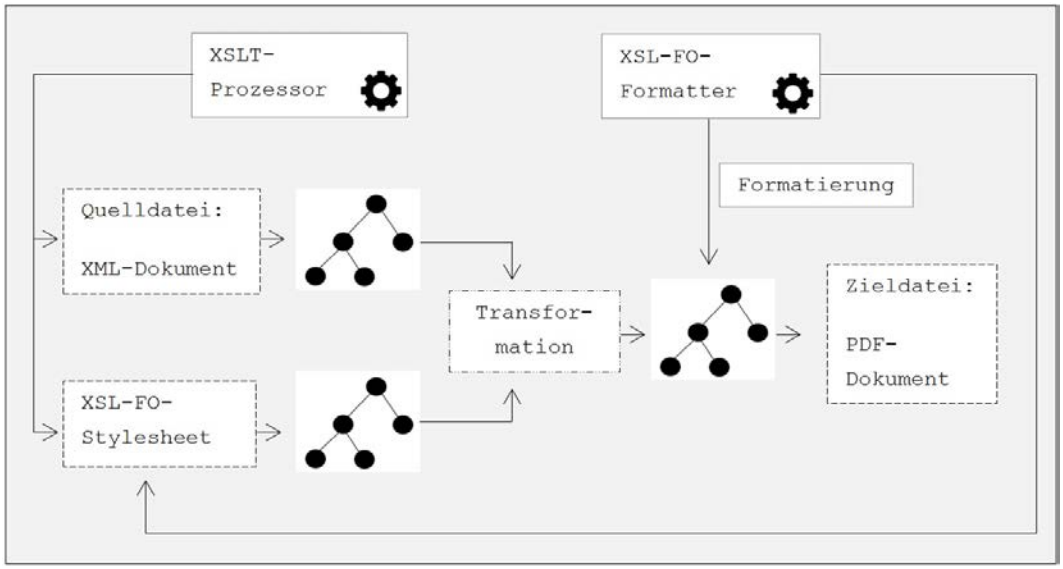

Abbildung 8: XSL-FO Formatierungsprozess

Der eigentliche Formatierungsprozess – in der Abbildung unter ‚Formatierung' zusammengefasst – besteht wiederum aus drei Phasen. In der ersten Phase werden alle Knoten des Ergebnisbaums in Formatierungsobjekte umgewandelt. Dadurch entsteht eine neue Baumstruktur. Dieser Baum wird in einer zweiten Phase verfeinert. Im Zuge der Verfeinerung werden beispielsweise alle Leerzeichen und Duplikate entfernt. In der entscheidenden dritten Phase wird die Baumstruktur in den sogenannten *Area Tree* umgewandelt. Dieser Baum repräsentiert alle Bereiche des gesamten Zieldokuments in Form von rechteckigen Blöcken. Alle Formatierungsanweisungen werden im *Area Tree* in die schon vorgestellten *Traits* umgewandelt. Durch die hierarchische Baumstruktur können alle Bereiche ähnlich wie mit XPath angesteuert werden. Nachdem der *Area Tree* sukzessive abgearbeitet wurde, entsteht das gewünschte PDF-Dokument.

Der Einsatzbereich von XSL-FO liegt in der automatisierten Aufbereitung von XML-Dokumenten. XSL-FO-Stylesheets sind in der XML-Syntax geschrieben und müssen daher wohlgeformt sein. Der Ablauf einer XSL-FO-Transformation ist in aller Regel automatisch. Zwar lassen sich die Ergebnisdateien mit Layout- und Satzprogrammen wie *Adobe InDesign* nachbearbeiten, für ein interaktive Gestaltung von Druckprodukten ist XSL-FO allerdings nicht verwendbar (vgl. Krüger / Welsch 2007, S. 11).

3.2.5 CSS

Interaktiver aber weniger komplex stellt sich die Formatierung mittels *Cascading Style Sheets* (CSS) dar. CSS ist keine XML-Sprache, spielt aber insbesondere in der Darstellung von HTML- und XHTML-Seiten sowie E-Books in EPUB eine zentrale Rolle und soll daher kurz vorgestellt werden. CSS liegt in Version 2.1

vor. Das W3C veröffentlichte den aktuellen Standard im Juni 2011.[35] CSS ist eine Beschreibungssprache, die es ermöglicht, Web-Dokumente einfach und schnell zu formatieren. CSS wurde für HTML entwickelt. Entsprechende Stylesheets lassen sich jedoch auch unproblematisch mittels Verarbeitungsanweisungen mit XML-Dokumenten verknüpfen. CSS basiert dabei auf der Idee, Inhalt und Layout streng zu trennen. Daher werden CSS-Formatierungen in aller Regel in externen Stylesheets definiert und nicht im zu formatierenden Dokument selbst.

Im Gegensatz zu XSL-FO arbeitet CSS nicht mit XSLT und XPath. CSS ist daher deutlich simpler, beschränkt sich allerdings auch auf die reine Formatierung von Quelldokumenten. Transformationen, Erweiterungen oder sonstige Änderung – wie sie mit XSL-FO möglich sind – können mit CSS nicht umgesetzt werden. CSS arbeitet wie auch XSL-FO mit zueinander in Verbindung stehenden rechteckigen Blöcken, sogenannte *Blockelemente*. Blöcke können vertikal und horizontal angeordnet sein. Alle Blöcke haben einen variablen Außen- und Innenabstand sowie einen Bereich, der aus den Informationen des zu formatierenden Dokuments besteht. Das Formatierungsmodell gleicht daher dem von XSL-FO in vielen Details (Vonhoegen 2011, S. 238–239). CSS-Stylesheets werden in Form von Listen formuliert. CSS hat dafür eine eigene, nicht mit XML formulierte Syntax:

```
Selektor {Eigenschaft: Wert}
```

Der Selektor hat die Aufgabe, die im Quelldokument zu formatierenden Elemente zu benennen. In der Umsetzung der Syntax wird dem Selektor einfach der Name des zu formatierenden Elements gegeben. Jeder XML-Elementtyp kann als Selektor verwendet werden. Dem Selektor folgt die Deklaration der Formatierungsregeln. Die Deklaration besteht stets aus einer oder mehreren Eigenschaften und den dazugehörigen Werten. So führt beispielsweise die Deklaration `absatz {text-align: left}` dazu, dass das im Selektor ausgewählte Element `absatz` linksbündig ausgerichtet wird (vgl. Castro/Hyslop 2012, S. 163–165; Erlenkötter 2012, S.54–56; Vonhoegen 2011, S. 227–230).

3.2.6 EPUB

Die letzte Schlüsseltechnologie, die in diesem Kapitel vorgestellt werden soll, hat den Namen *electronic Publication* (EPUB). Die aktuelle Version lautet seit 2011 EPUB 3.0.[36] Auch die 2010 verabschiedete Version 2.0.1[37] wird auf aktuellen Lesegeräten in vollem Umfang unterstützt. Im Gegensatz zu den vorangestellten Schlüsseltechnologien wurde der EPUB-Standard nicht vom W3C entwickelt, sondern vom *International Digital Publishing Forum*. EPUB ersetzt den älteren Standard *Open eBook Publication Structure* (OEBPS). Auch wenn EPUB weder eine eigene Bezeichnungssprache darstellt, noch vom W3C entwickelt wurde, spricht einiges dafür, EPUB als Schlüsseltechnologie für die vorliegende Masterarbeit zu betrachten. Erstens basiert EPUB wesentlich auf XML und XHTML.

[35] Vgl. hierzu: http://www.w3.org/TR/CSS21/%20 [23.02.2014].

[36] Vgl. hierzu: http://idpf.org/epub/30 [23.02.2014].

[37] Vgl. hierzu: http://idpf.org/epub/201 [23.02.2014].

Zweitens ist EPUB eines der drei Zielformate, die im Verlauf der Masterarbeit entstanden.

Unter dem Begriff EPUB werden drei Standards zusammengefasst: das *Open Publication Format* (OPF), die *Open Publication Structure* (OPS) und das *Open Container Format* (OCF). Alle drei Subformate von EPUB steuern die Struktur und die Inhalte einer EPUB-Datei. EPUB fungiert dabei als ein Container im ZIP-Dateiformat, in dem mehrere Dateien komprimiert und gespeichert werden. OPS hat das Ziel, die für die Publikation und Darstellung verwendeten Formate zu standardisieren. Für EPUB 2.0.1 gilt XHTML als Standardformat. Für EPUB 3.0 wurde dagegen auf die XML-Version von HTML5 zurückgegriffen. Struktur und Metadaten der Publikation werden mittels des OPF-Standards geregelt. Außerdem bietet OPF seit Version 2.0 ein Konzept zur Navigation in Form eines Inhaltsverzeichnisses für E-Books im EPUB-Format an. OCF regelt schließlich die Verpackung aller Teildateien in einer ZIP-Datei mit der Dateiendung `.epub`. Neben den durch OPS und OPF vorgegeben Dateien fordert der OCF-Standard eine nicht komprimierte Datei mit dem Namen `mimetype` und dem Inhalt `application/epub+zip` als Bestandteil des Containers. Dieser MIME-Type klassifiziert den Container als EPUB-Datei (vgl. Wang 2011, S. 219).

4 Die Umsetzung XML-basierter Textanreicherung – Von XML zum Zielformat

Nachdem in den vorigen Kapiteln der vorliegenden Masterarbeit eine theoretische Fundierung erarbeitet wurde, folgt in diesem Kapitel die praktische Anwendung eines möglichen Content-Managements. Für den praktischen Teil der Masterarbeit wurden eine Reihe von XML-basierten Stylesheets geschrieben, die in den entsprechenden Unterkapiteln vorgestellt werden.

Die für die Masterarbeit erstellten Stylesheets können aufgrund ihres Umfangs nicht in voller Länge abgebildet und erläutert werden. Vielmehr erfolgt die Darstellung auch hier in Form von repräsentativen Programmausdrücken. Alle produzierten Stylesheets befinden sich in vollem Umfang im Anhang.

4.1 Versuchsaufbau

Dieser Teil der Masterarbeit verfolgt das Ziel, den Arbeitsablauf von der Erstellung eines XML- Dokuments bis hin zu den gewünschten Zielformaten zu dokumentieren und die dafür erforderlichen Techniken zu erläutern. Der für ein XML-basiertes Content-Management notwendige Workflow erfolgt in vier Schritten und orientiert sich an dem in Kapitel 2.4 vorgestellten Content-Management-Prozess. Abbildung 9 zeigt den schematischen Versuchsaufbau für die vorliegende Masterarbeit.

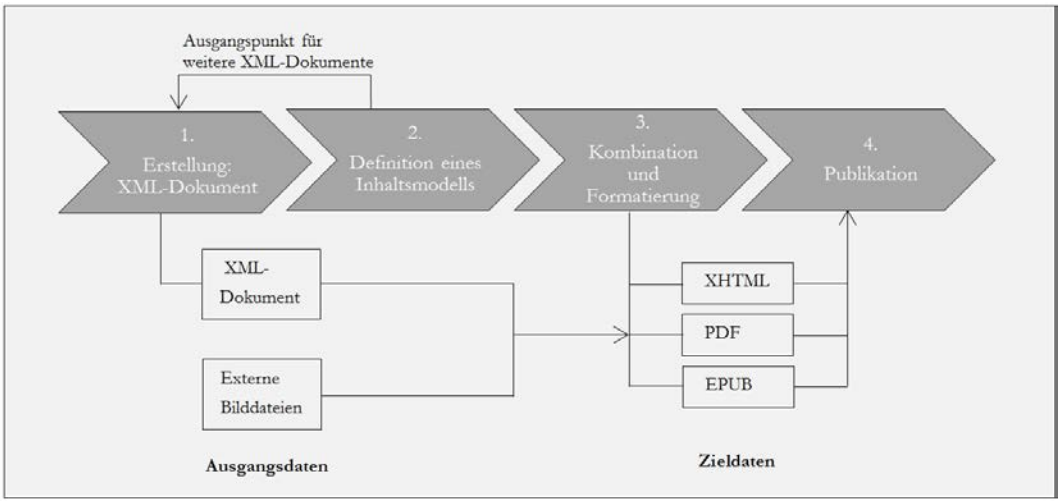

Abbildung 9: Schematischer Versuchsaufbau

Der erste Schritt sieht die Erstellung eines XML-Dokuments vor. Hierfür soll eine Textdatei mit XML-Markup angereichert werden. Als Ergebnis entsteht eine Ausgangsdatei im XML-Format. Anschließend erfolgt die Archivierung des vorläufigen XML-Dokuments. Zusätzlich existiert noch eine Reihe von Illustrationen in

Form von Bilddateien, die ebenfalls Ausgangsdaten für den CM-Prozess darstellen. Die Illustrationen wurden in einem gemeinsamen Projekt von der Grafikerin FRIEDERIKE WOLF[38] an der HFBK Hamburg angefertigt (vgl. Anhang Nr. 1). XML-Dokument und Bilddateien sind die in Kapitel 2.3 beschrieben Assets, die im Verlauf des praktischen Teils zu einem Contentobjekt zusammengefasst werden sollen. Im zweiten Schritt wird aus dem XML-Dokument ein Inhaltsmodell in Form einer XML-Schema-Definition abgeleitet. Die abgeleitete XSD kann als Konzept und Ausgangspunkt für die Erstellung weiterer Contentobjekte in Form von XML-Dokumenten dienen. Die Schema-Definition ist daher die Basis für weitere CM-Prozesse. Im dritten Schritt wird das archivierte XML-Dokument in die drei gewünschten Zielformate XHTML, PDF und EPUB transformiert. Hierfür findet außerdem eine Kombination des XML-Dokuments mit einer Reihe von Bilddateien statt. In einem vierten Schritt sollen die produzierten Zielformate schließlich publiziert werden. Für die Publikation wurde ein Webspace eingerichtet, der über die URL http://www.samsas-verwandlung.de/ erreicht werden kann.

Für den praktischen Teil der Masterarbeit wurde eine Reihe von Programmen verwendet. Für die Anreicherung der Textdatei mit XML-Markup, die Erstellung einer XML-Schema-Definition, die Validierung des XML-Dokuments und dessen Transformation in PDF wurden die XML-Editoren *XML-PAD 3* [39] und *XMLSpy2014* von *Altova* genutzt.[40] Für die XSLT-Transformation in XHTML wurde die Open-Source-Software *Kernow 1.7.2* verwendet.[41] Der EPUB-Container wurde mithilfe von *WinRAR 5.0.1* erstellt.[42] Die Überprüfung des E-Books geschah mithilfe der freien Software *EPUB-Checker 1.3.0* der Firma *pagina*.[43] Für die Darstellung des E-Books wurden die Programme *calibre 1.2.8* [44] und *Adobe Digital Editions 3.0* [45] sowie der E-Reader PRS-T3s von Sony verwendet. Außerdem wurde eine Reihe von Prozessoren verwendet, die in Kapitel 4.2 vorgestellt werden.

4.2 Programmierschnittstellen und Prozessoren

Um XML-Anwendungen in ein Betriebssystem zu implementieren, werden abstrakte Schnittstellen benötigt. Programmierschnittstellen werden entwickelt, um Programme an externe Softwaresysteme anzubinden. Die wichtigste Schnittstelle für XML-Dokumente wird von einer Spezifikation des W3C selbst definiert und trägt den Namen *Document Object Model* (DOM).[46] Die aktuelle Version ist DOM Level 3. DOM ist eine sprach- und plattformunabhängige Schnittstelle mit

[38] Vgl. hierzu: http://www.friederikewolf.com/ [23.03.2014].

[39] Vgl. hierzu: http://xml-pad.sourceforge.net/ [19.03.2014].

[40] Vgl. hierzu: http://www.altova.com/de/xmlspy.html [10.03.2014].

[41] Vgl. hierzu: http://kernowforsaxon.sourceforge.net/ [10.03.2014].

[42] Vgl. hierzu: http://www.rarlab.com/ [10.03.2014].

[43] Vgl. hierzu: http://www.pagina-online.de/produkte/epub-checker/ [10.03.2014].

[44] Vgl. hierzu: http://calibre-ebook.com/about#features [19.03.2014].

[45] Vgl. hierzu: http://www.adobe.com/de/products/digital-editions.html [19.03.2014].

[46] Vgl. hierzu: http://www.w3.org/DOM/ [07.02.2014].

der Aufgabe, den Zugriff auf XML-Dokumente zu vereinheitlichen. Hierzu werden mit DOM XML-Dokumente in ein abstraktes Baummodell übertragen. Dieses Baummodell beinhaltet alle Bestandteile des zu verarbeitenden XML-Dokuments. Das Modell kann anschließend von Programmen genutzt werden, um auf ein beliebiges darin gespeichertes Element zuzugreifen. DOM erlaubt für diesen Zugriff XPath-Ausdrücke (vgl. Erlenkötter 2012, S. 222–223). Durch die baumorientierte Verarbeitung besteht außerdem die Möglichkeit, XML-Dokumente zu validieren und bei Bedarf umzustrukturieren (vgl. Eckstein/Eckstein 2004, S. 7).

Neben DOM wird häufig eine weitere Schnittstelle genutzt. Diese trägt den Namen *Simple API for XML* (SAX)[47] und liegt aktuell in der Version 2.0.2 vor. SAX ist weniger komplex und speichersparender als DOM. Dieser Umstand liegt darin begründet, dass mit SAX kein Ansatz gewählt wurde, der auf einem Baummodell basiert, sondern ein ereignisorientierter: Mit SAX werden XML-Dokumente analysiert und sequenziert. Jedes Ereignis – beispielsweise das Parsen eines Elementes – wird anschließend an das mit SAX arbeitende Programm gemeldet, woraufhin dieses Programm Aktionen aus den Meldungen ableiten kann (vgl. Erlenkötter 2012, S. 211–212).

Für beide Schnittstellen sind XML-Prozessoren notwendig. Diese parsen XML-Dokumente und bilden das Front-End für die mit der Weiterverarbeitung beauftragten Programme (vgl. Vonhoegen 2011, S.41, S.608). Für die vorliegende Masterarbeit wurde eine Reihe von XML-Prozessoren verwendet. Die Verarbeitung der XML-Dokumente erfolgte mithilfe der *Microsoft XML Core Services* (MSXML).[48] MSXML stellt eine Reihe von Services zur Verfügung, die sowohl DOM als auch SAX unterstützen. Außerdem lassen sich XML-Dokumente mithilfe von MSXML und XSD validieren. Für die Transformation von XML-Dokumenten in XHTML mittels XSLT wurde der XSLT-Prozessor Saxon-HE verwendet. Saxon ist ein Open-Source-Prozessor und wird von dem Entwickler *Saxonica Limited* bereitgestellt.[49] Die Transformation des XML-Dokuments in PDF mit XSL-FO geschah mithilfe des *Formatting Objects Processor* (FOP). FOP ist ebenfalls ein Open-Source-Prozessor und basiert auf Java. FOP wird von dem Entwickler *Apache Software Foundation* veröffentlicht und laufend aktualisiert.[50]

4.3 Anreicherung von Text-Dateien: Das XML-Dokument

Grundlage für XML-basiertes Content-Management ist das XML-Dokument. Es stellt ein format- und plattformunabhängiges Quelldokument dar, das automatisch in verschiedene Zielformate transformiert werden kann. Für die vorliegende Masterarbeit wurde Kafkas *Die Verwandlung* als repräsentatives Beispiel mit XML-Tags angereichert. Der Text ist gemeinfrei und stammt aus dem *Projekt Guten-*

[47] Vgl. hierzu: http://www.saxproject.org/ [07.02.2014].

[48] Vgl. hierzu: http://www.microsoft.com/de-de/download/details.aspx?id=19662 [07.02.2014].

[49] Vgl. hierzu: http://saxon.sourceforge.net/#F9.4HE [07.02.2014].

[50] Vgl. hierzu: http://xmlgraphics.apache.org/fop/ [07.02.2014].

berg-DE.[51] In diesem Kapitel soll das entsprechende XML-Dokument in seinen Grundzügen vorgestellt werden. Das komplette Dokument `kafka_verwandlung.xml` befindet sich außerdem im Anhang (vgl. Anhang Nr. 2).

4.3.1 Der Prolog

Jedes XML-Dokument sollte über einen Prolog verfügen. Der Prolog umfasst die XML-Deklaration sowie optionale Verarbeitungsanweisungen (vgl. Kapitel 3.1). Listing 2 zeigt den Prolog von `kafka_verwandlung.xml`.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?xml>
```

Listing 2: XML-Prolog

Der Prolog des XML-Dokuments ist simpel. Die verwendete XML-Version ist 1.0. Für die Zeichenkodierung wurde `UTF-8` gewählt. Für westeuropäische Sprachen bietet sich zwar `ISO-8859-1` mehr an, da die für sie angedachte Zeichenkodierung auch Umlaute umfasst. `UTF-8` wird allerdings umfassender unterstützt, da es die vom W3C empfohlene Zeichenkodierung ist (vgl. Vonhoegen 2011, S. 52). Außerdem bietet sich `UTF-8` hinsichtlich des EPUB-Zielformates an, da auch das *International Digital Publishing Forum* (IDPF) `UTF-8` in der OPS-Empfehlung als Zeichenkodierung für Publikationen vorschlägt: »Publications may use the entire Unicode character set, using UTF-8 or UTF-16 encodings, as defined by Unicode […]« (vgl. IDPF 2010, OPS, Kapitel 1.3.6).

Außerdem wurde festgelegt, dass das XML-Dokument nur gültig ist, wenn es mit einer DTD oder XSD verknüpft ist. Zusätzliche Verarbeitungsanweisungen werden nicht benötigt, da alle Transformationsprozesse, die in den nächsten Kapiteln erläutert werden, über spezielle Verarbeitungssoftware auf das XML-Dokument zugreifen, ohne dass dafür spezielle Anweisungen im XML-Dokument selbst notwendig wären.

4.3.2 XML-Daten

Der gesamte Text von Kafkas *Die Verwandlung* wird im Bereich der XML-Daten (vgl. Abbildung 4) repräsentiert. Die für die Anreicherung verwendeten Tags werden von einer XSD vorgegeben und sind Gegenstand von Kapitel 4.4.

[51] Vgl. hierzu: http://gutenberg.spiegel.de/buch/165/1 [25.02.2014].

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?xml>

<programm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

     xsi:schemaLocation="http://www.ma-buwi-fh.de kafka_struktur.xsd"

     xmlns="http://www.ma-buwi-fh.de"

     xmlns:tst="http://www.ma-buwi-fh.de">             <!-- 1 -->

 <warengruppe tst:gruppenname="belletristik">             <!-- 2 -->

  <autor tst:autorname="Franz Kafka">                  <!-- 3 -->

   <buch tst:buchname="Die Verwandlung" tst:lang="de" tst:isbn="">  <!-- 4 -->

    <verfasser>Franz Kafka</verfasser>               <!-- 5 -->

    <titel>Die Verwandlung</titel>                 <!-- 6 -->

    <kapitel tst:nr="1">                       <!-- 7 -->

     <absatz>Als <person tst:idp="p1">Gregor Samsa</person> eines <!-- 8 -->

        Morgens aus unruhigen Tr&#228;umen erwachte...</absatz> <!-- 9 -->

     ...

    </kapitel>

    ...

   </buch>

  </autor>

 </warengruppe>

</programm>
```

Listing 3: XML-Daten

Listing 3 zeigt den Aufbau des XML-Dokuments. Das Dokument befindet sich im Anhang (vgl. Anhang Nr. 2).

(1) Das Wurzelelement lautet `programm`. Ihm wurden vier Attribute hinzugefügt. Diese Attribute dienen der Definition von Namensräumen und der Einbindung einer XSD. Namensräume werden in Kapitel 4.4.3 näher erläutert. Die Bedeutung von Namensräumen und den damit verbundenen Attributen kann an dieser Stelle nur skizziert werden. Grundsätzlich kann man sich einen Namensraum als fest definierte Gruppe von Vokabeln vorstellen, deren Bedeutung mittels einer XSD festgelegt werden kann. `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` legt fest, dass alle mit dem Präfix `xsi` ausgestatten Attribute zum *Schema Instance Namensraum* gehören. Dieser Namensraum ist vom W3C vorgegeben (vgl. W3C 2012, XSD 1.1 Part 1, Kapitel 1.3.1.2). `xmlns` bedeutet nichts anderes als *XML Namespace*. Das Attribut `xsi:schemaLocation` verfügt über das Präfix `xsi` und gehört daher zum genannten Namensraum. Mit `schemaLocation`

werden der Ort des XSD-Dokuments und der dort definierte Namensraum festgelegt. Durch diese Verknüpfung von XML-Dokument und einer XML-Schema-Definition kann somit der Inhalt des XML-Dokuments auf seine Gültigkeit überprüft werden. Die zwei weiteren Attribute `xmlns="http://www.ma-buwi-fh.de"` und `xmlns:tst="http://www.ma-buwi-fh.de"` bestimmen, dass der in der verknüpften XSD definierte Namensraum zum Standardnamensraum für das XML-Dokument bestimmt wird. Das XML-Dokument wird dadurch zur gültigen Instanz der XML-Schema-Definition, wenn sie den Regeln dieser Definition gerecht wird.

(2) Das Kindelement von `programm` lautet `warengruppe`. Die Warengruppe wird durch das Attribut `tst:gruppenname="belletristik"` spezifiziert. Die Idee für die Wahl der Namen ist, dass das gesamte Verlagsprogramm in Programmbereiche aufgeteilt werden kann. Die Programmbereiche sind der Warengruppen-Systematik des *Börsenvereins des deutschen Buchhandels* entlehnt (vgl. Börsenverein 2006, S. 6). Das bedeutet, dass die im Attribut getroffene Wahl des Programmbereichs schon eine Form der Systematisierung aller im Verlag publizierten Bücher ist.

(3) Das Kindelement von `warengruppe` ist `autor`. Auf den ersten Blick sind die neutralen Bezeichnungen von `warengruppe`, `autor` und `buch` irreführend, haben aber durchaus ihre Berechtigung. Die für die vorliegende Masterarbeit geschriebenen Stylesheets sollen nach dem Prinzip eines möglichst automatischen und generischen Content-Managements funktionieren. Alle Stylesheets, die zur Weiterverarbeitung des XML-Dokuments auf dessen Elemente zugreifen, haben den Anspruch grundsätzlich auch auf andere XML-Dokumente, die ihrem Aufbau nach dem hier verwendeten Dokument entsprechen, anwendbar zu sein. Das wird erst durch die neutralen Bezeichnungen ermöglicht, da anderenfalls jedes Stylesheet angepasst werden müsste. Um die XML-Dokumente dennoch unterscheidbar zu machen, werden Warengruppe, Autor und Titel durch entsprechende Attribute festgelegt. Durch die neutralen Bezeichnungen lassen sich Stylesheets somit auf eine ganze Klasse von XML-Dokumenten anwenden. Ein generisches Content-Management wird dadurch ermöglicht. Abbildung 10 verdeutlicht ein zentrales Motiv der vorliegenden Masterarbeit.

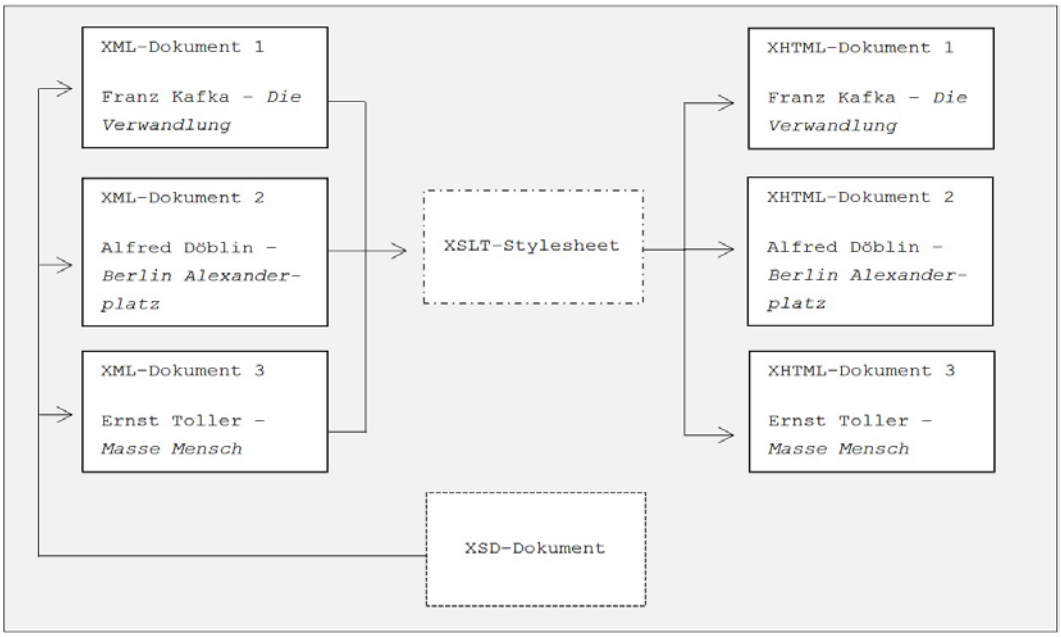

Abbildung 10: Generisches Content-Management

Die drei XML-Dokumente 1-3 entsprechen einer Klasse. Ihr Aufbau und Inhalt wird von einem XSD-Dokument bestimmt. Dadurch – und durch die neutralen Bezeichner – ist es möglich, dass ein XSLT-Stylesheet die gesamte Klasse von XML-Dokumenten in entsprechende XHTML-Dokumente transformiert, ohne dass das Stylesheet angepasst werden muss.

(4) Das Kindelement von `autor` heißt `buch`. Es ist das umfangreichste Element des XML-Dokuments, da es neben der Autoren- und Titelbenennung alle Kapitel umfasst. Das Element `buch` verfügt über drei Attribute: `tst:buchname`, `tst:lang` und `tst:isbn`. Das erste Attribut spezifiziert den Namen des Buchs. Mit `tst:lang` wird die darin verwendete Sprache festgelegt. Mithilfe des dritten Attributs kann dem Dokument eine ISBN-Nummer zugewiesen werden, wenn diese existiert.

(5) Nun beginnt der eigentliche Inhalt des Buchs. An erster Stelle wird der Verfasser genannt.

(6) An zweiter Stelle folgt der Titel des Buchs. Die Elemente `verfasser` und `titel` dürfen jeweils nur einmal und in dieser Reihenfolge auftreten.

(7) Das dritte Kindelement von `buch` heißt `kapitel`. Dieses Element darf beliebig oft auftreten. Über das Attribut `tst:nr` wird die Kapitelnummer festgelegt. Durch eine Eindeutigkeitsbestimmung in der XSD kann jede für das Attribut verwendete Nummer nur einmal vorkommen, sodass doppelt benannte Kapitel von Beginn an ausgeschlossen sind.

(8) Jedes Kapitel hat eine beliebige Anzahl von Absätzen. Das entsprechende Element heißt `absatz`. Der Inhalt von `absatz` ist gemischt. Er besteht aus Textzeichen – also dem eigentlich Text von Kafkas *Die Verwandlung* – sowie aus weiteren Kindelementen mit dem Namen `person`. Alle im Buch vorkommenden Personen werden mit entsprechenden Tags angereichert. Außerdem wird jeder Person mithilfe des Attributs `tst:idp` ein Kürzel zugewiesen. Erst dadurch wird ein Zugriff mit XPath auf die im Buch vorkommenden Personen und eine daraus resul-

tierende Weiterverarbeitung möglich. Dieser semantische Zugriff hat im Gegensatz zum rein textbasierten Zugriff den Vorteil, dass beispielsweise die drei Bezeichnungen ‚Schwester', ‚Grete' und ‚Tochter' alle als Gregors Schwester erkannt werden. Eine textbasierte Suche könnte den dafür benötigten Zusammenhang von Zeicheninhalt und Zeichenausdruck nicht herstellen. Entsprechend groß ist allerdings auch der Aufwand der Anreicherung.

(9) Im ersten Absatz auffällig ist folgender Ausdruck: `Tr&#228;umen`. Der Umlaut ‚ä' wurde in die Zeichenkette `&#228;` umgewandelt. Man spricht in einem solchen Fall von einer Zeichenreferenz oder Zeichenentität. Zeichenreferenzen werden genutzt, um Umlaute zu kodieren. Diese Kodierung erlaubt es XML-Prozessoren, Umlaute auszulesen, auch wenn diese im verwendeten Kodierungsverfahren nicht vorgesehen sind, wie es bei `UTF-8`, dem hier verwendeten Kodierungsverfahren, der Fall ist. Zeichenreferenzen beginnen immer mit einem Ampersand-Zeichen, gefolgt von einem Nummernzeichen und einer (hexa-)dezimalen Zahl, die für den jeweiligen Umlaut steht. Die Zeichenreferenz wird durch ein Semikolon abgeschlossen (vgl. Erlenkötter 2012, S. 27–28).

Dem in der vorliegenden Masterarbeit verwendete Verfahren, ein XML-Dokument zu erzeugen, indem ein Textdokument manuell mit XML-Tags angereichert wird, steht eine weitere Möglichkeit gegenüber, die an dieser Stelle kurz vorgestellt werden soll.

Mit Office 2007 führte Microsoft das XML-Dateiformat *Office Open XML* (OOXML) als Standardformat ein.[52] Die entsprechenden Endungen lauten `.docx` für Word, `.xlsx` für Excel und `.pptx` für PowerPoint. Auch wenn es nicht auf den ersten Blick ersichtlich ist, werden diese Formate immer in Form eines ZIP-Containers – vergleichbar mit dem EPUB-Format (vgl. Kapitel 4.7) – abgespeichert. Diese Container enthalten zum größten Teil XML-Dateien, aber auch eingebettete Bilder und Binärdaten. Die Struktur eines Containers kann aus beliebig vielen Komponenten bestehen. Der Zusammenhang der Komponenten wird in speziellen, sogenannte Beziehungskomponenten – ebenfalls in Form von XML-Dateien mit der Dateiendung `.rels` – festgelegt. Der eigentliche Inhalt wird in der Datei `document.xml` abgelegt. Kopf- und Fußzeile wiederum sind in die separaten Dateien `header.xml` und `footer.xml` ausgelagert. Dieses modulare Vorgehen hat den Vorteil, dass Entwickler gezielt einzelne Komponenten verändern können (vgl. Vonhoegen 2011, S. 523–529). Da Office-Anwendungen zu den am häufigsten benutzten Programmen gehören, liegt es Nahe, dass Verlage ihre ohnehin schon vorliegenden Office-Dokumente nutzen, um daraus XML-Dokumente zu gewinnen. Voraussetzung hierfür ist natürlich, dass die Office-Dokumente im OOXML-Format angelegt wurden.

Des Weiteren kann Word dazu genutzt werden, um Textdokumente – vergleichbar mit einem XML-Editor – direkt mit XML-Tags anzureichern und als XML-Dokument abzuspeichern. Office 2010 unterstützt für diesen Vorgang XML-Schema-Definitionen zur Validierung von XML-Dokumenten. Dokumententyp-Definition werden allerdings nicht berücksichtigt (vgl. Vonhoegen 2011, S. 535).

[52] Vgl. hierzu: http://msdn.microsoft.com/de-de/office/bb906068.aspx [28.02.2014].

4.4 Strukturierung: XML-Schema-Definition (XSD)

Strukturierungsvorschriften für XML-Dokumente haben die Aufgabe, zu verhindern, dass die Dokumentinstanz – also der eigentlich Inhalt des Dokuments – beliebig verschachtelt und angereichert werden kann. XML stellt hierfür zwei Varianten zur Verfügung: die Dokumententyp-Definition (DTD) und die XML-Schema-Definition (XSD).

Die DTD stellt die ältere Variante dar. Die DTD-Syntax ist zwar Bestandteil der XML-Spezifikation, entspricht dabei aber nicht der eigentlichen XML-Syntax. Eine DTD präsentiert eine Reihe von Regeln, die verwendet werden, um Dokumente eines bestimmten Typs zu deklarieren. Sie besteht dabei aus der Deklaration von Elementen, Attributen, Entities, Notationen, Verarbeitungsanweisungen und Kommentaren. Über die reine Deklaration aller Bestandteile der Zielinstanz hinaus, beschreibt die DTD insbesondere auch deren Reihenfolge und Verschachtelung. In anderen Worten: Die DTD beschreibt die Struktur eines bestimmten Dokumententyps (vgl. Eckstein/Eckstein 2004, S. 25–26; Erlenkötter 2012, S. 105–107; Lobin 2010, S. 34–35; Vonhoegen 2011, S. 74–75).

Dokumententyp-Definitionen bringen eine Reihe von Nachteilen mit sich. Sie entsprechen nicht der XML-Syntax und sind somit in einer eigenen Sprache verfasst. XML-Parser müssen also zwei Sprachen beherrschen, um eine XML-Instanz auf ihre Gültigkeit zu überprüfen. Zudem können komplexe Wertbereiche und Elementbeziehungen mit DTD nur unzureichend definiert werden. Schließlich arbeiten DTD nicht mit Namensräumen. Diese sind aber für einem komplexes Content-Management unabdingbar (vgl. Erlenkötter 2012, S. 140). Aus diesem Grund wurde die weitaus komplexere und leistungsfähigere XSD entwickelt. XSD unterscheiden sich von DTD insbesondere durch die Verwendung von Namensräumen und Datentypen.

In der Praxis spielt es eine sehr große Rolle, dass die verwendeten Standards umfassend unterstützt und wiederverwendbar sind. Ein nachhaltiges Content-Management wird daher auf offene und bewährte Standards, wie es beispielsweise die TEI-Guidelines oder DocBook sind, zurückgreifen. Nur so kann gewährleistet werden, dass die verwendeten XML-Daten auch in anderen Kontexten interpretiert und verwendet werden können. Ein Workflow mit unterschiedlichen Formaten, Markups und Standards führt unweigerlich zu einem enormen Zusatzaufwand und zu Kompatibilitätsproblemen. Dennoch soll für die vorliegende Masterarbeit ein eigenes XML-Schema zur Anreicherung von Textdateien entwickelt werden. Es dient als Experiment und ist insbesondere dem Anspruch geschuldet, alle für ein Content-Management erforderlichen Stylesheets im Rahmen der Masterarbeit selbst zu entwickeln.

4.4.1 Der Aufbau einer XSD

Eine XSD hat den typischen Grundaufbau eines XML- Dokuments: Im Prolog befindet sich die XML-Deklaration. Im Bereich der XML-Daten benötigt sie mindestens ein Wurzelelement. Da das Schema ein spezielles XML-Dokument darstellt, können in einer XSD nicht mehr beliebige Elementbezeichnungen gewählt

werden. Dem Schema-Designer steht dagegen eine vorgegebene Klasse von Bezeichnungen zur Verfügung. XML-Dokumente, die einer XSD entsprechen, werden als Instanz dieser XSD bezeichnet und sind nicht nur wohlgeformt, sondern auch gültig.

```
<?xml version="1.0"?>                          <!-- 1 -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"       <!-- 2 -->
      xmlns:tst="http://www.ma-buwi-fh.de"
      targetNamespace="http://www.ma-buwi-fh.de"
      elementFormDefault="qualified"
      attributeFormDefault="qualified">

 <!-- Wurzelelement der Dokumentinstanz -->
 <xsd:element name="programm" type="tst:programm"/>          <!-- 3 -->

 <!--Definition Programm -->
 <xsd:complexType name="programm">                   <!-- 4 -->
  <xsd:all>
   <xsd:element name="warengruppe" type="tst:autorenstamm"/>
  </xsd:all>
 </xsd:complexType>

 <!-- Definition Autorenstamm -->
 <xsd:complexType name="autorenstamm">                <!-- 5 -->
  <xsd:all>
   <xsd:element name="autor" type="tst:werke"/>
  </xsd:all>
  <xsd:attribute name="gruppenname" use="required">          <!-- 6 -->
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="belletrsitik"/>
      <xsd:enumeration value="kinderjugendbuch"/>
     …
   </xsd:simpleType>
 </xsd:complexType>
…
</xsd:schema>
```

Listing 4: Die XML-Schema-Definition

Listing 6 beschreibt den Grundaufbau der für die vorliegende Masterarbeit erstellten Schema-Definition. Das komplette Dokument befindet sich im Anhang (vgl. Anhang Nr. 3).

(1) Die Schema-Definition ist ein wohlgeformtes XML-Dokument. Sie besitzt daher eine XML-Deklaration. Die Verwendete Version lautet auch hier 1.0. Ein Kodierungsverfahren ist nicht nötig, da die XSD sich auf die Strukturierung ihrer Instanz beschränkt und selbst keinen Inhalt in Textform aufnimmt.

(2) Das Wurzelelement einer XSD lautet immer `schema`. Dem Wurzelelement wurde außerdem das Präfix `xsd:` vorangestellt. Damit wird dem validierenden XML-Prozessor vermittelt, dass das Element `schema` zum Namensraum `http://www.w3.org/2001/XMLSchema` gehört. Der Namensraum und das Präfix `xsd` werden mit dem Attribut `xmlns` festgelegt. Das Attribut `targetNamespace` deklariert den Namensraum für die Instanz des Schemas. Dieser lautet: `http://www.ma-buwi-fh.de`. Das bedeutet, dass alle im XML-Dokument `kafka_verwandlung.xml` mit einem entsprechenden Präfix versehenen Elemente und Attribute dem Namensraum angehören, der in der XSD gestaltet wird. Die beiden letzten Attribute `elementFormDefault` sowie `attributeFormDefault` sind optional und nehmen den Wert `qualified` an. Dadurch wird bestimmt, dass alle in der Instanz vorkommenden Elemente und Attribute *QNames* sein müssen, damit die Instanz gültig ist (vgl. Erlenkötter 2012, S. 143–145).

(3) Als erstes Element wird das Wurzelelement der Instanz definiert. Die Definition von Elementen und deren Struktur in der Instanz erfolgt in der XSD mit dem vorgegeben Element `element`. Dessen Attribut `name` legt den Namen des Zielelements fest. Dieser lautet `programm`. Elemente können einfach und komplex sein. Einfache Typen bestehen aus Textzeichen. Komplexe Typen enthalten weitere Elemente oder Attribute (vgl. Kapitel 3.2.1). Das Wurzelelement `programm` ist ein komplexes Element. Der entsprechende Datentyp wird gesondert definiert. Der Verweis auf diese Definition erfolgt mithilfe des Attributs `type`. Dieses legt fest, dass das Element `programm` dem Datentyp `programm` entspricht.

(4) Die Definition des Datentyps `programm` erfolgt an dieser Stelle. Das Element `complexType` legt fest, dass es sich um einen komplexen Datentyp handelt. Das Attribut `name` benennt den Datentyp. Das Element `programm` soll in der Instanz ein weiteres Element mit dem Namen `warengruppe` enthalten. Dies wird mithilfe eines Kompositors erreicht. XSD unterscheiden zwischen drei Kompositoren: `sequence`, `choice` und `all`. Alle drei Kompositoren legen fest, wie die Struktur von Kindelementen eines Elternelements in der Instanz aussehen muss. Mit `sequence` wird eine feste Reihenfolge der Kindelemente festgelegt. Der Kompositor `all` erlaubt eine freie Elementgruppierung, mit `choice` kann in der Instanz genau ein Element aus einer Liste von mehreren definierten Elementen aufgeführt werden (vgl. Vonhoegen 2011, S. 146). Da das Element `programm` ohnehin nur ein Kindelement mit dem Namen `warengruppe` hat, genügt in diesem Fall der simple Kompositor `all`. Mittels des komplexen Datentyps `programm` wurde also bestimmt, dass das Element `programm` in der Instanz ein Kindelement mit dem Namen `warengruppe` besitzen muss, damit die Instanz gültig ist.

(5) Da das Element `warengruppe` ebenfalls ein Kindelement enthält, ist sein Datentyp auch komplexer Natur. Innerhalb einer Warengruppe werden alle in diesem Verlagsbereich publizierenden Autoren gefasst. Daher trägt der Datentyp des Elements `warengruppe` den Namen `autorenstamm`. Da die Autoren in der Instanz mithilfe von Attributen identifiziert werden, genügt auch in diesem Fall ein Kindelement mit dem Namen `autor`. Entsprechend wurde auf den Kompositor `all` zurückgegriffen.

(6) Der Datentyp des Elements `warengruppe` ist aber nicht nur deshalb komplex, weil es ein Kindelement besitzt. Zusätzlich verfügt das Element auch über ein Attribut mit dem Namen `gruppenname`. Die genaue Warengruppe soll mit diesem Attribut festgelegt werden. Mit `use="required"` wird bestimmt, dass das Attribut in der Instanz immer angegeben sein muss. Attribute haben immer einen simplen Datentyp. Das Inhaltsmodell des Attributs `gruppenname` erfolgt daher innerhalb des Elements `simpleType` (vgl. Vonhoegen 2011, S. 605). Der einfache Datentyp lautet `string`. Es handelt sich dabei um einen vorgegebenen Datentyp, der festlegt, dass das Attribut eine Abfolge von beliebigen Unicode-Zeichen enthält. Der Wert des Attributs kann demnach verschiedene Namen annehmen. Da nun jedoch die Warengruppensystematik aus wenigen Gruppen besteht, empfiehlt es sich, die möglichen Attributswerte einzuschränken. Dies geschieht mithilfe des Elements `restriction`, das wiederum eine beliebige Menge von möglichen Attributswerten als Kindelemente besitzt. Diese werden mit dem Element `enumeration` festgelegt (vgl. Erlenkötter 2012, S. 170–171). Auf Grund dieser Einschränkung kann das Attribut `gruppenname` nun die Werte `belletristik` oder `kinderjugendbuch` annehmen. Namen, die nicht mit `enumeration` vorgegeben wurden, sind in der Instanz allerdings nicht erlaubt.

Listing 6 zeigt, dass eine XML-Schema-Definition aus einzelnen Blöcken besteht. Jeder Block – mit Ausnahme des ersten Blocks – stellt einen Datentyp dar. Die Datentypen sind wiederum verschachtelt, da sie selbst aus anderen, komplexen Datentypen bestehen. Der Verweis innerhalb der Datentypen geschieht mithilfe des Attributs `type`, dessen Wert den Namen eines Datentyps annehmen muss.

Jeder Datentyp repräsentiert Elemente aus der Instanz der XSD. Elemente, die selbst Elemente oder Attribute enthalten, werden in der XSD durch komplexe Datentypen repräsentiert. Attribute und Elemente, die nur Text enthalten, werden in der XSD mit simplen Datentypen beschrieben. Damit ein XML-Dokument einer XSD entsprechend validiert werden kann, müssen beide Dateien abschließend verknüpft werden. Das geschieht innerhalb des Wurzelelements `programm` im XML-Dokument `kafka_verwandlung.xml` und wurde in Kapitel 4.3.2 bereits dargelegt.

4.4.2 Designvarianten

Bei der Konzeption von XSD bestehen verschiedene Möglichkeiten in der Strukturierung und Anordnung der Datentypen. Dabei werden insbesondere zwei Modelle unterschieden: das Babuschka-Modell und das Stufenmodell.

Beim Babuschka-Modell orientiert sich der Schema-Designer am XML-Dokument und verschachtelt die Datentypen gemäß der Verschachtelung der Elemente in der Instanz. Es gibt nur ein globales Element und alle lokalen Typendefinitionen befinden sich innerhalb des globalen Elements. Da die Schachtelung in diesem Fall viele Stufen erreicht und nur schwer lesbar ist, besteht die Gefahr, dass der Überblick schnell verloren geht.

Um den Überblick zu wahren, bietet sich die Möglichkeit des Stufenmodells an. Dieses zerlegt die komplexe Baumstruktur des Babuschka-Modells in seine Einzelteile und definiert diese auf einer globalen Ebene, also auf einer Ebene unterhalb des Wurzelelements `schema`. Sukzessive können mit diesem Modell alle simplen Datentypen definiert werden. Durch Referenzen auf die simplen Datentypen können komplexe Datentypen in einer zweiten Stufe zusammengefasst werden. Auf der obersten Stufe werden die komplexen Datentypen zu Sequenzen gebündelt. Dieses Verfahren ähnelt in der Vorgehensweise sehr an eine Modellierung mit Dokumententyp-Definitionen.

Die für die vorliegende Masterarbeit geschriebene Schema-Definition stellt eine Mischform beider Designvarianten dar. Gemäß dem Babuschka-Modell orientieren sich die komplexen Datentypen in ihrer Anordnung am XML-Dokument. Um die Übersicht zu wahren, sind alle komplexen Datentypen allerdings gesondert definiert und mittels `type`-Attributen verknüpft. Man spricht in diesem Fall von benannten Typen (vgl. Vonhoegen 2011, S. 161). Dieses Vorgehen erinnert an die Verwendung von Referenzen im Stufenmodell.

4.4.3 Namensräume in XML

Namensräume spielen, wie bereits erwähnt, für komplexe XML- und insbesondere XSD-Dokumente eine signifikante Rolle. Namensräume wurden eingeführt, um polyseme Bezeichnungen in XML-Dokumenten zu vermeiden. Zur Veranschaulichung vergleicht HELMUT ERLENKÖTTER Namensräume mit Familiennamen. Ein Namensraum trägt demnach einen eindeutigen Namen. Innerhalb des Namensraums muss ebenfalls jeder Name eindeutig sein. In verschiedenen Namensräumen dürfen dieselben Namen dagegen auftreten. Während die Elementbezeichnung dabei den Vornamen darstellt, ist der Namensraum mit dem jeweiligen Familiennamen vergleichbar (vgl. Erlenkötter 2012, S. 143). Schaubild 11 verdeutlicht das Konzept von Namensräumen.

Namensraum 1	Namensraum 2
`xmlns="http://www.namensraum-1.de"`	`xmlns:ns2="http://www.namensraum-2.de">`
`<buch>`	`<ns2:buch>`
`<verfasser>`	`<ns2:verfasser>`
`<titel>`	`<ns2:titel>`

Abbildung 11: XML-Namensräume

Alle in den Namensräumen genannten Elemente können trotz ihrer identischen Bezeichnung innerhalb eines XML- Dokuments aufgeführt werden, da sie sich durch ihre Zugehörigkeit zu den jeweiligen Namensräumen unterscheiden. Beide Elemente `<verfasser>` sind daher *QNames*. Namensräume sind somit eine Ansammlung von Namen, die einem bestimmten Gegenstandsbereich zugeordnet werden können.

XML-Namensräume werden in aller Regel über *Uniform Resource Locators* (URL) identifiziert. Mittels einer URL wird ein Namensraum mit einem für jeden XML-Prozessor eindeutigen Identifikator versehen. XML-Prozessoren prüfen allerdings nicht nach, ob bei der angegebenen URL tatsächlich ein entsprechendes Regelwerk hinterlegt wurde. Die verwendete URL muss folglich nicht tatsächlich existieren, sondern stellt lediglich eine Formalität zur Identifizierung von Namensräumen dar.

Die Reichweite eines Namensraums hängt davon ab, wo er in einem XML-Dokument verwendet wird. So können Namensräume für das gesamte Wurzelelement gültig sein, wenn sie dort deklariert wurden. Es ist auch möglich, Namensräume nur in einem tiefer verwurzelten Element zu gebrauchen. Die Deklaration von Namensräumen geschieht in Form eines Attributs des auszuzeichnenden Elements:

```
<programm
    xmlns="http://www.namensraum-1.de"
    xmlns:ns2="http://www.namensraum-2.de">
```

Für das Wurzelelement `<programm>` wurden zwei Namensräume deklariert. Die Deklaration geschieht über das reservierte Attribut `xmlns`, danach folgt die URL des Namensraums. Während der erste Namensraum im Beispiel als Standardnamensraum deklariert wurde, da ihm kein Präfix zugeordnet ist, verfügt der zweite Namensraum über das Präfix `ns2`. Alle innerhalb des Wurzelelements verschachtelten Elemente ohne Präfix gehören folglich zum Namensraum 1, während alle mit Präfix ausgestatteten Elemente Namensraum 2 zugehörig sind (vgl. Erlenkötter 2012, S. 143–145; Eckstein/Eckstein 2004, S. 51–53; Vonhoegen 2011, S. 66–70).

```
<programm

   xmlns="http://www.namensraum-1.de"

   xmlns:ns1="http://www.namensraum-1.de"              <!-- 1 -->

   xmlns:ns2="http://www.namensraum-2.de">             <!-- 2 -->

   . . .

   <buch>                                <!-- 3 -->

      <ns2:verfasser>Franz Kafka</verfasser>          <!-- 4 -->

      <titel ns1:name="Die Verwandlung"/>            <!-- 5 -->

   </buch>

</programm>
```

Listing 4: Namensräume einsetzen

Listing 7 verdeutlicht, wie Elemente und Attribute in einem XML-Dokument einem Namensraum zugeordnet werden können:

(1) Im Wurzelelement `programm` wurden zwei Namensräume deklariert. Der erste Namensraum ist der Standardnamensraum. Er gilt für alle nachfolgenden Elemente ohne Präfix. Eine Sonderregel gilt für Attribute. Attribute erben nicht den Namensraum des Elementes, zu dem sie gehören, sondern müssen in jedem Fall mittels eines Präfixes einem Namensraum zugeordnet werden. Für Namensraum 1 wird daher noch das Präfix `ns1` definiert.

(2) Der zweite Namensraum ist ein untergeordneter Namensraum. Er gilt nur für solche Elemente, die über das entsprechende Präfix `ns2` verfügen.

(3) Das Element `buch` hat kein Präfix und gehört daher zum Standardnamensraum `http://www.namensraum-1.de`.

(4) Das Element `verfasser` hat dagegen das Präfix `ns2` und lässt sich daher dem Namensraum `http://www.namensraum-2.de` zuordnen.

(5) `titel` ist im Beispiel ein leeres Element. Der Name des Buches wird daher über das Attribut `name` festgelegt. Das Element `titel` gehört zwar automatisch zum Standardnamensraum, überträgt diese Zugehörigkeit allerdings nicht an sein Attribut. Das Attribut muss daher mittels des Präfixes `ns1` dem Namensraum 1 zugeordnet werden.

4.4.4 Schematische Darstellung der Schema-Definition

Die Schema-Definition gibt den gesamten Aufbau ihrer Instanz – dem XML-Dokument – vor. Sie muss nicht nur wohlgeformt und fehlerfrei sein, vor allem ihr logischer Aufbau muss gut konzipiert sein. Abbildung 12 stellt den Aufbau der verwendeten XSD schematisch dar.

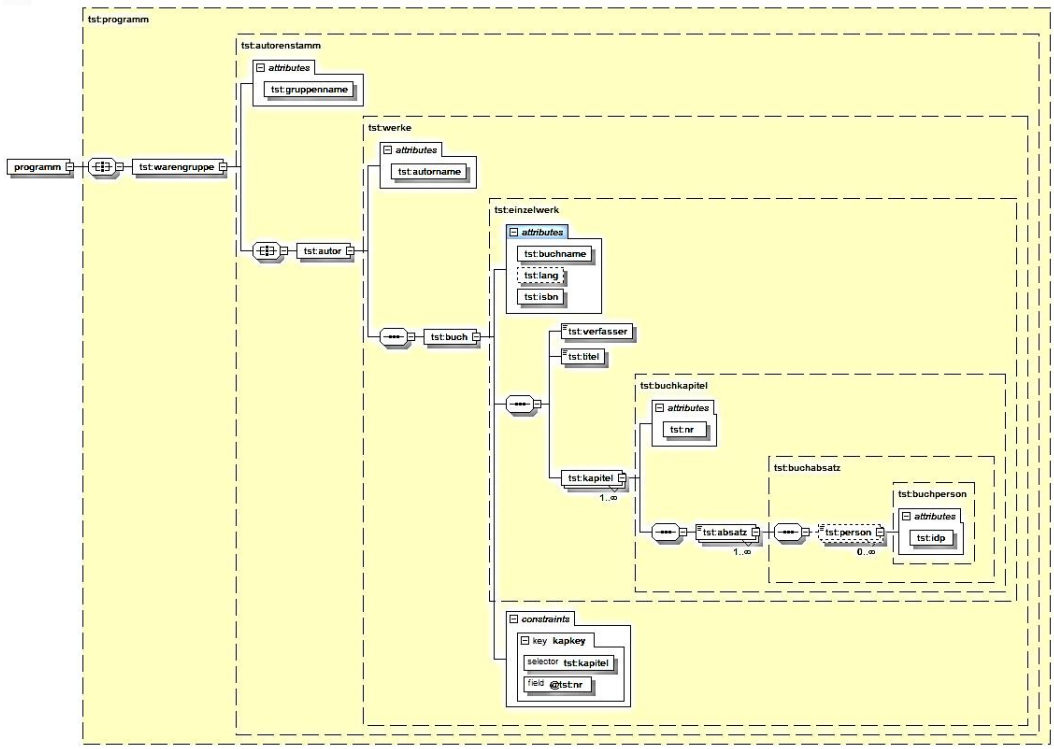

Abbildung 12: Schematische Darstellung der Schema-Definition mit XMLSpy

Um die Abbildung richtig zu interpretieren, muss der Zusammenhang zwischen Elementen und deren Beschreibung im Schema verstanden werden. Elemente werden im Schema – wie bereits erwähnt – mit komplexen Datentypen beschrieben. Jedes Element, das im XML-Dokument vorkommt, steht in der Abbildung in einem kleinen weißen Kasten. Die dem Element zugeordneten Datentypen werden durch große, gelb eingefärbte Bereiche abgebildet. Innerhalb der Datentypen befinden sich wiederum hierarchisch nachgestellte Kindelemente. Kindelemente und simple Datentypen in Form von Attributen zweigen von den Elementen, die sie beinhalten, ab. Durch diese Verschachtelung entsteht eine horizontal ausgerichtete Baumstruktur.

Das Wurzelelement heißt `programm`. Es wird durch den komplexen Datentyp `programm` definiert, der durch den äußersten, alle andern Bereiche umfassenden Block dargestellt wird. Das Programm des Verlags teilt sich in verschiedene Warengruppen auf. Jede Warengruppe verfügt über einen Autorenstamm, der alle Autoren der jeweiligen Gruppe umfasst. Das entsprechende Element im XML-Dokument heißt `autor`. Jeder Autor wiederum verfügt über eine Anzahl von Büchern, die vom Verlag publiziert werden. Der komplexe Datentyp `werke` fasst diese zusammen. Das entsprechende Element im XML-Dokument heißt `buch`. Einzelne Bücher werden mithilfe des komplexen Datentyps `einzelwerk` beschrieben. Dieser Datentyp bestimmt, dass jedes Buch die Kindelemente `verfasser`, `titel` und `kapitel` enthalten muss, die ebenfalls mithilfe von entsprechenden Datentypen bestimmt werden. Einzelne Kapitel werden dem Schema entsprechend durch den komplexen Datentyp `buchkapitel` definiert und in Absätze aufgeteilt. Das Attribut `kapkey` dient der eindeutigen Benennung der einzelnen Kapitel (vgl. Vonhoegen 2011, S. 151). Die letzten beiden komplexen Datentypen `buchabsatz` und

buchperson beschreiben alle Absätze des XML-Dokuments sowie alle darin genannten Personen. Die entsprechenden Elemente im XML-Dokument lauten **absatz** und **person**.

Wichtig hierbei ist, dass die verwendeten Bezeichnungen sehr neutral sind, damit die konzipierte XSD auf alle im Verlag publizierten Bücher angewendet werden kann, ohne dass das Schema angepasst werden muss. Nur so kann eine automatische Weiterverarbeitung mit XSLT und XSL-FO gewährleistet werden (vgl. Kapitel 4.3.2; Abbildung 10).

4.5 Zielformat I: XHTML

Die ersten zwei Schritte des Workflows wurden in den vorangegangen Kapiteln erläutert. Es exisitiert zu diesem Zeitpunkt ein valides XML-Dokument **kafka_verwandlung.xml**, dessen Aufbau und Inhalt durch die XML-Schema-Definition **kafka_struktur.xsd** vorgeben ist. Der nächste Schritt innerhalb des Workflows sieht die Transformation des XML-Dokuments in die gewünschten Zielformate vor.

Das erste Zielformat heißt *Extensible Hypertext Markup Language* (XHTML). XHTML ist eine auf XML basierte Weiterentwicklung von HTML 4.0. Mit XHTML können Texte, Bilder und Hyperlinks in Webdokumenten ausgezeichnet werden: »XHTML 2 is a general-purpose markup language designed to represent documents for a wide range of purposes across the World Wide Web« (W3C 2014, XHTML 2.0, Abstract). XHTML wurde von W3C entwickelt. Die modularisierte Version 1.1 liegt seit November 2010 vor.[53] Der entscheidende Unterschied zwischen HTML und XHTML besteht darin, dass jedes XHTML-Dokument zugleich auch ein gültiges XML-Dokument darstellt. Die Grundvoraussetzung für jedes XHTML-Dokument ist daher seine Wohlgeformtheit (vgl. Vonhoegen 2011, S. 307–308).

4.5.1 Der Aufbau eines XHTML- Dokuments

Da XHTML-Dokumente in der XML-Syntax formuliert sind, gleicht ihr Aufbau denen von gewöhnlichen XML-Dokumenten.

[53] Vgl. hierzu: http://www.w3.org/TR/xhtml11/ [06.03.2014]

```
<?xml version="1.0" encoding="UTF-8"?>                <!-- 1 -->

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"           <!-- 2 -->

  "http://www.w3.org/TRxhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">              <!-- 3 -->

  <head>                                  <!-- 4 -->

    <title>…</title>

  </head>

  <body>                                  <!-- 5 -->

    <p>…</p>

  </body>

</html>
```

Listing 5: Aufbau eines XHTML-Dokuments

Listing 8 zeigt den Rohaufbau eines XHTML-Dokuments.

(1) Wie jedes wohlgeformte XML-Dokument besitzt auch dieses eine XML-Deklaration mit der Benennung der verwendeten Version. Da das XHTML Dokument Inhalte in Form von Textzeichen aufnimmt, muss dem interpretierenden Prozessor das Kodierungsverfahren mitgeteilt werden. In diesem Fall wird das Standardformat `UTF-8` gewählt.

(2) Dahinter muss eine DOCTYPE-Deklaration folgen. Diese verknüpft das XHTML-Dokument mit einer Dokumententyp-Definition. Der nachfolgende Pfad sowie die URL sind vom W3C vorgegeben.

(3) Das Wurzelelement lautet immer `html`. Darin muss außerdem der für XHTML reservierte Namensraum mittels des bekannten `xmlns`-Attributs genannt sein. Der Namensraum lautet `http://www.w3.org/1999/xhtml`.

(4) Der sogenannte *Header* wird mit dem Element `head` dargestellt. Er enthält alle Kopfdaten, wie etwa Angaben zum Titel, Autor oder weitere Metadaten.

(5) Das `body`-Element umfasst den eigentlich anzuzeigenden Inhalt. Dieser kann in Form von Text, Verweisen oder anderen multimedialen Daten vorliegen.

4.5.2 Die Bausteine von XSLT

Um das ursprüngliche XML-Dokument in sein Zielformat zu transformieren, wurde die Transformationssprache XSLT entwickelt. XSLT greift mithilfe von XPath auf die Elemente des XML-Dokuments zu und transformiert diese in XHTML-Elemente. Durch die Transformation entsteht ein wohlgeformtes XHTML-Dokument.

Für die Transformation benötigt der XSLT-Prozessor ein Stylesheet, das den Ablauf der Transformation bestimmt. Stylesheets bestehen aus zwei unterschiedlichen Bestandteilen. Die sogenannten *Literal Result Elements* (LRE) werden ohne Veränderung – also literal oder buchstäblich – direkt in das Ergebnisdokument kopiert. LREs befinden sich ausschließlich im Stylesheet und nicht im Quelldokument. Um auf das Quelldokument zuzugreifen, verfügt XSLT darüber hinaus über spezielle Instruktionen. XSL-Instruktionen sind daher der zweite Bestandteil von XSLT-Stylesheets. Sie können genutzt werden, um Inhalte aus dem Quelldokument zu kopieren oder neue Inhalte hinzuzufügen. XSL-Instruktionen erkennt man an dem Präfix `xsl`. Abbildung 13 veranschaulicht den Zusammenhang von LREs und XSL-Instruktionen:

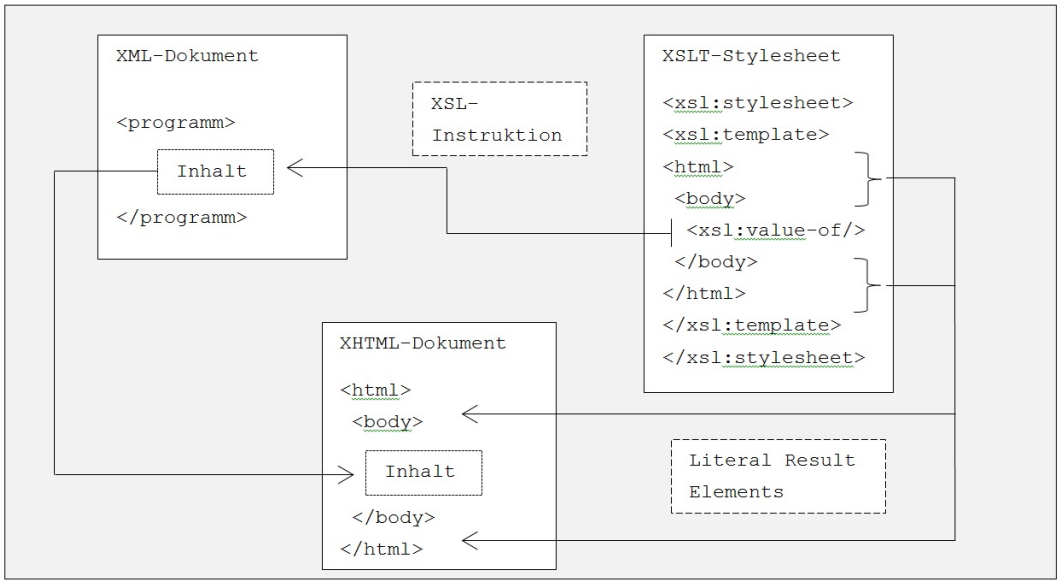

Abbildung 13: XSL-Instruktionen und Literal Result Elements nach BONGERS 2008, S. 33

Das Grundgerüst des XHTML-Dokuments liegt – wie in Abbildung 13 deutlich wird – bereits in Form von LREs im Stylesheet vor. Die LREs werden im Transformationsprozess ohne Veränderung in das Zieldokument kopiert. Anschließend setzt der XSLT-Prozessor die XSL-Instruktionen um. Mit `xsl:value-of` erteilt das Stylesheet dem Prozessor den Befehl, gezielt Inhalte aus dem Quelldokument in das Zieldokument zu kopieren. Es ist im Stylesheet also erlaubt, LREs mit Instruktionen zu vermischen. Dadurch kann ein XML-Dokument in eine beliebig komplexe XHTML-Seite transformiert werden (vgl. Bongers 2008, S. 31–34).

Der Rumpf eines Stylesheets bildet das Element `xsl:stylesheet`. Es ist das Wurzelelement des Stylesheets und beinhaltet alle verschachtelten Anweisungen und LREs. Im Wurzelelement wird die verwendete XSLT-Version festgelegt und alle benötigten Namensräume vereinbart (vgl. Bongers 2008, S. 38). Unmittelbare Kindelemente von `xsl:stylesheet` werden als *Toplevel-Elemente* bezeichnet. XSLT unterscheidet zwischen zwölf solcher Elemente. Die zwei wichtigsten sind `xsl:output` und `xsl:template`.

Mit `xsl:output` kann das Zielformat festgelegt werden. Es verfügt über das Attribut `method=""`, das die Werte `xml`, `html`, `xhtml` und `text` annehmen kann (vgl.

Vonhoegen 2011, S. 251). Soll das Zielformat eine XHTML-Datei sein, so muss das Attribut `method` den Wert `xhtml` haben. Zusätzlich müssen die zwei Attribute `doctype-public` und `doctype-system` hinzugefügt werden, damit der XSLT-Prozessor die für XHTML notwendigen Dokumententyp-Definitionen in die Zieldatei einfügen kann. Außerdem muss im Wurzelelement des Stylesheets der XHTML-Namensraum als Standardnamensraum hinzugefügt werden (vgl. Bongers 2008, S. 49–50).

Das wichtigste Toplevel-Element heißt `xsl:template`. Es bildet das Gerüst des Stylesheets. Ein Stylesheet besteht in der Regel aus mehreren *Templates*, wobei jedes dieser Templates für einen Teil des Ergebnisdokuments zuständig ist. Templates verfügen über `match`-Attribute. Mithilfe dieser Attribute kann der Aufruf eines Templates vom Prozessor gesteuert werden. Das `match`-Attribut beinhaltet ein Vergleichsmuster (engl. *Pattern*), das bestimmt, wann das Template ausgeführt wird. Das Vergleichsmuster wird mit XPath formuliert. So wird das Template `<xsl:template match="/">` dann ausgeführt, wenn der XSLT-Prozessor ein passendes Element im Quelldokument findet. Das Vergleichsmuster `"/"` ist ein Sonderfall, da es den Dokumentknoten und damit das gesamte Quelldokument ‚an sich' aktiviert (vgl. Kapitel 4.5.3). Wenn ein Template ein match-Attribut besitzt, bezeichnet man es als Template-Regel (vgl. Bongers 2008, S. 39–41).

4.5.3 Der Aufbau eines XSLT-Stylesheets

Ein solches XSLT-Stylesheet kann nun genutzt werden, um das Quelldokument `kafka_verwandlung.xml` in das Zieldokument `kafka_verwandlung.xhtml` zu transformieren.

```
<?xml version="1.0" ?>
<xsl:stylesheet version="2.0"                      <!-- 1 -->
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:fn="http://www.w3.org/2005/xpath-functions"
 xmlns:tst="http://www.ma-buwi-fh.de"
 xmlns="http://www.w3.org/1999/xhtml">

<xsl:output method="xhtml"                       <!-- 2 -->
 encoding="UTF-8"
 indent="yes"
 doctype-public="-//W3C//DTD XHTML 1.1//EN"
 doctype-system="http://www.w3.org/TRxhtml11/DTD/xhtml11.dtd"/>
...

<xsl:template match="/">                        <!-- 3 -->
 <html xml:lang="de" lang="de">                     <!-- 4 -->
  <!-- Angaben zum Titel -->
  <head>                                 <!-- 5 -->
   ...
  </head>
  <!-- Anzuzeigender Inhalt -->
  <body>                                 <!-- 6 -->
   <!-- Reihenfolge der Ausgabe -->
   <xsl:apply-templates select="tst:programm/tst:literatur/
                tst:autor/tst:buch/tst:verfasser"/>  <!-- 7 -->
   ...
  </body>
 </html>
</xsl:template>

<!-- Ausgaberegel Verfasser -->
<xsl:template match="tst:verfasser">                 <!-- 8 -->
<h1 id="{generate-id()}"><xsl:value-of select="self::node()"/></h1>
</xsl:template>

...
```

Listing 6: Das XSLT-Stylesheet

Listing 9 zeigt einen Ausschnitt des für die vorliegende Masterarbeit geschriebenen XSLT-Stylesheets. Das komplette Stylesheet liegt im Anhang vor (vgl. Anhang Nr. 4).

(1) Das Wurzelelement heißt `xsl:stylesheet`. Es verfügt über die Versionsangabe `2.0` sowie alle im Stylesheet benötigten Namensräume. Der für XSL reservierte Namensraum lautet `http://www.w3.org/1999/XSL/Transform`. Der zweite Namensraum wir für XPath-Funktionen benötigt und lautet `http://www.w3.org/2005/xpath-functions.` Ihm wird das Präfix `fn` zugewiesen. Der dritte Namensraum ist der in der XSD definierte Namensraum `http://www.ma-buwi-fh.de`. Er wird benötigt, um auf die Elemente im XML-Dokument zuzugreifen, da es sich dabei um *QNames* – also einem Namensraum zugehörige Elemente – handelt. Der letzte Namensraum ist der Standardnamensraum und lautet `http://www.w3.org/1999/xhtml`. Wie bereits erläutert, wird dieser Namensraum benötigt, um ein gültiges XHTML-Dokument zu produzieren (vgl. Bongers 2008, S. 43–45, S. 51).

(2) Das `output`-Element bestimmt das Zielformat der Transformation. Da das gewünschte Format XHTML ist, hat das `method`-Attribut den entsprechenden Attributwert. Mittels des `encoding`-Attributs wird der Kodierungsstandard `UTF-8` festgelegt. Mit `indent="yes"` wird bestimmt, dass die Struktur des Zieldokuments durch Einrückung deutlich gemacht wird. Die letzten beiden Attribute `doctype-public` und `doctype-system` fügen dem Ergebnisdokument die Verweise auf die für XHTML benötigte DTD hinzu (vgl. Bongers 2008, S.49–53).

(3) Der wichtigste Baustein des Stylesheets lautet `<xsl:template template="/">`. Diese Templateregel trägt in ihrem Rumpf etwaige Metadaten, die Struktur des Ausgangsdokuments in Form von LREs und die Reihenfolge der Ausgabe aller später folgenden Templateregeln. Dadurch, dass sich das Vergleichsmuster auf den Dokumentknoten bezieht, gilt das Template für das gesamte Quelldokument. Der Startpunkt einer XSLT-Transformation ist immer der Dokumentknoten. Der XSLT-Prozessor speichert den Dokumentknoten und verarbeitet ihn mit der dazu passenden Templateregel (vgl. Vonhoegen 2011, S. 256).

(4) Sukzessive werden nun alle in der Templateregel enthaltenen LREs ins Ergebnisdokument kopiert. Da es sich um ein XHTML-Dokument handeln soll, lautet das spätere Wurzelelement `html`. Die Attribute `xml:lang` und `lang` können in einem anschließenden Verarbeitungsprozess Aussage über die im Dokument verwendete Sprache machen, sind aber optional.

(5) Das erste Kindelement von `html` lautet `head`. Es trägt alle Metainformationen – insbesondere Angaben zum Titel – in sich. Der Übersicht wegen wird das Element – im Gegensatz zum tatsächlichen Stylesheet – im Listing ohne Inhalt dargestellt.

(6) Wichtiger ist das zweite Kindelement von `html`. Es heißt `body` und bestimmt alle im XHTML-Dokument anzuzeigenden Inhalte. Der Struktur des Stylesheets geschuldet, werden die konkreten Ausgaberegeln in weitere Templateregeln ausgelagert. Im Ergebnisdokument werden allerdings alle Inhalte im `body`-Element angezeigt.

(7) Dieser Umstand liegt darin begründet, dass das `body`-Element eine Reihe von XSL-Instruktionen mit dem Namen `apply-templates` beinhaltet. Diese Instruktionen können als Stellvertreter für spätere im Stylesheet angesiedelte Templateregeln betrachtet werden. Die Instruktion `xsl:apply-templates` aktiviert weitere Templateregeln mithilfe eines `select`-Attributs, das einen XPath-Ausdruck zur Auswahl der entsprechenden Knoten besitzt. In diesem Fall gibt daher die `apply-templates`-Instruktion einen Befehl an den XSLT-Prozessor weiter, sich an eine spätere Templateregel zu wenden, und diese an der jetzigen Stelle umzusetzen (vgl. Bongers 2008, S. 84–85).

(8) Diese spätere Templateregel befindet sich an dieser Stelle des Stylesheets. Mittels des `match`-Attributs wird bestimmt, dass sie für das Element `tst:verfasser` des XML-Dokuments gilt. Findet der XSLT-Prozessor im XML-Dokument ein für das `match`-Attribut passendes Element, wird diese Templateregel aktiv. In diesem Fall wird mit der Instruktion `xsl:value-of` bestimmt, dass der Inhalt der Elements `tst:verfasser` im Zieldokument in ein Element `<h1>` kopiert wird. Dadurch entsteht im XHTML-Dokument eine Überschrift mit dem Namen des jeweiligen Autors. Mit `id="{generate-id()}"` wird dem Element außerdem eine eigene Nummer zugewiesen. Diese wird später für die Navigation im Zieldokument benötigt. Anschließend werden sukzessive weitere Template-Regeln abgearbeitet und im `body`-Element umgesetzt.

Listing 9 zeigt nur einen Ausschnitt des gesamten Stylesheets. Der tatsächliche Transformationsprozess ist komplexer und variiert je nach Design des Stylesheets. Der Transformationsprozess wurde schematisch bereits in Kapitel 3.2.2 dargelegt und soll in seinem ganzen Umfang an dieser Stelle nicht besprochen werden. Wichtig ist jedoch, dass beide an der Transformation beteiligten Dokumentenbäume Knoten für Knoten abgearbeitet werden. Findet der XSLT-Prozessor mithilfe eines XPath-Vergleichsmusters eine Templateregel für den aktuellen Knoten, wird diese aktiviert und der Knoten verarbeitet. Der aktuelle Knoten wird als Kontextknoten bezeichnet und durchläuft das Dokument. Relative XPath-Ausdrücke müssen daher immer aus dessen Perspektive formuliert werden. Exisitiert für den Kontextknoten keine Templateregel, wird er in der Regel übersprungen und nicht verarbeitet (vgl. Bongers 2008, S. 84–86). Wie gezeigt werden konnte, spielt insbesondere die Instruktion `xsl:apply-templates` eine wichtige Rolle. Sie wird dann benötigt, wenn mehrere Templateregeln vorliegen und fungiert in diesem Fall wie eine Aktivierungskette. Mittels des `select`-Attributs stellt die Instruktion nachfolgenden Templates eine Knotensequenz zusammen, die dann abgearbeitet werden können. Fehlen in diesem Fall weitere Templates, so werden – je nach Knotensequenz – nur die Stringwerte der zu verarbeitenden Elemente ins Ergebnisdokument kopiert. XSLT aktiviert diesbezüglich vorgegebene *Default*-Templateregeln (vgl. Bongers 2008, S. 101–105).

4.6 Zielformat II: PDF

Das zweite Zielformat ist das *Portable Document Format* (PDF). Es ist ein plattformunabhängiges Dateiformat, das von *Adobe Systems* entwickelt wurde. Seit Juli 2007 liegt die Version 1.7 vor.[54] PDF spielt insofern eine wichtige Rolle, da es sowohl für den Printbereich als auch für digitale Publikation genutzt werden kann. Um das XML-Dokument `kafka_verwandlung.xml` in das PDF-Dokument `kafka_verwandlung.pdf` zu transformieren, wird – wie auch schon bei der Transformation in XHTML – ein XSL-Stylesheet benötigt, das einem XSLT-Stylesheet in vielen Belangen ähnelt.

4.6.1 Die Bausteine von XSL-FO

Wie bereits in Kapitel 3.2.4 beschrieben, besteht ein XSL-FO-Stylesheet aus Formatierungsobjekten, die typografische Kategorien repräsentieren. Der Inhalt des XML-Dokuments wird in Form von rechteckigen Bereichen formatiert. Für den Formatierungsprozess nutzt XSL-FO dieselben Instruktionen wie auch XSLT. Für Vergleichsmuster werden außerdem ebenfalls XPath-Ausdrücke herangezogen. Zusätzlich werden allerdings Formatierungsanweisungen – auch *Traits* genannt – an einen XSL-FO-Prozessor weitergereicht. Während die Gestaltung von XHTML-Dokumenten mittels gesonderter CSS-Dokumente umgesetzt wird, beinhaltet ein XSL-FO-Stylesheet diese bereits.

Die Bausteine von XSL-FO erkennt man an ihrem Präfix `fo`. Das Wurzelelement der Layoutvorlage lautet `fo:root`. Es ist das erste Formatierungsobjekt, das in einem XSL-Stylesheet zugelassen wird, und die Wurzel des in Kapitel 3.2.4 erwähnten Baums der Formatierungsobjekte. Die Kindelemente von `fo:root` lauten `fo:layout-master-set` und `fo:page-sequence`. Mit `fo:layout-master-set` lassen sich Seitenvorlagen festlegen, die den Aufbau und die Abfolge von Buchseiten definieren. Der konkrete Seitenverlauf wird mit `fo:page-sequence` bestimmt. Die Formatierungsanweisungen werden im Detail von `fo:block`- und `fo:inline`-Elementen umgesetzt (vgl. Krüger / Welsch 2007, S. 27–30; Vonhoegen 2011, S. 339–341). Im Vergleich zu XSLT wird deutlich, dass das Raummodell von XSL-FO auf Buchseiten setzt. Der Ablauf einer Transformation mit XSL-FO verläuft daher im Grund wie eine Transformation mit XSLT, mit dem Unterschied, dass die Elemente des Quelldokuments formatiert und anschließend auf die im `fo:layout-master-set` bestimmten Seitenvorlagen verteilt werden.

4.6.2 Der Aufbau eines XSL-FO-Stylesheets

Mithilfe des für die Masterarbeit geschriebenen XSL-FO-Stylesheets kann jedes XML-Dokument, das im Aufbau `kafka_verwandlung.xml` gleicht, in ein PDF-Dokument umgewandelt werden.

[54] Vgl. hierzu: http://www.adobe.com/devnet/pdf/pdf_reference.html [07.03.2014].

```
<?xml version="1.0" encoding="ISO-8859-1">

<xsl:stylesheet version="2.0"                          <!-- 1 -->

 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

 xmlns:fn="http://www.w3.org/2005/xpath-functions"

 xmlns:tst="http://www.ma-buwi-fh.de"

 xmlns:fo="http://www.w3.org/1999/XSL/Format">


<xsl:template match="/">                           <!-- 2 -->

 <!-- Wurzelelement Layoutvorlage>

 <fo:root>                                  <!-- 3 -->

  <fo:layout-master-set>                          <!-- 4 -->

   <!-- Seitenvorlagen -->

   <!-- Text -->

   <fo:simple-page-master master-name="standardseite"          <!-- 5 -->

    page-height=300mm" page-width="210mm"

    margin-top="10mm" margin-left="30mm"

    margin-bottom="10mm" margin-right="30mm">

    <fo:region-body region-name="textkorpus"               <!-- 6 -->

    margin-top="20mm" margin-left="5mm"

    margin-bottom="20mm" margin-right="5mm">

    ...

   </fo:simple-page-master>

   <!-- Seitenverlaufsvorlagen -->

    ...

   <fo:page-sequence-master master-name="kapitel">            <!-- 7 -->

    <fo:repeatable-page-master-alternatives>

     <fo:conditional-page-master-reference

      master-reference="standardseite"/>

    </fo:repeatable-page-master-alternatives>

   </fo:page-sequence-master>

  </fo:layout-master-set>

  ...
```

```
  <!-- Konkreter Seitenverlauf -->

  ...

  <!-- Kapitel -->

  <fo:page-sequence master-reference="kapitel">              <!-- 8 -->

   <!-- Fließtext -->

   <fo:flow flow-name="textkorpus">

    <xsl:apply-templates select="//tst:kapitel"/>

   </fo:flow>

  </fo:page-sequence>

 </fo:root>

</xsl:template>


<!-- Templates -->

<!-- Template Kapitel -->

<xsl:template match="tst:kapitel">                      <!-- 9 -->

 <fo:block font-family="Amatic Bold" font-size="24pt" ...>         <!-- 10 -->

   Kapitel <xsl:value-of select="./@tst:nr"/>

 </fo:block>

 <xsl:apply-templates select="tst:absatz"/>

 <xsl:if test="fn:not(fn:position()=fn:last())">             <!-- 11 -->

  <fo:block page-break-before="always"/>

 </xsl:if>

</xsl-template>

...
```

Listing 7: Das XSL-FO-Stylesheet

Listing 10 zeigt einen Ausschnitt des XSL-FO-Stylesheets, das für die vorliegende Masterarbeit geschrieben wurde. Das gesamte Dokument befindet sich im Anhang (vgl. Anhang Nr. 5).

(1) Da es sich grundsätzlich um ein XSL-Stylesheet handelt, heißt das Wurzelelement auch hier `xsl:stylesheet`. Im Gegensatz zum XSLT-Stylesheet wird für die Formatierungsobjekte noch ein weiterer Namensraum benötigt. Dieser lautet `http://www.w3.org/1999/XSL/Format` und trägt das Präfix `fo`.

(2) An dieser Stelle folgt die wichtigste Templateregel. Sie bezieht sich auf den Dokumentknoten und somit auf das komplette Quelldokument. In ihr befinden sich die wichtigsten Layoutvorlagen.

(3) Das Wurzelelement der Layoutvorlagen heißt `fo:root`. Es ist das erste Formatierungsobjekt im Stylesheet. Innerhalb von `fo:root` werden alle für das PDF-Dokument wichtigen Vorlagen für die Seitengestaltung des PDF-Dokuments festgelegt.

(4) Das erste Kindelement von `fo:root` lautet `fo:layout-master-set`. Es beinhaltet eine Sammlung von Definitionen der Layoutvorlagen für das PDF-Dokument.

(5) Das konkrete Seitenlayout wird mithilfe des Elementes `fo:simple-page-master` bestimmt. Jedes dieser Elemente legt eine Seitenvorlage fest. Das Attribut `master-name` bestimmt einen Namen für die Seitenvorlage, sodass zu einem späteren Zeitpunkt im Stylesheet auf die Vorlage zurückgegriffen werden kann. Der Name der Vorlage lautet `standardseite` und ist als Vorlage für den Fließtext konzipiert. Höhe und Breite der Standardseite werden mit den Attributen `page-height` und `page-width` realisiert. Die Stege werden mit den entsprechenden `margin`-Attributen festgelegt (vgl. Krüger / Welsch 2007, S. 29–30).

(6) Anschließend wird die Standardseite in die drei Bereiche Textfeld, Kopf- und Fußzeile eingeteilt. Listing 10 beschränkt sich auf die Darstellung des Textfelds. Das dafür notwendige Formatierungsobjekt heißt `fo:region-body`. Auch ihm wird mit dem Attribut `region-name` ein Name zugewiesen, damit der Fließtext in diesem Bereich später platziert werden kann. Mittels der `margin`-Attribute werden zusätzlich noch die Abstände zum Seitenrand sowie Kopf- und Fußzeile festgelegt (vgl. Krüger / Welsch 2007, S. 37–38).

(7) Nachdem nun alle Seitenvorlagen bestimmt wurden, wird an dieser Stelle der Verlauf der einzelnen Seiten festgelegt. Dafür stellt XSL-FO das Formatierungsobjekt `fo:page-sequence-master` zur Verfügung. Um die Vorlage später nutzen zu können, wird auch ihr mit `master-name` ein Name zugewiesen. Dieser lautet `kapitel`, da die Vorlage für die Verteilung der einzelnen Kapitel von Kafkas *Verwandlung* auf den Seitenverlauf des PDF-Dokuments verantwortlich ist. Im Stylesheet existieren mehrere Verlaufsvorlagen, da für jeden logischen Bestandteil der Publikation eine eigene Seitenfolge festgelegt werden muss. Das Element `<repeatable-page-master-alternatives>` ermöglicht es, dass mit `fo:simple-page-master` festgelegte Seiten beliebig wiederholt werden können. Erst dadurch wird ein Fließtext über viele Seiten möglich. Mit `fo:conditional-page-master-reference` wird festgelegt, welche Seitenvorlage verwendet wird (vgl. Krüger / Welsch 2007, S. 39–43). In diesem Fall wird somit geregelt, dass alle Kapitel auf die Seitenvorlage `standardseite` verteilt werden, wobei für den Seitenverlauf festgelegt wird, dass sich diese Seitenvorlage beliebig oft wiederholen darf. An dieser Stelle sind alle Vorlagen bestimmt und der erste logische Teil des Stylesheets abgeschlossen.

(8) Im zweiten Teil des Stylesheets wird der konkrete Seitenverlauf des PDF-Dokuments bestimmt. Dafür werden die im ersten Teil festgelegten Vorlagen genutzt. Das Formatierungsobjekt `fo:page-sequence` ist dafür zuständig, konkrete Seitenverläufe umzusetzen. Erst wenn der XSL-FO-Formatter diese Elemente umsetzt, werden die Seiten des PDF-Dokuments erzeugt (vgl. Vonhoegen 2011, S. 341). Mithilfe seines Kindelements `fo:flow` wird der Fließtext schließlich posi-

tioniert. Das Attribut `flow-name` legt fest, dass der Fließtext auf den Seitenbereich `textkorpus` der Seitenvorlage `standardseite` verteilt wird (vgl. Krüger / Welsch 2007, S. 42). Der eigentlich Inhalt wird durch die XSL-Instruktion `xsl:apply-templates` erzeugt. Die Instruktion aktiviert weitere Templates, die sich im dritten logischen Teil des Stylesheets befinden.

(9) Nun beginnt der dritte Teil des Stylesheets. An dieser Stelle des Stylesheets befinden sich die ausgelagerten Templateregeln. Sie sind daher nicht im Element `fo:root` vorhanden. Mithilfe von XSL-Instruktionen werden sie aber aus `fo:root` heraus aktiviert. Im Stylesheet wurden für die einzelnen Bestandteile der Publikation – wie etwa Absätze, Titelei und Inhaltsverzeichnis – eigene Templateregeln angelegt. Stellvertretend für diese wird im Listing nur das Template für die Elemente `tst:kapitel` im Quelldokument aufgeführt. Grundsätzlich ist der Transformationsprozess der vom XSLT-Stylesheet bekannte Prozess. Findet der XSLT-Prozessor im Quelldokument ein Element `tst:kapitel`, so wird das dazu passende Template aktiviert.

(10) Zusätzlich zur bekannten Transformation wird allerdings an dieser Stelle ein XSL-FO-Formatter aktiviert, der die Formatierungsbefehle des `fo:block`-Elements umsetzt. Dadurch entsteht im PDF-Dokument ein eigener Bereich mit dem Inhalt ‚Kapitel'. Der Bereich entspricht den Formatierungsbefehlen des `fo:block`-Elements bezüglich Schriftgröße und -art (vgl. Krüger / Welsch 2007, S. 52–55). Die Kapitelnummer wird mithilfe der XSL-Instruktion `xsl:value-of` hinzugefügt, die auf das Attribut `tst:nr` im Quelldokument zugreift. Da der Kontextknoten während der Transformation von Kapitel zu Kapitel springt, wird automatisch die richtige Kapitelnummer erzeugt. Um unter der Kapitelüberschrift die einzelnen Absätze des jeweiligen Kapitels anzuordnen, wird mit der XSL-Instruktion `xsl:apply-templates` die Aktivierungskette fortgesetzt und die dafür notwendigen Templateregeln werden aktiviert.

(11) Der letzte Teil des Templates soll bewirken, dass am Ende des Kapitels ein Seitenumbruch stattfindet. Da diese Regel nicht für das letzte Kapitel gelten soll, muss an dieser Stelle eine Bedingung formuliert werden. Hierfür stellt XSL die Instruktion `xsl:if` zur Verfügung. Nur wenn der als XPath-Funktion formulierte Text `fn:not(fn:position()=fn:last())` ergibt, dass das aktuelle Kapitel nicht das letzte Kapitel ist, wird ein Seitenumbruch eingefügt (vgl. Bongers 2008, S.125–127).

Listing 10 zeigt, dass ein XSL-FO-Stylesheet einem XSLT-Stylesheet in vielerlei Belangen gleicht, zusätzlich aber über Formatierungsobjekte verfügt, die dem verwendeten XSL-FO-Formatter Anweisungen über die Formatierung der zu transformierenden Elemente geben. XSL-FO-Stylesheets lassen sich außerdem in drei logische Bereiche gliedern. Im ersten Teil des Stylesheets werden die Seitenvorlagen und deren Verlauf im späteren PDF-Dokument festgelegt. Im zweiten Teil wird dieser Verlauf konkretisiert. Hierzu wird der anzuzeigende Inhalt auf die Vorlagen des ersten Teils verteilt. Im dritten Teil des Stylesheets befinden sich alle ausgelagerten Templateregeln. Sie sind für die Art und Weise zuständig, wie die Platzierung und Formatierung im Detail abläuft.

4.7 Zielformat III: EPUB

Das dritte Zielformat ist das vom *International Digital Publishing Forum* entwickelte EPUB-Format. Unter EPUB werden die drei Standards OPS, OPF und OCF zusammengefasst, die am konkreten Fallbeispiel erläutert werden sollen. Eine EPUB-Datei ist ein Zip-Container mit der Endung `.epub`. Die drei Standards bestimmen die Struktur und den Inhalt des Containers. Als Ausgangsdatei wird die im Kapitel 4.5 produzierte XHTML-Datei `kafka_verwandlung.xhtml` verwendet. Als Ergebnis entsteht das E-Book `kafka_verwandlung.epub`. Abbildung 14 zeigt die Ordnerstruktur einer EPUB-Datei.

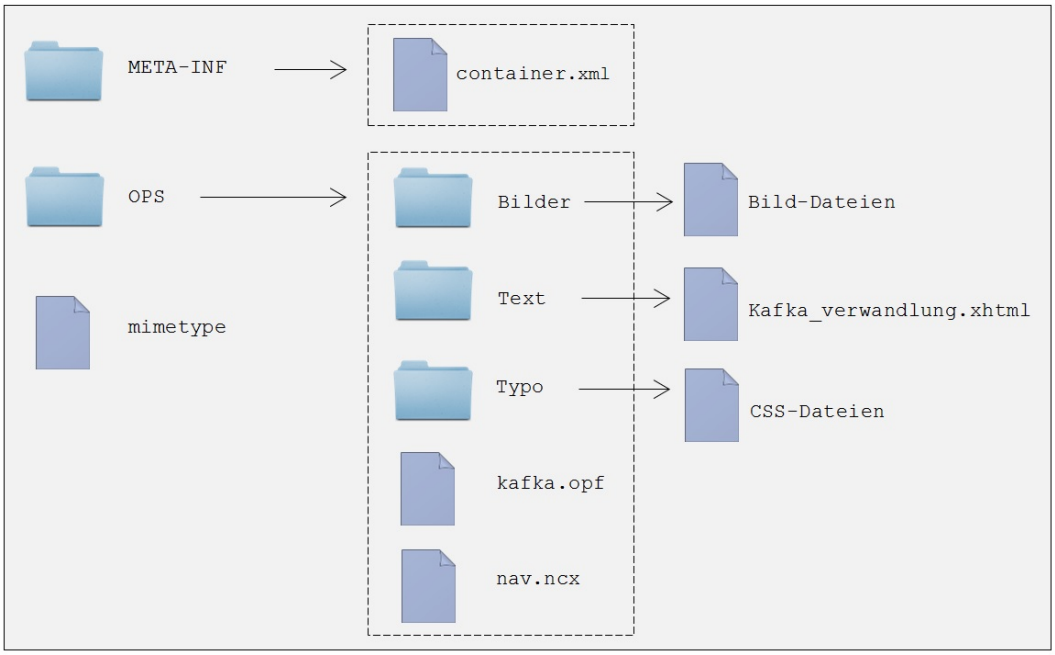

Abbildung 14: Ordnerstruktur EPUB

4.7.1 Open Container Format

Ein EPUB-Container besteht aus mehreren Unterordnern. Die Ordnerstruktur wird durch das *Open Container Format* (OCF) geregelt. Der Standard sieht ein Wurzelverzeichnis in Form der Datei `mimetype` vor, die schlicht den Text `application/epub+zip` enthält. Auf diese Weise wird der Medientyp der Datei an auswertende Programme mitgeteilt. Wichtig ist, dass die Datei `mimetype` beim Verpacken der Unterordner immer an erster Stelle des ZIP-Containers steht, und dass sie nicht komprimiert wird. Des Weiteren sieht der OCF-Standard einen Ordner mit dem Namen META INF vor. Dieser kann grundsätzlich eine Reihe von Dateien enthalten. Die XML-Datei `container.xml` muss dabei aber immer vorhanden sein. Der Zweck von `container.xml` ist es, auf die OPF-Datei zu verweisen, die für die Weiterverarbeitung des EPUB-Formats grundlegend ist.

Schließlich bietet OCF zwei vom W3C entwickelte XML-Standards zur Verwendung von Wasserzeichen und zur Verschlüsselung der Inhalte. Wasserzeichen werden über den *XML-Signature-Standard* geregelt. Sie dienen dazu, die Echtheit

eines E-Books zu belegen. Die Verschlüsselung erfolgt mithilfe des *XML-Encryption-Standards* und kann als Mittel eines Digital Rights Management eingesetzt werden (vgl. Wang 2011, S. 336–338).

4.7.2 Open Publication Standard

Die für EPUB verwendbaren Inhalte werden durch die *Open Publication Structure* (OPS) geregelt. OPS 2.0 bietet zwei Formate an, um Inhalte in Form von Text in EPUB-Dateien einzubinden. Dabei handelt es sich um das Dokumentformat *DTBook*[55] und das bekannte XHTML-Format. Außerdem besteht seit OPS 2.0 die Möglichkeit, Grafiken im SVG-Format – ein auf XML basierender Standard – zu verwenden und neue Schriftarten in CSS-Dokumente einzubetten (vgl. Wang 2011, S. 249). Da unter OPS alle Inhalte zusammengefasst werden, wird in der Ordnerstruktur des EPUB-Containers ein entsprechender Unterordner mit dem Namen ‚OPS' erstellt. Dieser Ordner enthält wiederum drei Unterordner. Im Ordner ‚Bilder' werden alle im E-Book verwendeten Bilder abgespeichert. Der Ordner ‚Text' enthält das XHTML-Dokument `kafka_verwandlung.xhtml`. Im Ordner ‚Typo' befinden sich außerdem die für die typografische Darstellung des E-Books benötigten CSS-Dokumente. Dadurch wird die im Kapitel 2.3 geforderte Trennung von Inhalt und Darstellung eingehalten.

4.7.3 Open Publication Format

Das *Open Publication Format* (OPF) schreibt zwei Dateien vor, die für die Verarbeitung des E-Books elementar sind. Beide Dateien sind in XML verfasst.

Die erste Datei heißt `nav.ncx`. Die Dateiendung steht für *Navigation Control File for XML Application*s. Ihre Aufgabe besteht in der hierarchischen Gliederung der Publikation. Sie ermöglicht es dem Leser, direkt zwischen den einzelnen Ebenen des E-Books zu wechseln. Für die Navigation stellt die NCX-Datei `navPoint`-Elemente bereit. Jeder logischen Hierarchieebene werden mit diesen Elementen Navigationspunkte zugewiesen. Im vorliegenden E-Book stellen die Buchkapitel solche Ebenen dar. Als Ergebnis wird dem Leser ein Inhaltsverzeichnis zur Verfügung gestellt, mit dem er jedes Kapitel im E-Book ansteuern kann (vgl. Wang 2011, S. 237–238, S. 241).

Die zweite durch den OPF-Standard vorgeschriebene Datei heißt `kafka.opf`. Diese Datei hat vor allem zwei Aufgaben. Erstens beinhaltet sie alle relevanten Daten des E-Books. Zweitens beschreibt Sie dessen Bestandteile und Strukturierung. Listing 11 zeigt den Aufbau von `kafka.opf`.

[55] Vgl. hierzu: http://www.daisy.org/structure-guidelines [08.03.2014].

```
<?xml version="1.0" ?>
<package version="2.0" unique-identifier="kakfa_ma"          <!-- 1 -->
 xmlns:opf="http://www.idpf.org/2007/opf"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:dcterms="http://purl.org/terms/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.idpf.org/2007/opf"

 <metadata>                              <!-- 2 -->
  <!--Metadaten des E-Books -->
  <dc:title>Die Verwandlung</dc:title>
  <dc:creator opf:role="aut" opf:file-as="Kafka, Franz">
   Franz Kafka</dc:creator>
  ...
  </metadata>

  <manifest>                               <!-- 3 -->
   <!-- Einband vorne -->
   <item id="titelseite" href="Texte/titelseite.xhtml"
   media-type="application/xhtml+xml"/>
   ...
   <!-- Inhalt des E-Books -->
   <item id="inhalt" href="Texte/kafka_verwandlung.xhtml"
   media-type="application/xhtml+xml"/>
   ...
  <spine toc="ncx">                           <!-- 4 -->
   <itemref idref="titelseite" linear="no"/>
   ...
   <itemref idref="inhalt" linear="yes/>
   ...
  </spine>

  <guide>                                 <!-- 5 -->
   <reference href="Texte/titelseite.xhtml type="cover"
   titel="Cover"/>
   ...
   <reference href="Texte/kafka_verwandlung.xhtml type="cover"
   titel="Cover"/>
   ...
  </guide>
</package>
```

Listing 8: Aufbau von kafka.opf

Listing 11 zeigt einen repräsentativen Ausschnitt aus der Datei **`kafka.opf`**. Die komplette Datei befindet sich im Anhang (vgl. Anhang Nr. 6)

(1) Das Wurzelelement hat den Namen **`package`** und muss die zwei Pflichtattribute **`version`** und **`unique-identifier`** enthalten. Die für das produzierte E-Book

verwendete Version von OPF lautet `2.0`. Mit dem Attribut `unique-identifier` wird eine eindeutige Bezeichnung für die gesamte Publikation festgelegt. Anschließend erfolgt die Benennung der verwendeten Namensräume (vgl. Wang 2011, S. 219–220).

(2) Das erste Kindelement von `package` heißt `metadata`. Hier erfolgt die Festlegung aller Metadaten des E-Books. Jede professionelle Publikation benötigt grundlegende Angaben zu Titel, Autor und Verlag. Der OPF-Standard verwendet hierfür den Metadaten-Standard *Dublin Core* der *Dublin Core Metadata Initiative*.[56] Mit `dc:title` und `dc:creator` werden beispielsweise Verfasser und Titel des Buchs benannt. Das Attribut `opf:role` spezifiziert die Verfasserangabe und legt mit dem Attributswert `aut` fest, dass die genannte Person der Autor des Buchs ist. Die zweite Ergänzung `opf:file-as` stellt eine normalisierte und leichter recherchierbare Autorenangabe dar (vgl. Wang 2011, S. 223–224).

(3) Das zweite Kindelement von `package` hat den Namen `manifest`. Hier werden alle im E-Book verwendeten Dateien in Form von `item`-Elementen zusammengestellt. Im Listing werden zwei Bestandteile des E-Books aufgeführt: der Einband und der eigentliche Text. Für beide Bestandteile liegen in der Ordnerstruktur eigene Dateien vor. Der Pfad zu den Dateien wird mit dem Attribut `href` bestimmt. Das Attribut `id` legt eine eindeutige Bezeichnung für alle Bestandteile fest und mit dem Attribut `media-type` wird außerdem deren MIME-Typ angegeben (vgl. Wang 2011, S. 227–228).

(4) Das Element `spine` – auch Buchrücken genannt – ist ebenfalls ein Pflichtelement im OPF-Dokument. Seine Aufgabe ist die Festlegung der Reihenfolge aller Elemente im E-Book. Die Festlegung geschieht mithilfe des Elements `itemref`, das über das Attribut `idref` auf die Bezeichner der Bestandteile im `manifest`-Element verweist. Ob die aufgeführten Inhalte primärer oder sekundärer Natur sind, wird über das Attribut `linear` festgelegt, das die Werte `yes` und `no` annehmen kann (vgl. Wang 2011, S. 232–233).

(5) Das letzte Element heißt `guide` und dient lediglich der Referenzierung und Benennung aller Inhaltsdokumente, die im E-Book zusammengefasst sind (vgl. Wang 2011, S. 234–236).

Neben der eigentlichen Inhaltsdatei `kafka_verwandlung.xhtml` spielt `kafka.opf` die wichtigste Rolle für die Entstehung des E-Books. Hier werden nicht nur alle Metadaten der Publikation festgelegt, sondern auch deren Bestandteile genannt und in Verbindung gebracht. Das `manifest`-Element listet alle im E-Book verwendeten Dateien auf. Das betrifft sowohl Inhaltsdateien als auch CSS-Dokumente und Bilddateien. Das `code`-Element legt die Reihenfolge aller Bestandteile während der eigentlichen Darstellung des E-Books fest. Das optionale `guide`-Element referenziert alle Bestandteile abschließend und bestimmt deren Semantik (vgl. Wang 2011, S. 237).

[56] Vgl. hierzu: http://dublincore.org/metadata-basics/ [10.03.2014].

5 Evaluation

In Kapitel 4 wurden die für ein XML-basiertes Content-Management notwendigen Schritte erläutert. Im Verlauf des Kapitels entstanden aus einer Ausgangsdatei im XML-Format und einer Reihe von Bilddateien drei Zieldateien im XHTML-, PDF- und EPUB-Format. Kapitel 5 soll Aufschluss über die Potentiale eines XML-basierten CMs für Verlage geben.

5.1 Freie oder proprietäre Software

Die Potentiale, die sich aus einem XML-basierten CM ableiten, hängen sehr stark von der verwendeten Software ab. Da XML vom W3C als freier Standard konzipiert ist, gibt es eine Vielzahl unterschiedlicher auf XML basierender Software. Grundsätzlich stellt sich für Verlage dabei die Frage, inwiefern freie und kostenlose oder proprietäre Softwarelösungen genutzt werden.

Ein großer Vorteil, den XML bietet, ist die umfassende Unterstützung des Formats in Web- und Datenbanksystemen. Mit XML-angereicherte Texte zeichnen sich daher durch ihre Plattformunabhängigkeit aus. Sie können über entsprechende Programmierschnittstellen in eine Vielzahl von Anwendungen integriert werden. Außerdem existieren für einen Großteil der Aufgaben, die in einem CM anfallen, freie, XML-basierte Softwarelösungen. Das betrifft sowohl integrierende CM-Systeme als auch einzelne CM-Anwendungen. So wurden für Kapitel 4 größtenteils Open-Source-Programme verwendet.

In einem ersten Schritt wurde ein XML-Dokument erstellt und abgespeichert. Im zweiten Schritt wurde aus diesem Dokument ein Inhaltsmodell in Form einer XML-Schema-Definition abgeleitet. Beide Schritte lassen sich mit gängigen XML-Editoren umsetzten. Die im dritten Schritt umgesetzten Transformationen geschahen mithilfe der Open-Source-Prozessoren *Saxon-HE* für XSLT und des *Formatting Objects Processor* (FOP) für XSL-FO. Dabei stieß FOP bei der Transformation des XML-Dokuments in PDF bereits an seine Grenzen, da der Formatierer einer Reihe von Einschränkungen unterliegt. So unterstützt FOP weder Fließelemente (`fo:float`) noch Sonderregeln für Initialen (`fo:initial-property-set`). Welche XSL-FO-Elemente unterstützt werden und welche nicht, lässt sich der Website der Entwickler entnehmen.[57] In der Praxis empfiehlt es sich daher, kostenpflichtige und proprietäre Prozessoren zu nutzen, da diese nahezu die gesamten XSL-FO-Standards unterstützen (vgl. Krüger / Welsch 2007, S. 21–23). Beispiele sind der *XSL Formatter* von *Antenna House* [58] oder der *XEP*-Formatierer des US-amerikanischen Herstellers *RenderX*.[59] Die Wahl des XSLT-Prozessors ist dagegen nebensächlich, da XSLT-Prozessoren in aller Regel standardkonform arbeiten und lizenzfrei bezogen werden können (vgl. Krüger / Welsch 2007, S. 24).

[57] Vgl. hierzu: http://xmlgraphics.apache.org/fop/compliance.html [19.03.2014].

[58] Vgl. hierzu: http://www.antennahouse.com/ [19.03.2014].

[59] Vgl. hierzu: http://www.renderx.com/ [19.03.2014].

Bei der Erstellung des EPUB-Formats kann ebenfalls eine Reihe von Schwierigkeiten auftreten. Diese beziehen sich in erster Linie auf die verwendete Anzeigesoftware sowie die genutzten Endgeräte. So werden neue Schriftarten, die über CSS-Formatierungsanweisungen in den EPUB-Container integriert wurden, bei *Adobe Digital Editions 3.0* angezeigt, wohingegen *calibre 1.2.8* dazu nicht in der Lage ist. Da CSS-Anweisungen normalerweise Pixel als Größenangaben verwenden, müssen außerdem die Bildschirmgrößen und -auflösungen der Endgeräte bei der Produktion von E-Books mit EPUB berücksichtigt werden.

Der in Kapitel 4 präsentierte Workflow muss schließlich als Experiment betrachtet werden. Für große Verlage ist kann es sinnvoller sein, nicht auf einzelne CM-Anwendungen zurückzugreifen, sondern ein integrierendes, proprietäres CM-System zu verwenden. Systeme dieser Art bestehen meist aus einer Client- und einer Serverkomponente und gewährleisten dadurch eine einheitliche und zentralisierte Verwaltung von Contentobjekten. Mögliche Zusatzfunktionen – wie beispielsweise Versionsmanagementsysteme und Workflow-Manager –, die für die Verwaltung einer Vielzahl von heterogenen Contentobjekten und entsprechend komplexen Workflows erforderlich sind, erweitern den Nutzen solcher Systeme. Unabhängig von der Verwendung solcher Systeme oder differenter Anwendungen, hängen die Potentiale, die sich durch XML-basiertes CM für Verlage ergeben, von einer Vielzahl von Faktoren ab.

5.2 Externe Faktoren: Der Buchmarkt

Die Rolle, die CM für Verlage spielt, steht in kausalem Zusammenhang mit den Entwicklungen des deutschen Buchhandels im Bereich der Digitalisierung und der Akzeptanz von E-Books und elektronischen Lesegeräten bei den Endabnehmern. Laut einer aktuellen Studie, die in Zusammenarbeit des *Börsenvereins des deutschen Buchhandels* mit der *Gesellschaft für Konsumforschung* (GfK) entstand, steigt die Akzeptanz von E-Books in Deutschland kontinuierlich (Börsenverein 2013, S. 3). Vor allem auf Verlagsseite wird diese Entwicklung forciert. Im Jahr 2013 beteiligten sich bereits 53 % der 437 befragten Verlage am E-Book-Markt und 84 % planten die Produktion und Distribution von E-Books für die Zukunft. Für einen Großteil der Verlage gehören E-Books somit zum festen Verlagsprogramm. Mehr als die Hälfte aller Novitäten erscheint außerdem bereits als E-Book (vgl. Börsenverein 2013, S. 10–11). Eine Studie von *Pricewaterhouse Coopers* (pwc) aus dem Jahr 2013 bestätigt diese Tendenzen. Zwar liegt der Umsatzanteil von E-Books bei noch 2,4 % des gesamten Branchenumsatzes (vgl. pwc 2013, S. 11), der E-Book-Markt gerät aber insgesamt zusehend in Schwung. Die Voraussetzungen für das Umsatzwachstum wurden einerseits von den Verlagen selbst geschaffen. So hat sich das Angebot deutschsprachiger E-Books in den letzten Jahren vervielfacht. Insbesondere aber differenzierte sich das Angebot von E-Readern in Bezug auf Funktionalität und Preis weiter aus (vgl. pwc 2013, S. 5). Die Akteure des deutschen Buchmarkts reagierten auf diese Tendenzen mit neuen Absatzkanälen und Vertriebsplattformen: »[...] ein Großteil der deutschen Buch-

verlage hat inzwischen ebenfalls die nötigen Investitionen unternommen, um mit digitalen Büchern alle relevanten Vertriebskanäle zu bestücken« (pwc 2013, S. 5).

5.3 Interne Faktoren: Der Verlag

Ob sich Potentiale aus einem XML-basierten CM für einen Verlag ergeben, hängt davon ab, inwiefern E-Publishing- und E-Commerce-Prozesse in der Unternehmensstrategie der jeweiligen Verlage eine Rolle spielen. Verlage, deren Fokus auf der Herstellung von Printbüchern liegt, werden einen geringeren Nutzen aus der Implementierung eines Content-Management-Systems ziehen, als Verlage, die sich auf digitale Absatzkanäle spezialisieren. Dessen ungeachtet können auch reine Printverlage von einem umfassenden Content-Management profitieren, da das PDF, das ebenfalls in Kapitel 4 hergestellt wurde, als Grundlage für hochwertige Printbücher genutzt werden kann.

Abgesehen vom reinen Publikationsprozess, können Verlage aber auch anderweitig Vorteile aus CMS ableiten. Mithilfe von CMS können Arbeitsabläufe über Workflow-Elemente gesteuert werden. So erlauben CMS die projektgesteuerte Zuordnung redaktioneller Aufgaben an Mitarbeiter und Abteilungen. Allen Projektbeteiligten können außerdem spezifische Zugangsrechte zum Content Repository eingeräumt werden. Versionsfähige CMS unterstützen die parallele Verwaltung von verschiedenen Versionen aller im Content Repository archivierten Contentobjekte. Informationsstrukturen können dadurch um ein Vielfaches effektiver gestaltet werden (vgl. Rothfuss / Ried 2003, S. 130–131). Semistrukturierte Textdokumente können formatunabhängig zwischen verschiedenen Anwendungen getauscht werden. Mittels spezifischer Programmierschnittstellen können CMS außerdem an andere Systeme angebunden werden. Der Nutzen von CMS für Verlage ergibt sich somit aus einer zentralisierten Informationsverwaltung, die einen schnellen Zugriff auf relevante Dokumente und Dateien für verschiedene Benutzergruppen innerhalb und außerhalb des Verlags ermöglicht (vgl. Christ 2003, S. 172). Noch größere Potentiale birgt XML-basiertes CM für Verlage, die gezielt digitale Absatzkanäle nutzen. Die Transformation der XML-Dokumente in XHTML und HTML5 ermöglicht die Publikation der Contentobjekte über Websites, plattformunabhängige Web-Apps, mobile Applikationen oder über Cloud-Systeme. Mit EPUB steht außerdem ein sehr verbreiteter und offener Standard zur Produktion von E-Books zur Verfügung. Schließlich können mithilfe des PDFs hochwertige Druckergebnisse erzielt werden. Die Vorteile, die ein XML-basiertes CM mit sich bringt, liegen auf der Hand. Sie betreffen sowohl den Publikationsprozess selbst als auch die verlagsinterne Informations- und Datenverwaltung.

5.4 Risiken

Zugleich entsteht auch eine Reihe von Risiken. Diese betreffen einerseits die Implementierung eines CMS in einem Verlag. Alle Geschäfts- und Publikationsprozesse müssen auf das System angepasst oder in das System integriert werden. Da-

bei können erhebliche Kompatibilitätsprobleme auftreten. Mit der Integration alter Datenbestände in ein neues CMS ist außerdem ein hoher Zeit- und Kostenaufwand verbunden. Zusätzlich müssen alle Beteiligten für das neue System geschult werden. Das betrifft unter anderem Redakteure, Autoren und Mediengestalter, die auf XML-Editoren oder vergleichbare CMA zurückgreifen müssen. Je nach Komplexität des Systems müssen außerdem weitere Fachkräfte, die XML und verwandte Technologien beherrschen, eingestellt werden. Sowohl die erfolgreiche Realisierung als auch eine nachhaltige Integration eines CMS in einen Verlag hängt maßgeblich von allen beteiligten Angestellten ab (vgl. Koop et al 2001, S. 129–131). Zu den genannten internen Risikofaktoren können weitere externe Risiken durch die Endabnehmer entstehen. Es besteht vor allem die Gefahr, dass die Kunden eines Verlags – je nach Zielgruppe – digital publizierte Verlagsprodukte nicht annehmen. Hinzu kommt das Risiko, dass digitale Nutzungsrechte umgangen oder missachtet werden. Die Implementierung eines Content-Management-Systems muss daher gut geplant und zielgerichtet umgesetzt werden.

Um den internen und externen Risiken, die durch die Einführung neuer Systeme entstehen können, zu begegnen, sollte die Implementierung eines CMS durch ein Change- und Innovationsmanagement vorbereitet und begleitet werden. Changemanagement fasst dabei alle Tätigkeiten zusammen, die für die Einführung komplexer und neuer Systeme notwendig sind. Dazu gehört die Formulierung der angestrebten Ziele, die Erfassung des Status quo und die Planung der benötigten Strategien, um tiefgreifende Veränderungen in der Unternehmenskultur und den Geschäftsprozessen zu verankern (vgl. Koop et al. 2001, S. 237–239). Innovationsmanagement meint dagegen die zielgerichtete Organisation aller Aktivitäten, die zur Entwickelung von Innovationen erforderlich sind. Innovationsmanagement hat somit einen weiteren Anspruch als Changemanagement: Es dient nicht nur der Integration neuer Prozesse und Systeme, sondern umfasst insbesondere die Erforschung, Entwicklung und Gestaltung neuer Prozesse und Produkte (vgl. Hagenhoff 2008, S. 14). Eine zentrale Rolle im Innovationsmanagement spielt die Formulierung einer Strategie, die dazu verhelfen soll, Technologiepositionen so einzusetzen, dass diese den wirtschaftlichen Erfolg einer Organisation nachhaltig und signifikant verbessern. Die einzelnen Schritte eines Innovationsmanagements umfassen dabei die Analyse der Umwelt, der Konkurrenz und des eigenen Unternehmens. Die Umweltanalyse soll Aufschluss über neue technologischen Trends und deren Relevanz für die eigene Wertschöpfungskette geben. Die Konkurrenzanalyse untersucht die Tätigkeiten potentieller Wettbewerber. Wettbewerber mit ähnlichen Wettbewerbsstrategien werden zu diesem Zweck in strategischen Gruppen zusammengefasst (vgl. Hungenberg 2012, S. 129–130). Das Ziel einer innovationsorientierten Unternehmensanalyse ist schließlich die Bewertung der eigenen Innovationsleistung in der Vergangenheit sowie die Bewertung zukunftsbezogener Fähigkeiten (vgl. Hagenhoff 2008, S. 23–25). Für Verlage, die erfolgreich ein CMS implementieren wollen, spielen diese Analyseverfahren eine signifikante Rolle, da sowohl Produktions- und Publikationsprozesse als auch die hergestellten Verlagsprodukte dem *State of the Art* der Buchbrache entsprechen müssen, damit ein Verlag langfristig konkurrenzfähig bleiben kann. Zugleich muss sichergestellt werden, dass die Einführung eines CMS und die daraus resultieren-

den Produkte mit dem Innovationspotential des Verlags und den Kundenerwartungen vereinbar sind.

Die Potentiale eines XML-basierten Content-Managements müssen multikausal bewertet werden. Sie hängen stark von der allgemeinen Branchensituation, dem technischen Kontext und Fortschritt sowie der Strategie und Ausrichtung des Verlags ab. Dabei spielen die anvisierten Marktsegmente und Kundenerwartungen eine ebenso wichtige Rolle, wie die Verlagsprodukte selbst. Verlage, die E-Books und Web-Content anbieten, können die Potentiale von XML-basiertem Content-Management besser abschöpfen, als Verlage, die sich ausschließlich auf den Printbereich konzentrieren. Auch die Endabnehmer spielen eine wichtige Rolle. Verlage, die über einen internet-affinen Kundenkreis verfügen, können entsprechende Erlösmodelle besser einsetzen, als Verlage, deren Kundekreis auch zukünftig größtenteils gedruckte Bücher nutzen will.

6 Fazit

Die vorliegende Masterarbeit umfasst zwei Teile. Im ersten Teil wurden die theoretischen Grundlagen gelegt und zentrale Begriffe geklärt. Kapitel 2 behandelt das Thema Content Management (CM). CM wurde daher als theoretisches Fundament für den praktischen Teil gewählt, weil es anschaulich die mögliche Produktion, Verwaltung und Publikation von Contentobjekten erläutert und vielfach im Kontext von XML Erwähnung findet. Die im praktischen Teil der Masterarbeit produzierten Dateien werden als Contentobjekte betrachtet.

Um den komplexen Begriff Content zu klären, wurde Content in das Konstrukt von Daten, Informationen und Wissen eingeordnet und anschließend für die vorliegende Masterarbeit definiert. Content wurde dabei als ein aus Daten bestehendes, mehrdimensionales Gebilde von Einzelinformationen beschrieben, das in einer Publikationsabsicht produziert wurde. Als Assets wurden alle medialen Bestandteile bezeichnet, die in einem Contentobjekt zusammengefasst sind. Zentral für Contentobjekte ist die separate Speicherung von Inhalt, Layout und Struktur, wobei die Verbindung von Speicherort und Container referenzieller Natur ist. Dieser Umstand erlaubt die vielfache Verwendung einzelner Assets, Struktur- und Layoutvorgaben in unterschiedlichen Publikationsabsichten. Der zweite Begriff, den es für die Masterarbeit zu klären galt, ist Management. Management wurde nach HUNGENBERG als leistungsoptimierendes ›Einflusshandeln‹ (Hungenberg 2012, S. 20) auf materielle und immaterielle Güter in Unternehmen bezeichnet. In Anlehnung an HAGENHOFF wurde ein funktionaler Managementbegriff für die Masterarbeit gewählt. Funktionales Management beschreibt Aufgaben und Handlungen, die zur Steuerung eines Leistungsprozesses innerhalb einer Organisation nötig sind (vgl. Hagenhoff 2008, S. 20–21). CM bezeichnet zusammenfassend alle Aufgaben und Handlungen, die eingesetzt werden, um Content in einen unternehmerischen Leistungsprozess zu integrieren.

Die Anwendung eins möglichen CMs erfolgt prozesshaft und wurde in Kapitel 2.4 dargelegt. Ein idealtypischer CM-Prozess durchläuft sechs Schritte: die Definition eines Konzeptes, die Erstellung oder Beschaffung von Content, dessen Kombination zu neuen Contentobjekten, die Formatierung der Objekte in das gewünschte Zielformat und deren Publikation sowie die Archivierung oder Löschung der Contentobjekte. Jeder dieser Schritte kann mithilfe separater Anwendungen oder integrierender CMS umgesetzt werden. Zur Veranschaulichung wurde in Kapitel 2.5 ein CMS aus der wissenschaftlichen Praxis vorgestellt, das den Namen TextGrid trägt und die XML-gestützte Produktion und Verwaltung wissenschaftlicher Datenbestände ermöglicht.

Kapitel 3 dient ebenfalls der theoretischen Fundierung. In diesem Kapitel wurden die für den praktischen Teil der Masterarbeit benötigten Grundlagen der *eXtensible Markup Language* (XML) erläutert. Hierfür wurden die vom W3C definierten Bausteine von XML vorgestellt. Mit XML werden Datenobjekte beschrieben, die als XML-Dokumente bezeichnet werden. XML-Dokumente haben eine physikalische und eine logische Struktur. Die physikalische Struktur ergibt sich aus den separaten Textstücken, in die ein XML-Dokument aufteilt werden kann. Diese Textstücke werden auch als Entitäten bezeichnet. Die logische Struktur ei-

nes XML-Dokuments besteht wiederum aus Informationsträgern, die im *XML Information Set* (Infoset) definiert werden. Die entsprechende W3C-Empfehlung umfasst insgesamt elf unterschiedliche Informationsträger. Die wichtigsten Informationsträger sind Elemente. Jedes XML-Dokument besteht aus einer Reihe von Elementen, die zueinander in Beziehung stehen. Anschließend wurden weitere Schlüsseltechnologien vorgestellt, die ebenfalls im praktischen Teil der Masterarbeit verwendet wurden. Dazu gehören *Dokumententyp-Definitionen* (DTD) und *XML-Schema-Definitionen* (XSD) zur Festlegung der Struktur von XML-Dokumenten, die *eXtensible Stylesheet Language* (XSL), die für die Umwandlung von XML-Dokumenten entwickelt wurde und sich in die zwei Teilbereiche *XSL Transformation* (XSLT) und *XSL Formatting Objects* (XSL-FO) aufteilt sowie der E-Book-Standard *electronic Publication* (EPUB) und die Gestaltungssprache *Cascading Style Sheets* (CSS), die für die Darstellung des XHTML-Dokuments und der EPUB-Datei genutzt wurde.

Im zweiten Teil der Masterarbeit erfolgte die praktische Umsetzung eines XML-basierten Content-Managements, das anhand einer Reihe von Stylesheets erläutert wurde. Der Workflow des entwickelten CMs wurde in Kapitel 4.1 vorgestellt. Dieser sah vier Schritte vor: die Erstellung eines XML-Dokuments, die Definition eines Inhaltsmodells, die Kombination und Formatierung der Ausgangsdateien sowie die Publikation der Ergebnisdateien. Grundlage ist das in Kapitel 4.3 vorgestellte XML-Dokument. Hierfür wurde Kafkas *Die Verwandlung* als repräsentatives Beispiel mit XML-Tags angereichert. Das Ergebnisdokument trägt den Namen `kafka_verwandlung.xml`. Anschließend erfolgte die Ableitung einer Struktur mithilfe einer XSD. Die XSD `kafka_struktur.xsd` kann als Schablone für alle weiteren XML-Dokumente genutzt werden, um diese in das entwickelte CMS einzubinden. In Kapitel 4.5 und Kapitel 4.6 wurden die Ausgangsdateien in die zwei Formate XHTML und PDF transformiert. Die entsprechenden Stylesheets tragen die Namen `kafla_xslt.xsl` und `kafka_xslfo.xsl`. Die in Kapitel 4.5 entstandene XHTML-Datei wurde schließlich in einen EPUB-Container eingebunden. Als Ergebnis des praktischen Teils entstanden die drei Ergebnisdateien `kafka_verwandlung.xhtml`, `kafka_verwandlung.pdf` und `kafka_verwandlung.epub`. Für die Publikation wurde eine Webspace eingerichtet, der über die URL http://www.samsas-verwandlung.de/ erreicht werden kann.

Abschließend wurden die Potentiale eines XML-basierten Content-Managements in Kapitel 5 diskutiert. Die Potentiale ergeben sich aus einer Vielzahl von Faktoren, die mitunter vom jeweiligen Verlag, von der Branchensituation und von den Endabnehmern des Verlags abhängen. Die Potentiale müssen daher multikausal bewertet werden. Insbesondere für Verlage, die digitale Absatzkanäle nutzen, kann die Nutzung eines CMS zu einem hohen Mehrwert führen. Da die Implementierung eines CMS zugleich mit einigen Risiken einhergeht, sollte sie immer durch ein Change- und Innovationsmanagement vorbereitet und begleitet werden.

Literaturverzeichnis

AIIM 2013: What is Enterprise Content Management (ECM)?, auf: http://www.aiim.org/What-is-ECM-Enterprise-Content-Management [12.03.2014].

Bernstein, Jay H. 2009: The Data-Information-Knowledge-Wisdom Hierarchy and its Antithesis. Proceedings North American Symposium on Knowledge Organization, Vol. 2:68-75, http://arizona.openrepository.com/arizona/handle/10150/105414 [12.03.2014], S. 68–75.

Bodendorf, Freimut 2006: Daten- und Wissensmanagement, 2., aktualisierte und erweiterte Auflage, Nürnberg 2006.

Bodrow, Wladimir / Bergmann, Philipp 2003: Wissensbewertung in Unternehmen. Bilanzieren von intellektuellem Kapital, Berlin 2003.

Boiko, Bob 2005: Content Management Bible, 2nd Edition, Indianapolis 2005.

Bongers, Frank 2008: XSLT 2.0 & XPath 2.0, 2., aktualisierte und erweiterte Auflage, Bonn 2008.

Börsenverein 2013: E-Book-Studie. Kurzzusammenfassung, o.O 2013, https://www.etracker.de/lnkcnt.php?et=QS9axK&url=http://www.boersenverein.de/sixcms/media.php/976/E-Book-Studie%202012%20PRESSEMAPPE_print.pdf [19.03.2014].

Börsenverein 2006: Warengruppen-Systematik neu (WGSneu) – Version 2.0. Einheitlicher Branchenstandard ab 1. Januar 2007, o.O 2006, https://www.etracker.de/lnkcnt.php?et=QS9axK&url=http://www.etracker.de/lnkcnt.php?et=QS9axK&url=http%3A%2F%2Fwww.boersenverein.de%2Fsixcms%2Fmedia.php%2F976%2Fwgs2012.pdf&lnkname=wgs2012.pdf [19.03.2014].

Brügmann, Hinnerk 2011: Management of unstructured Information using semantic Metadata, Dissertation, Nürnberg 2011.

Bullinger, Hans-Jörg et al. 2001: Content Management Systeme. Auswahlstrategien, Architekturen und Produkte (Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO), 5., erweiterte und überarbeitete Auflage, Stuttgart 2001.

Castro, Elizabeth / Hyslop, Bruce 2012: HTML5 und CSS3. Der Meisterkurs, München 2012.

Christ, Oliver 2003: Content-Management in der Praxis. Erfolgreicher Aufbau und Betrieb unternehmensweiter Portale, Berlin 2003.

Cmscritic 2014: Software Lists, auf: http://www.cmscritic.com/resource-lists/ [12.03.2014].

Dengel, Andreas 2012: Semantische Technologien. Grundlagen – Konzepte – Anwendungen, Heidelberg 2012.

Eckstein, Rainer / Eckstein, Silke 2004: XML und Datenmodellierung. XML-Schema und RDF zur Modellierung von Daten und Metadaten einsetzen, Heidelberg 2004.

Ehlers, Lars H. 2003: Content Management Anwendungen. Spezifikation von Internet- Anwendungen auf Basis von Content Management Systemen (Advances in Information Systems and Management Science, 1), Berlin 2003.

Erlenkötter, Helmut 2012: XML. Extensible Markup Language von Anfang an (Grundkurs Computerpraxis), 3. Auflage, Hamburg 2012.

Fey, Dietmar 2010: Grid-Computing. Eine Basistechnologie für Computational Science, Heidelberg 2010, http://link.springer.com/book/10.1007%2F978-3-540-79747-0 [15.03.2014], DOI 10.1007/978-3-540-79747-0.

Frické, Martin 2008: The Knowledge Pyramid: A Critique of the DIKW Hierarchy, preprint (2008), http://arizona.openrepository.com/arizona/handle/10150/105670 [12.03.2014], S. 1–13.

Fröschle, Hans-Peter / Behrendt, Werner 2007: Technologien und Trends für Wissensarbeit und Wissensmanagement, in: Fröschle, Hans-Peter / Reich, Siegfried (Hrsg.): Enterprise Content Management (HMD 258), 2007, S. 6–15.

Fuchs, Daniel 2007: Web Content Management Systeme. Evaluation anhand eines Praxisbeispiels, Saarbrücken 2007.

Gietz, Peter et al. 2006: TextGrid and eHumanities, in: Conference on e-Science and Grid Computing (e-Science 2006), Second IEEE International, 2006, S. 133–140.

Götzer, Klaus et al. 2008: Dokumenten-Management. Informationen im Unternehmen effizient nutzen, 4., vollständig überarbeitete und erweiterte Auflage, Heidelberg 2008.

Güldenberg, Stefan 1998: Wissensmanagement und Wissenscontrolling in lernenden Organisationen. Ein systemtheoretischer Ansatz (Edition Österreichisches Controller-Institut), 2., durchgesehene Auflage, Wiesbaden 1998.

Hagenhoff, Svenja 2008: Innovationsmanagement für Kooperationen. Eine instrumentenorientierte Betrachtung, Göttingen 2008, www.oapen.org/ download?type=document&docid=359553 [20.03.2014].

Hagenhoff, Svenja et al. 2007: Neue Formen der Wissenschaftskommunikation. Eine Fallstudienuntersuchung (Göttinger Schriften zur Internetforschung, Band 4), Göttingen 2007, www.oapen.org/download?type=document&docid=359552 *[20.03.2014]*.

Hansen, H. R. / Neumann, G. 2001: Wirtschaftsinformatik I. Grundlagen betrieblicher Informationsverarbeitung, 8., völlig neubearbeitete und erweiterte Auflage (Grundwissen der Ökonomik), Stuttgart 2001.

Hasler Roumois, Ursula 2007: Studienbuch Wissensmanagement. Grundlagen der Wissensarbeit in Wirtschafts-, Non-Profit- und Public-Organisationen, Zürich 2007.

Hontheim, Erwin / Herzog-Kienast, Andrea 2012: Das TYPO3-Buch. Schritt für Schritt zur professionellen Web-Präsenz, 2. Auflage, München 2012.

Hotho, Andreas et al. 2005: A brief Survey of Text Mining, in: LDV Forum – *GLDV Journal for Computational Linguistics and Language Technology (Band 20), Heft 1, 2005, S. 19–62.*

Hungenberg, Harald 2012: Strategisches Management in Unternehmen. Ziele – Prozesse – Verfahren, 7., aktualisierte Auflage, Wiesbaden 2012, http://link.springer.com/book/10.1007%2F978-3-8349-3841-1 [21.03.2014], DOI 10.1007/978-3-8349-3841-1.

Ide, Nancy / Sperberg-McQuenn, C.M. 1995: The Text Encoding Initiative: Its History, Goals, and Future Development, in: Ide, Nancy / Véronis, Jean (Hrsg.): Text Encoding Inititiative. Background and Context, Dordrecht 1995, S. 5–15.

IDPF 2010: OPS, Kapitel 1.3.6, auf: http://www.idpf.org/epub/20/spec/OPS_2.0.1_draft.htm#Section1.3.6 [12.03.2014].

Klingelhöller, Harald 2001: Dokumentenmanagementsysteme. Handbuch zur Einführung (Xpert.press), Berlin / Heidelberg 2001.

Kretschmar, Oliver / Dreyer, Roland 2004: Medien-Datenbank und Medien-Logistik-Systeme. Anforderungen und praktischer Einsatz, München 2004.

Krüger, Manfred / Welsch, Ursula 2007: XSL-FO Praxis (schnell + kompakt), Frankfurt 2007.

Küster, Marc Wilhelm et al. 2009: TextGrid: eScholarship und vernetzte Angebote, in: IT – Information Technology (Band 51), Heft 4, 2009, S. 183–190.

Koop, Hans Jochen et al. 2001: Erfolgsfaktor Content Management. Vom Web Content bis zum Knowledge Management, Braunschweig / Wiesbaden 2001.

Lehner, Franz 2009: Wissensmanagement. Grundlagen, Methoden und technische Unterstützung, 3. Aktualisierte und erweiterte Auflage, München / Wien 2009.

Lobin, Henning 2010: Computerlinguistik und Texttechnologie, Paderborn 2010.

Marriott, Jennifer 2013: The official JOOMLA! book, Upper Saddle River, NJ u.a., 2013, http://proquest.tech.safaribooksonline.de/book/web-development/joomla/9780132978972 [12.03.2014].

Maass, Wolfgang / Kowatsch, Tobias 2012: Semantic Technologies in Content Management Systems. Trends, Applications and Evaluations, Heidelberg 2012., http://link.springer.com/chapter/10.1007%2F978-3-642-24960-0_9 [13.03.2014], DOI 10.1007/978-3-642-24960-0.

OASIS Consortium 2014: DocBook Publishers SC, auf: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=docbook-publishers [12.03.2014].

Probst, Gilbert et al. 2012: Wissen managen. Wie Unternehmen ihre wertvollste Ressource optimal nutzen, 7. Auflage, Wiesbaden 2012, http://link.springer.com/book/10.1007/978-3-8349-4563-1/page/1 [13.03.2014], DOI 10.1007/978-3-8349-4563-1.

pwc 2013: Media Trend Outlook. E-Books im Aufwind, o.O. 2013, http://www.pwc.de/de/technologie-medien-und-telekommunikation/assets/whitepaper-ebooks.pdf [19.03.2014].

Rautenberg, Ursula 2003: Buch, in: Rautenberg, Ursula (Hrsg.): Reclams Sachlexikon des Buches, 2. verbesserte Auflage, Stuttgart 2003, S. 82–86.

Riggert, Wolfgang 2009: ECM – Enterprise Content Management. Konzepte und Techniken rund um Dokumente, Wiesbaden 2009.

Rothfuss, Gunther / Ried, Christian 2003: Content Management mit XML. Grundlagen und Anwendungen (Xpert.press), 2., überarbeitete Auflage, Berlin / Heidelberg 2003.

Sauer, Moritz 2013: Das WordPress-Buch, Beijing u.a. 2013.

Spörrer, Stefan 2009: Content Management Systeme. Begriffsstruktur und Praxisbeispiel, Köln 2009.

Tal, Liran 2014: Dupal 7 media, Birmingham 2013, http://proquest.tech.safaribooksonline.de/book/web-development/drupal/9781849516082 [12.03.2014].

TEI Consortium 2014: TEI History, auf: http://www.tei-c.org/About/history.xml [12.03.2014].

TextGrid 2014a: Das Projekt, auf: http://www.textgrid.de/ueber-textgrid/projekt/ [12.03.2014].

TextGrid 2014b: Die Digitale Bibliothek bei TextGrid (2014), auf: http://www.textgrid.de/Digitale-Bibliothek [15.03.2014].

TextGrid 2012: User Manual 2.0, o.O 2012, https://dev2.dariah.eu/wiki/display/TextGrid/User+Manual+2.0 [15.03.2014].

Thygs, Michael 2001: Gestaltungsempfehlungen für Content Management Systeme (CMS) zur Unterstützung des Projektmanagements, Münster 2001.

Vonhoegen, Helmut 2011: Einstieg in XML. Grundlagen, Praxis, Referenz, 6., aktualisierte und erweiterte Auflage, Bonn 2011.

W3C 2014: XHTML 2.0, Abstract , auf: http://www.w3.org/TR/xhtml2/ [12.03.2014].

W3C 2012: XSD 1.1, Part 1, Kapitel 1.3.1.2 , auf: http://www.w3.org/TR/xmlschema11-1/ [12.03.2014].

W3techs 2013: Usage of content management systems for websites, auf: http://w3techs.com/technologies/overview/content_management/all [12.03.2014].

Wang , Victor 2011: E-Books mit ePub. Von Word zum E-Book mit XML, Heidelberg u.a. 2011.

Weitzel, Thomas 2014: Contao für Webdesigner. Mit responsiver Beispielwebsite, Tutorials, Checklisten, München 2014.

Whitmore: Andrew 2013: A Student's Guide to XSLT, o. O. 2013.

Wilhelm, Stephan 2006: Verfahren zur Einführung eines internetbasierten Content Management für Qualitätsregelkreise in der Produktion (Fraunhofer-Institut für Arbeitswissenschaft und Technologiemanagement IAT), Dissertation, Heimsheim 2006.

Anhang

1. Bilddateien

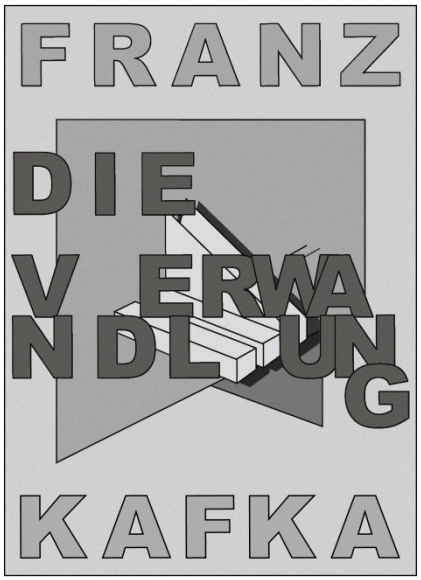

Illustration 1: Titelbild (F. Wolf)

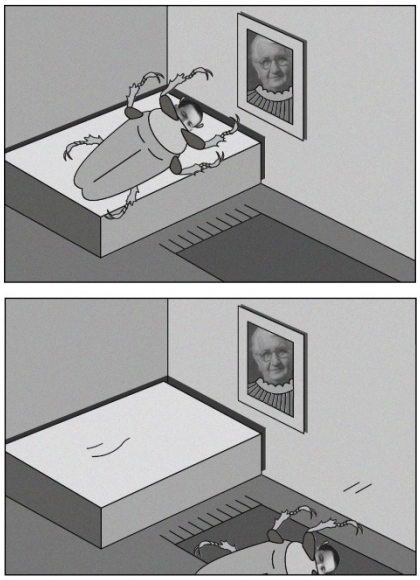

Illustration 2: Kapitel 3 (F. Wolf)

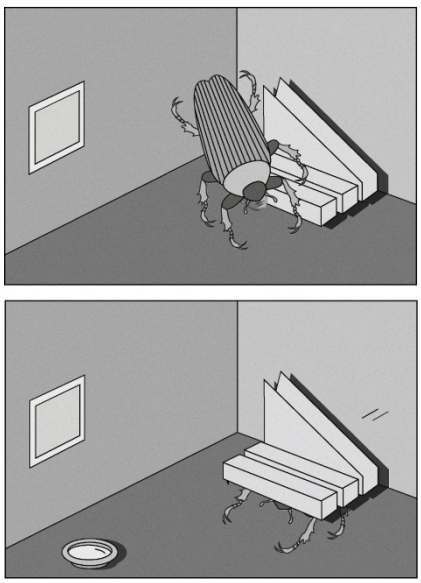

Illustration 3: Kapitel 7 (F. Wolf)

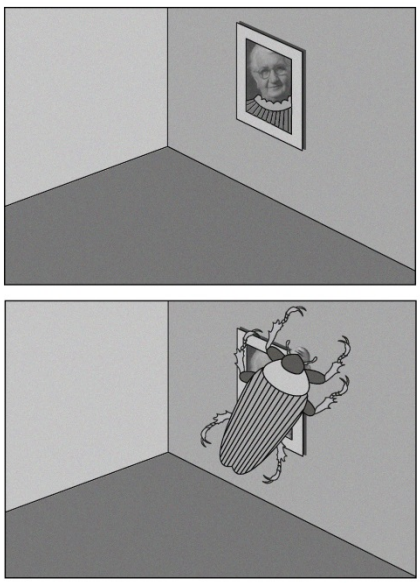

Illustration 4: Kapitel 12 (F. Wolf)

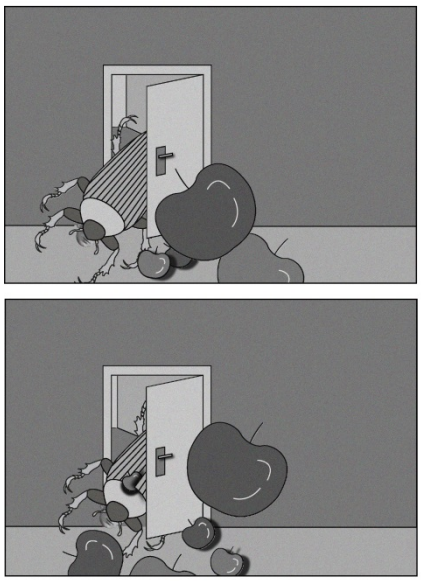

Illustration 5: Kapitel 13 (F. Wolf)

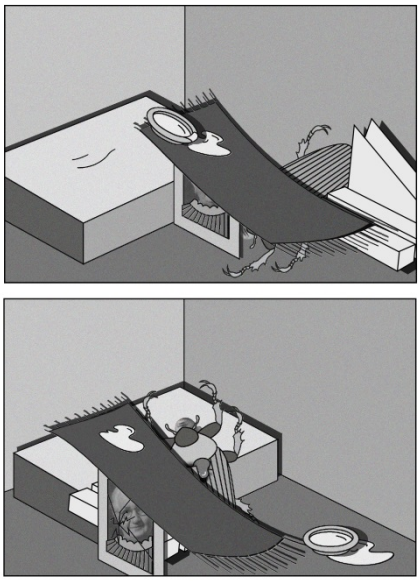

Illustration 6: Kapitel 15 (F. Wolf)

Friederike Wolf: http://www.friederikewolf.com/

Illustration 7: Autor
(http://commons.wikimedia.org/wiki/File%3AKafka1906_cropped.jpg)

2. XML-Dokument

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<programm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ma-buwi-fh.de kafka_struktur.xsd"
xmlns="http://www.ma-buwi-fh.de" xmlns:tst="http://www.ma-buwi-fh.de">

        <warengruppe tst:gruppenname="belletristik">

          <autor tst:autorname="Franz Kafka">

                <buch tst:buchname="Die Verwandlung" tst:lang="de" tst:isbn="">

                                <verfasser> Franz Kafka </verfasser>

                                <titel> Die Verwandlung </titel>

                                <kapitel tst:nr="1">

                                        <absatz>Als <person tst:idp="p1">Gregor
Samsa</person> eines Morgens aus Träumen erwachte, fand er sich in seinem Bett zu
einem ungeheueren Ungeziefer verwandelt. Er lag auf seinem panzerartig harten
Rücken und sah, wenn er den Kopf ein wenig hob, seinen gewölbten, braunen, von
bogenförmigen Versteifungen geteilten Bauch, auf dessen Höhe sich die Bettdecke,
zum gänzlichen Niedergleiten bereit, kaum noch erhalten konnte. Seine vielen, im
Vergleich zu seinem sonstigen Umfang kläglich dünnen Beine flimmerten ihm hilflos
vor den Augen.</absatz> [...] </kapitel>

                                <kapitel tst:nr="2">

                                        <absatz>Als er dies alles in größter Eile
überlegte, ohne sich entschließen zu können, das Bett zu verlassen – gerade
schlug der Wecker dreiviertel sieben – klopfte es vorsichtig an die Tür am
Kopfende seines Bettes.</absatz> [...] </kapitel>

                                <kapitel tst:nr="3">

                                        <absatz>Dann aber sagte er sich: »Ehe es
einviertel acht schlägt, muß ich unbedingt das Bett vollständig verlassen haben.
Im übrigen wird auch bis dahin jemand aus dem Geschäft kommen, um nach mir zu
fragen, denn das Geschäft wird vor sieben Uhr geöffnet.« Und er machte sich nun
daran, den Körper in seiner ganzen Länge vollständig gleichmäßig aus dem Bett
hinauszuschaukeln. Wenn er sich auf diese Weise aus dem Bett fallen ließ, blieb
der Kopf, den er beim Fall scharf heben wollte, voraussichtlich unverletzt. Der
Rücken schien hart zu sein; dem würde wohl bei dem Fall auf den Teppich nichts
geschehen. Das größte Bedenken machte ihm die Rücksicht auf den lauten Krach, den
es geben müßte und der wahrscheinlich hinter allen Türen wenn nicht Schrecken, so
doch Besorgnisse erregen würde. Das mußte aber gewagt werden.</absatz> [...]
<bild/> [...] </kapitel>

                                <kapitel tst:nr="4">

                                        <absatz>Warum ging denn die <person
```

```
tst:idp="p2">Schwester</person> nicht zu den anderen? Sie war wohl erst jetzt aus
dem Bett aufgestanden und hatte noch gar nicht angefangen sich anzuziehen. Und
warum weinte sie denn? Weil er nicht aufstand und den <person
tst:idp="p5">Prokuristen</person> nicht hereinließ, weil er in Gefahr war, den
Posten zu verlieren und weil dann der Chef die Eltern mit den alten Forderungen
wieder verfolgen würde? Das waren doch vorläufig wohl unnötige Sorgen. Noch war
<person tst:idp="p1">Gregor</person> hier und dachte nicht im geringsten daran,
seine Familie zu verlassen. Augenblicklich lag er wohl da auf dem Teppich, und
niemand, der seinen Zustand gekannt hätte, hätte im Ernst von ihm verlangt, daß er
den <person tst:idp="p5">Prokuristen</person> hereinlasse. Aber wegen dieser klei-
nen Unhöflichkeit, für die sich ja später leicht eine passende Ausrede finden
würde, konnte <person tst:idp="p1">Gregor</person> doch nicht gut sofort wegge-
schickt werden. Und <person tst:idp="p1">Gregor</person> schien es, daß es viel
vernünftiger wäre, ihn jetzt in Ruhe zu lassen, statt ihn mit Weinen und Zureden
zu stören. Aber es war eben die Ungewißheit, welche die anderen bedrängte und ihr
Benehmen entschuldigte.</absatz> [...] </kapitel>

                    <kapitel tst:nr="5">

                              <absatz><person tst:idp="p1">Gregor</person> war
aber viel ruhiger geworden. Man verstand zwar also seine Worte nicht mehr, trotz-
dem sie ihm genug klar, klarer als früher, vorgekommen waren, vielleicht infolge
der Gewöhnung des Ohres. Aber immerhin glaubte man nun schon daran, daß es mit ihm
nicht ganz in Ordnung war, und war bereit, ihm zu helfen. Die Zuversicht und Si-
cherheit, mit welchen die ersten Anordnungen getroffen worden waren, taten ihm
wohl. Er fühlte sich wieder einbezogen in den menschlichen Kreis und erhoffte von
beiden, vom Arzt und vom Schlosser, ohne sie eigentlich genau zu scheiden, großar-
tige und überraschende Leistungen. Um für die sich nähernden entscheidenden Be-
sprechungen eine möglichst klare Stimme zu bekommen, hustete er ein wenig ab,
allerdings bemüht, dies ganz gedämpft zu tun, da möglicherweise auch schon dieses
Geräusch anders als menschlicher Husten klang, was er selbst zu entscheiden sich
nicht mehr getraute. Im Nebenzimmer war es inzwischen ganz still geworden. Viel-
leicht saßen die Eltern mit dem <person tst:idp="p5">

Prokuristen</person> beim Tisch und tuschelten, vielleicht lehnten alle an der
Türe und horchten.</absatz> [...] </kapitel>

                    <kapitel tst:nr="6">

                              <absatz>Aber der <person
tst:idp="p5">Prokurist</person> hatte sich schon bei den ersten Worten <person
tst:idp="p1">Gregors</person> abgewendet, und nur über die zuckende Schulter hin-
weg sah er mit aufgeworfenen Lippen nach <person tst:idp="p1">Gregor</person>
zurück. Und während <person tst:idp="p1">Gregors</person> Rede stand er keinen
Augenblick still, sondern verzog sich, ohne <person tst:idp="p1">Gregor</person>
aus den Augen zu lassen, gegen die Tür, aber ganz allmählich, als bestehe ein
geheimes Verbot, das Zimmer zu verlassen. Schon war er im Vorzimmer, und nach der
plötzlichen Bewegung, mit der er zum letztenmal den Fuß aus dem Wohnzimmer zog,
hätte man glauben können, er habe sich soeben die Sohle verbrannt. Im Vorzimmer
aber streckte er die rechte Hand weit von sich zur Treppe hin, als warte dort auf
ihn eine geradezu überirdische Erlösung. </absatz> [...] </kapitel>

                              <kapitel tst:nr="7">

                                        <absatz>Wenn nur nicht dieses unerträgliche
Zischen des <person tst:idp="p3">Vaters</person> gewesen wäre! <person
tst:idp="p1">Gregor</person> verlor darüber ganz den Kopf. Er war schon fast ganz
umgedreht, als er sich, immer auf dieses Zischen horchend, sogar irrte und sich
wieder ein Stück zurückdrehte. Als er aber endlich glücklich mit dem Kopf vor der
```

```
Türöffnung war, zeigte es sich, daß sein Körper zu breit war, um ohne weiteres
durchzukommen. Dem <person tst:idp="p3">Vater</person> fiel es natürlich in seiner
gegenwärtigen Verfassung auch nicht entfernt ein, etwa den anderen Türflügel zu
öffnen, um für <person tst:idp="p1">Gregor</person> einen genügenden Durchgang zu
schaffen. Seine fixe Idee war bloß, daß <person tst:idp="p1">Gregor</person> so
rasch als möglich in sein Zimmer müsse. Niemals hätte er auch die umständlichen
Vorbereitungen gestattet, die <person tst:idp="p1">Gregor</person>

 brauchte, um sich aufzurichten und vielleicht auf diese Weise durch die Tür zu
kommen. Vielmehr trieb er, als gäbe es kein Hindernis, <person
tst:idp="p1">Gregor</person> jetzt unter besonderem Lärm vorwärts; es klang schon
hinter <person tst:idp="p1">Gregor</person> gar nicht mehr wie die Stimme bloß
eines einzigen <person tst:idp="p3">Vaters</person>; nun gab es wirklich keinen
Spaß mehr, und <person tst:idp="p1">Gregor</person> drängte sich – geschehe
was wolle – in die Tür. Die eine Seite seines Körpers hob sich, er lag schief
in der Türöffnung, seine eine Flanke war ganz wundgerieben, an der weißen Tür
blieben häßliche Flecken, bald steckte er fest und hätte sich allein nicht mehr
rühren können, die Beinchen auf der einen Seite hingen zitternd oben in der Luft,
die auf der anderen waren schmerzhaft zu Boden gedrückt – da gab ihm der
<person tst:idp="p3">Vater</person> von hinten einen jetzt wahrhaftig erlösenden
starken Stoß, und er flog, heftig blutend, weit in sein Zimmer hinein. Die Tür
wurde noch mit dem Stock zugeschlagen, dann war es endlich still.</absatz> [...]
</kapitel>

                              <kapitel tst:nr="8">

                                    <absatz>Dort blieb er die ganze Nacht, die
er zum Teil im Halbschlaf, aus dem ihn der Hunger immer wieder aufschreckte, ver-
brachte, zum Teil aber in Sorgen und undeutlichen Hoffnungen, die aber alle zu dem
Schlusse führten, daß er sich vorläufig ruhig verhalten und durch Geduld und größ-
te Rücksichtnahme der Familie die Unannehmlichkeiten erträglich machen müsse, die
er ihr in seinem gegenwärtigen Zustand nun einmal zu verursachen gezwungen
war.</absatz> [...] </kapitel>

                              <kapitel tst:nr="9">

                                    <absatz>Während aber <person
tst:idp="p1">Gregor</person> unmittelbar keine Neuigkeit erfahren konnte, erhorch-
te er manches aus den Nebenzimmern, und wo er nur einmal Stimmen hörte, lief er
gleich zu der betreffenden Tür und drückte sich mit ganzem Leib an sie. Besonders
in der ersten Zeit gab es kein Gespräch, das nicht irgendwie, wenn auch nur im
geheimen, von ihm handelte. Zwei Tage lang waren bei allen Mahlzeiten Beratungen
darüber zu hören, wie man sich jetzt verhalten solle; aber auch zwischen den Mahl-
zeiten sprach man über das gleiche Thema, denn immer waren zumindest zwei Famili-
enmitglieder zu Hause, da wohl niemand allein zu Hause bleiben wollte und man die
Wohnung doch auf keinen Fall gänzlich verlassen konnte. Auch hatte das <person
tst:idp="p6">Dienstmädchen</person> gleich am ersten Tag – es war nicht ganz
klar, was und wieviel sie von dem Vorgefallenen wußte – kniefällig die <per-
son tst:idp="p4">Mutter</person> [...] </kapitel>

                              <kapitel tst:nr="10">

                                    <absatz>Nun genügte dieses Geld aber ganz
und gar nicht, um die Familie etwa von den Zinsen leben zu lassen; es genügte
vielleicht, um die Familie ein, höchstens zwei Jahre zu erhalten, mehr war es
nicht. Es war also bloß eine Summe, die man eigentlich nicht angreifen durfte, und
die für den Notfall zurückgelegt werden mußte; das Geld zum Leben aber mußte man
verdienen. Nun war aber der <person tst:idp="p3">Vater</person> ein zwar gesunder,
aber alter Mann, der schon fünf Jahre nichts gearbeitet hatte und sich jedenfalls
```

```
nicht viel zutrauen durfte; er hatte in diesen fünf Jahren, welche die ersten
Ferien seines mühevollen und doch erfolglosen Lebens waren, viel Fett angesetzt
und war dadurch recht schwerfällig geworden. Und die alte <person
tst:idp="p4">Mutter</person> sollte nun vielleicht Geld verdienen, die an Asthma
litt, der eine Wanderung durch die Wohnung schon Anstrengung verursachte, und die
jeden zweiten Tag in Atembeschwerden auf dem Sopha beim offenen Fenster verbrach-
te? Und die <person tst:idp="p2">Schwester</person> sollte Geld verdienen, die
noch ein Kind war mit ihren siebzehn Jahren, und der ihre bisherige Lebensweise so
sehr zu gönnen war, die daraus bestanden hatte, sich nett zu kleiden, lange zu
schlafen, in der Wirtschaft mitzuhelfen, an ein paar bescheidenen Vergnügungen
sich zu beteiligen und vor allem Violine zu spielen? Wenn die Rede auf diese Not-
wendigkeit des Geldverdienens kam, ließ zuerst immer <person
tst:idp="p1">Gregor</person> die Türe los und warf sich auf das neben der Tür
befindliche kühle Ledersofa, denn ihm war ganz heiß vor Beschämung und Trau-
er.</absatz> [...] </kapitel>

                              <kapitel tst:nr="11">

                                    <absatz>Der Wunsch <person
tst:idp="p1">Gregors</person>, die <person tst:idp="p4">Mutter</person> zu sehen,
ging bald in Erfüllung. Während des Tages wollte <person
tst:idp="p1">Gregor</person> schon aus Rücksicht auf seine Eltern sich nicht beim
Fenster zeigen, kriechen konnte er aber auf den paar Quadratmetern des Fußbodens
auch nicht viel, das ruhige Liegen ertrug er schon während der Nacht schwer, das
Essen machte ihm bald nicht mehr das geringste Vergnügen, und so nahm er zur Zer-
streuung die Gewohnheit an, kreuz und quer über Wände und Plafond zu kriechen.
Besonders oben auf der Decke hing er gern; es war ganz anders, als das Liegen auf
dem Fußboden; man atmete freier; ein leichtes Schwingen ging durch den Körper; und
in der fast glücklichen Zerstreutheit, in der sich <person
tst:idp="p1">Gregor</person> dort oben befand, konnte es geschehen, daß er zu
seiner eigenen Überraschung sich losließ und auf den Boden klatschte. Aber nun
hatte er natürlich seinen Körper ganz anders in der Gewalt als früher und beschä-
digte sich selbst bei einem so großen Falle nicht. Die <person
tst:idp="p2">Schwester</person> nun bemerkte sofort die neue Unterhaltung, die
<person tst:idp="p1">Gregor</person> für sich gefunden hatte – er hinterließ
ja auch beim Kriechen hie und da Spuren seines Klebstoffes –, und da setzte
sie es sich in den Kopf, <person tst:idp="p1">Gregor</person> das Kriechen in
größtem Ausmaße zu ermöglichen und die Möbel, die es verhinderten, also vor allem
den Kasten und den Schreibtisch, wegzuschaffen.</absatz> [...] </kapitel>

                              <kapitel tst:nr="12">

                                    <absatz>Trotzdem sich <person
tst:idp="p1">Gregor</person> immer wieder sagte, daß ja nichts Außergewöhnliches
geschehe, sondern nur ein paar Möbel umgestellt würden, wirkte doch, wie er sich
bald eingestehen mußte, dieses Hin- und Hergehen der Frauen, ihre kleinen Zurufe,
das Kratzen der Möbel auf dem Boden, wie ein großer, von allen Seiten genährter
Trubel auf ihn, und er mußte sich, so fest er Kopf und Beine an sich zog und den
Leib bis an den Boden drückte, unweigerlich sagen, daß er das Ganze nicht lange
aushalten werde. Sie räumten ihm sein Zimmer aus; nahmen ihm alles, was ihm lieb
war; den Kasten, in dem die Laubsäge und andere Werkzeuge lagen, hatten sie schon
hinausgetragen; lockerten jetzt den schon im Boden fest eingegrabenen Schreib-
tisch, an dem er als Handelsakademiker, als Bürgerschüler, ja sogar schon als
Volksschüler seine Aufgaben geschrieben hatte, – da hatte er wirklich keine
Zeit mehr, die guten Absichten zu prüfen, welche die zwei Frauen hatten, deren
Existenz er übrigens fast vergessen hatte, denn vor Erschöpfung arbeiteten sie
schon stumm, und man hörte nur das schwere Tappen ihrer Füße.</absatz> [...]
</kapitel>
```

```
                                <kapitel tst:nr="13">

                                        <absatz>Nun aber war er recht gut aufge-
richtet; in eine straffe blaue Uniform mit Goldknöpfen gekleidet, wie sie Diener
der Bankinstitute tragen; über dem hohen steifen Kragen des Rockes entwickelte
sich sein starkes Doppelkinn; unter den buschigen Augenbrauen drang der Blick der
schwarzen Augen frisch und aufmerksam hervor; das sonst zerzauste weiße Haar war
zu einer peinlich genauen, leuchtenden Scheitelfrisur niedergekämmt. Er warf seine
Mütze, auf der ein Goldmonogramm, wahrscheinlich das einer Bank, angebracht war,
über das ganze Zimmer im Bogen auf das Kanapee hin und ging, die Enden seines
langen Uniformrockes zurückgeschlagen, die Hände in den Hosentaschen, mit vebisse-
nem Gesicht auf <person tst:idp="p1">Gregor</person> zu.</absatz> [...] </kapitel>

                                <kapitel tst:nr="14">

                                        <absatz>Sobald die Uhr zehn schlug, suchte
die <person tst:idp="p4">Mutter</person> durch leise Zusprache den <person
tst:idp="p3">Vater</person> zu wecken und dann zu überreden, ins Bett zu gehen,
denn hier war es doch kein richtiger Schlaf und diesen hatte der <person
tst:idp="p3">Vater</person>, der um sechs Uhr seinen Dienst antreten mußte, äu-
ßerst nötig. Aber in dem Eigensinn, der ihn, seitdem er Diener war, ergriffen
hatte, bestand er immer darauf noch länger bei Tisch zu bleiben, trotzdem er re-
gelmäßig einschlief, und war dann überdies nur mit der größten Mühe zu bewegen,
den Sessel mit dem Bett zu vertauschen. Da mochten <person
tst:idp="p4">Mutter</person> und <person tst:idp="p2">Schwester</person> mit klei-
nen Ermahnungen noch so sehr auf ihn eindringen, viertelstundenlang schüttelte er
langsam den Kopf hielt, die Augen geschlossen und stand nicht auf. Die <person
tst:idp="p4">Mutter</person> zupfte ihn am Ärmel, sagte ihm Schmeichelworte ins
Ohr, die <person tst:idp="p2">Schwester</person> verließ ihre Aufgabe, um der
<person tst:idp="p4">Mutter</person> zu helfen, aber beim <person
tst:idp="p3">Vater</person> verfing das nicht. Er versank nur noch tiefer in sei-
nen Sessel. Erst bis ihn die Frauen unter den Achseln faßten, schlug er die Augen
auf, sah abwechselnd die <person tst:idp="p4">Mutter</person> und die <person
tst:idp="p2">Schwester</person> an und pflegte zu sagen: »Das ist ein Leben. Das
ist die Ruhe meiner alten Tage.« Und auf die beiden Frauen gestützt, erhob er
sich, umständlich, als sei er für sich selbst die größte Last, ließ sich von den
Frauen bis zur Türe führen, winkte ihnen dort ab und ging nun selbständig weiter,
während die <person tst:idp="p4">Mutter</person> ihr Nähzeug, die <person
tst:idp="p2">Schwester</person> ihre Feder eiligst hinwarfen, um hinter dem <per-
son tst:idp="p3">Vater</person> zu laufen und ihm weiter behilflich zu
sein.</absatz> [...] </kapitel>

                                <kapitel tst:nr="15">

                                        <absatz>Aber selbst wenn die <person
tst:idp="p2">Schwester</person>, erschöpft von ihrer Berufsarbeit, dessen über-
drüssig geworden war, für <person tst:idp="p1">Gregor</person>, wie früher, zu
sorgen, so hätte noch keineswegs die <person tst:idp="p4">Mutter</person> für sie
eintreten müssen und <person tst:idp="p1">Gregor</person> hätte doch nicht ver-
nachlässigt werden brauchen. Denn nun war die Bedienerin da. Diese alte Witwe, die
in ihrem langen Leben mit Hilfe ihres starken Knochenbaues das Ärgste überstanden
haben mochte, hatte keinen eigentlichen Abscheu vor <person
tst:idp="p1">Gregor</person>. Ohne irgendwie neugierig zu sein, hatte sie zufällig
einmal die Tür von <person tst:idp="p1">Gregors</person> Zimmer aufgemacht und war
im Anblick <person tst:idp="p1">Gregors</person>, der, gänzlich überrascht, trotz-
dem ihn niemand jagte, hin und herzulaufen begann, die Hände im Schoß gefaltet
staunend stehen geblieben. Seitdem versäumte sie nicht, stets flüchtig morgens und
abends die Tür ein wenig zu öffnen und zu <person tst:idp="p1">Gregor</person>
hineinzuschauen. Anfangs rief sie ihn auch zu sich herbei, mit Worten, die sie
```

wahrscheinlich für freundlich hielt, wie »Komm mal herüber, alter Mistkäfer!« oder »Seht mal den alten Mistkäfer!« Auf solche Ansprachen antwortete <person tst:idp="p1">Gregor</person> mit nichts, sondern blieb unbeweglich auf seinem Platz, als sei die Tür gar nicht geöffnet worden. Hätte man doch dieser Bedienerin, statt sie nach ihrer Laune ihn nutzlos stören zu lassen, lieber den Befehl gegeben, sein Zimmer täglich zu reinigen! Einmal am frühen Morgen – ein heftiger Regen, vielleicht schon ein Zeichen des kommenden Frühjahrs, schlug an die Scheiben – war <person tst:idp="p1">Gregor</person>, als die Bedienerin mit ihren Redensarten wieder begann, derartig erbittert, daß er, wie zum Angriff, allerdings langsam und hinfällig, sich gegen sie wendete. Die Bedienerin aber, statt sich zu fürchten, hob bloß einen in der Nähe der Tür befindlichen Stuhl hoch empor, und wie sie mit groß geöffnetem Munde dastand, war ihre Absicht klar, den Mund erst zu schließen, wenn der Sessel in ihrer Hand auf <person tst:idp="p1">Gregors</person> Rücken niederschlagen würde. »Also weiter geht es nicht?« fragte sie, als <person tst:idp="p1">Gregor</person> sich wieder umdrehte, und stellte den Sessel ruhig in die Ecke zurück.</absatz>[...] </kapitel>

<kapitel tst:nr="16">

<absatz>Gerade an diesem Abend –
 <person tst:idp="p1">Gregor</person> erinnerte sich nicht, während der ganzen Zeit die Violine gehört zu haben – ertönte sie von der Küche her. Die Zimmerherren hatten schon ihr Nachtmahl beendet, der mittlere hatte eine Zeitung hervorgezogen, den zwei anderen je ein Blatt gegeben, und nun lasen sie zurückgelehnt und rauchten. Als die Violine zu spielen begann, wurden sie aufmerksam, erhoben sich und gingen auf den Fußspitzen zur Vorzimmertür, in der sie aneinandergedrängt stehen blieben. Man mußte sie von der Küche aus gehört haben, denn der <person tst:idp="p3">Vater</person> rief: »Ist den Herren das Spiel vielleicht unangenehm? Es kann sofort eingestellt werden.« »Im Gegenteil«, sagte der mittlere der Herren, »möchte das Fräulein nicht zu uns hereinkommen und hier im Zimmer spielen, wo es doch viel bequemer und gemütlicher ist?« »O bitte«, rief der <person tst:idp="p3">Vater</person>, als sei er der Violinspieler. Die Herren traten ins Zimmer zurück und warteten. Bald kam der <person tst:idp="p3">Vater</person> mit dem Notenpult, die <person tst:idp="p4">Mutter</person> mit den Noten und die <person tst:idp="p2">Schwester</person> mit der Violine. Die <person tst:idp="p2">Schwester</person> bereitete alles ruhig zum Spiele vor; die Eltern, die niemals früher Zimmer vermietet hatten und deshalb die Höflichkeit gegen die Zimmerherren übertrieben, wagten gar nicht, sich auf ihre eigenen Sessel zu setzen; der <person tst:idp="p3">Vater</person> lehnte an der Tür, die rechte Hand zwischen zwei Knöpfe des geschlossenen Livreerockes gesteckt; die <person tst:idp="p4">Mutter</person> aber erhielt von einem Herrn einen Sessel angeboten und saß, da sie den Sessel dort ließ, wohin ihn der Herr zufällig gestellt hatte, abseits in einem Winkel.</absatz> [...] </kapitel>

<kapitel tst:nr="17">

<absatz>Der <person tst:idp="p3">Vater</person> wankte mit tastenden Händen zu seinem Sessel und ließ sich in ihn fallen; es sah aus, als strecke er sich zu seinem gewöhnlichen Abendschläfchen, aber das starke Nicken seines wie haltlosen Kopfes zeigte, daß er ganz und gar nicht schlief. <person tst:idp="p1">Gregor</person> war die ganze Zeit still auf dem Platz gelegen, auf dem ihn die Zimmerherren ertappt hatten. Die Enttäuschung über das Mißlingen seines Planes, vielleicht aber auch die durch das viele Hungern verursachte Schwäche machten es ihm unmöglich, sich zu bewegen. Er fürchtete mit einer gewissen Bestimmtheit schon für den nächsten Augenblick einen allgemeinen über ihn sich entladenden Zusammensturz und wartete. Nicht einmal die Violine schreckte ihn auf, die, unter den zitternden Fingern der <person tst:idp="p4">Mutter</person> hervor, ihr vom Schoße fiel und einen hallenden Ton

```
von sich gab.</absatz> [...] </kapitel>

                                <kapitel tst:nr="18">

                                        <absatz>Erst als er schon in der Tür war,
wendete er den Kopf, nicht vollständig, denn er fühlte den Hals steif werden,
immerhin sah er noch, daß sich hinter ihm nichts verändert hatte, nur die <person
tst:idp="p2">Schwester</person> war aufgestanden. Sein letzter Blick streifte die
<person tst:idp="p4">Mutter</person>, die nun völlig eingeschlafen war.</absatz>
[...] </kapitel>

                                <kapitel tst:nr="19">

                                        <absatz>Sie beschlossen, den heutigen Tag
zum Ausruhen und Spazierengehen zu verwenden; sie hatten diese Arbeitsunterbre-
chung nicht nur verdient, sie brauchten sie sogar unbedingt. Und so setzten sie
sich zum Tisch und schrieben drei Entschuldigungsbriefe, <person tst:idp="p3">Herr
Samsa</person> an seine Direktion, <person tst:idp="p4">Frau Samsa</person> an
ihren Auftraggeber, und <person tst:idp="p2">Grete</person> an ihren Prinzipal.
Während des Schreibens kam die Bedienerin herein, um zu sagen, daß sie fortgehe,
denn ihre Morgenarbeit war beendet. Die drei Schreibenden nickten zuerst bloß,
ohne aufzuschauen, erst als die Bedienerin sich immer noch nicht entfernen wollte,
sah man ärgerlich auf. »Nun?« fragte <person tst:idp="p3">Herr Samsa</person>. Die
Bedienerin stand lächelnd in der Tür, als habe sie der Familie ein großes Glück zu
melden, werde es aber nur dann tun, wenn sie gründlich ausgefragt werde. Die fast
aufrechte kleine Straußfeder auf ihrem Hut, über die sich <person
tst:idp="p3">Herr Samsa</person> schon während ihrer ganzen Dienstzeit ärgerte,
schwankte leicht nach allen Richtungen. »Also was wollen Sie eigentlich?« fragte
<person tst:idp="p4">Frau Samsa</person>, vor welcher die Bedienerin noch am meis-
ten Respekt hatte. »Ja«, antwortete die Bedienerin und konnte vor freundlichem
Lachen nicht gleich weiter reden, »also darüber, wie das Zeug von nebenan wegge-
schafft werden soll, müssen Sie sich keine Sorge machen. Es ist schon in Ordnung.«
<person tst:idp="p4">Frau Samsa</person> und <person tst:idp="p2">Grete</person>
beugten sich zu ihren Briefen nieder, als wollten sie weiterschreiben; <person
tst:idp="p3">Herr Samsa</person>, welcher merkte, daß die Bedienerin nun alles
ausführlich zu beschreiben anfangen wollte, wehrte dies mit ausgestreckter Hand
entschieden ab. Da sie aber nicht erzählen durfte, erinnerte sie sich an die große
Eile, die sie hatte, rief offenbar beleidigt: »Adjes allseits«, drehte sich wild
um und verließ unter fürchterlichem Türezuschlagen die Wohnung.</absatz> [...]
</kapitel>

                        </buch>

                </autor>

        </warengruppe>

</programm>
```

kafka_verwandlung.xml

3. XML-Schema-Definition

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tst="http://www.ma-
buwi-fh.de" targetNamespace="http://www.ma-buwi-fh.de" element-
FormDefault="qualified" attributeFormDefault="qualified">

        <!-- Wurzelelement Dokumentinstanz -->

        <xsd:element name="programm" type="tst:programm"/>

        <!-- Definition Programm -->

        <xsd:complexType name="programm">

                <xsd:all>

                        <xsd:element name="warengruppe" type="tst:autorenstamm"/>

                </xsd:all>

        </xsd:complexType>

        <!-- Definition Autorenstamm -->

        <xsd:complexType name="autorenstamm">

                <xsd:all>

                        <xsd:element name="autor" type="tst:werke"/>

                </xsd:all>

                <xsd:attribute name="gruppenname" use="required">

                        <xsd:simpleType>

                                <xsd:restriction base="xsd:string">

                                        <xsd:enumeration value="belletristik"/>

                                        <xsd:enumeration value="kinderjugendbuch"/>

                                        <xsd:enumeration value="reise"/>

                                        <xsd:enumeration value="ratgeber"/>

                                    <xsd:enumeration
                        value="geisteswissenschaften"/>

                                    <xsd:enumeration
                        value="naturwissenschaften"/>

                                    <xsd:enumeration
                        value="sozialwissenschaften"/>

                                        <xsd:enumeration value="schule"/>
```

```
                                    <xsd:enumeration value="sachbuch"/>
                            </xsd:restriction>
                    </xsd:simpleType>
            </xsd:attribute>
    </xsd:complexType>
    <!-- Definition Werke-->
    <xsd:complexType name="werke">
            <xsd:sequence>
                    <xsd:element name="buch" type="tst:einzelwerk">
                            <xsd:key name="kapkey">
                                    <xsd:selector xpath="tst:kapitel"/>
                                    <xsd:field xpath="@tst:nr"/>
                            </xsd:key>
                    </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="autorname" type="xsd:string" use="required"/>
    </xsd:complexType>
    <!-- Definition Einzelwerk -->
    <xsd:complexType name="einzelwerk">
            <xsd:sequence>
                    <xsd:element name="verfasser" type="xsd:string"/>
                    <xsd:element name="titel" type="xsd:string"/>
                    <xsd:element name="kapitel" type="tst:buchkapitel"
            maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="buchname" type="xsd:string" use="required"/>
            <xsd:attribute name="lang" type="xsd:language" default="de"/>
            <xsd:attribute name="isbn" type="xsd:token" use="required"/>
    </xsd:complexType>
    <!-- Definition Buchkapitel -->
    <xsd:complexType name="buchkapitel">
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
```

```
                        <xsd:choice>
                                <xsd:element name="absatz" type="tst:buchabsatz"/>
                                <xsd:element name="bild" type="tst:buchbild"/>

                        </xsd:choice>
                </xsd:sequence>
                <xsd:attribute name="nr"
          type="xsd:positiveInteger" use="required"/>
        </xsd:complexType>
        <!-- Definition Buchabsatz -->
        <xsd:complexType name="buchabsatz" mixed="true">
                <xsd:sequence>
                        <xsd:element name="person" type="tst:buchperson"
              minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
        </xsd:complexType>
        <!-- Definition Buchbild -->
        <xsd:complexType name="buchbild">
                <xsd:sequence>
                        <xsd:element name="bild" minOccurs="0"
              maxOccurs="unbounded"/>
                </xsd:sequence>
        </xsd:complexType>
        <!-- Definition Buchperson -->
        <xsd:complexType name="buchperson">
                <xsd:simpleContent>
                        <xsd:extension base="xsd:string">
                                <xsd:attribute name="idp" use="required">
                                        <xsd:simpleType>
                                                <xsd:restriction base="xsd:string">
                                                        <xsd:enumeration
                                 value="p1"/>
                                                        <xsd:enumeration
```

```
                                value="p2"/>
                                                        <xsd:enumeration
                                value="p3"/>
                                                        <xsd:enumeration
                                value="p4"/>
                                                        <xsd:enumeration
                                value="p5"/>
                                                        <xsd:enumeration
                                value="p6"/>
                                                </xsd:restriction>
                                        </xsd:simpleType>
                                </xsd:attribute>
                        </xsd:extension>
                </xsd:simpleContent>
        </xsd:complexType>
</xsd:schema>
```

kafka_struktur.xsd

4. XSLT-Stylesheet

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

<xsl:stylesheet version="2.0"

        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

        xmlns:fn="http://www.w3.org/2005/xpath-functions"

        xmlns:tst="http://www.ma-buwi-fh.de"

        xmlns="http://www.w3.org/1999/xhtml">



<xsl:output method="xhtml"

  encoding="UTF-8"

  indent="yes"

  doctype-public='-//W3C//DTD XHTML 1.1//EN'

  doctype-system='http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd'

  />



<xsl:strip-space elements="*"/>

<xsl:preserve-space elements="tst:absatz"/>



<xsl:template match="/">

<html xml:lang="de" lang="de">

<!-- Angaben zum Titel -->

<head>

<title><xsl:value-of select="(//@tst:autorname, //@tst:buchname, //tst:isbn)"
separator=", "/></title>

<!-- CSS-Integration -->

<link href="../Typo/text.css" type="text/css" rel="stylesheet"/>

</head>

<!-- Anzuzeigender Inhalt -->

<body>

<!-- Reihenfolge der Ausgabe -->

<xsl:apply-templates sel-
ect="tst:programm/tst:warengruppe/tst:autor/tst:buch/tst:verfasser"/>
```

```
<xsl:apply-templates sel-
ect="tst:programm/tst:warengruppe/tst:autor/tst:buch/tst:titel"/>

<xsl:apply-templates se-
lect="tst:programm/tst:warengruppe/tst:autor/tst:buch/tst:kapitel"/>

</body>

</html>

</xsl:template>



<!-- Ausgaberegel Verfasser -->

<xsl:template match="tst:verfasser">

<h1 id="{generate-id()}"><xsl:value-of select="self::node()"/></h1>

</xsl:template>



<!-- Ausgaberegel Titel -->

<xsl:template match="tst:titel">

<h2 id="{generate-id()}"><xsl:value-of select="self::node()"/></h2>

</xsl:template>



<!-- Ausgaberegel Kapitel -->

<xsl:template match="tst:kapitel">

<div><b id="{generate-id()}">Kapitel <xsl:number format="1"
count="tst:kapitel"/></b>

<xsl:apply-templates select="child::node()"/>

</div>

</xsl:template>



<!-- Ausgaberegel Absatz -->

<xsl:template match="tst:absatz">

<xsl:choose>

        <xsl:when test="fn:position() = 1">

                <p><span class="initial"><xsl:value-of se-
lect="substring(.,1,1)"/></span>

                        <xsl:value-of select="substring(.,2)"/>
```

```
                </p>

        </xsl:when>

        <xsl:otherwise>

                <p><xsl:value-of select="."/></p>

        </xsl:otherwise>

</xsl:choose>

</xsl:template>



<xsl:template match="tst:bild">

        <img src="../Bilder/kapitel{../@tst:nr}.jpg" alt="Bild Kapitel
{../@tst:nr}"/>

</xsl:template>



</xsl:stylesheet>
```

kafka_xslt.xsl

5. XSL-FO-Stylesheet

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

<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

        xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:tst="http://www.ma-
buwi-fh.de" xmlns:fn="http://www.w3.org/2005/xpath-functions">

        <xsl:template match="/">

                <!-- Wurzelelement Layoutvorlage -->

                <fo:root>

                        <fo:layout-master-set>

                                <!-- Seitenvorlagen -->

                                <!-- Text -->

                                <fo:simple-page-master master-name="standardseite"

                                        page-height="300mm" page-width="210mm"

                                        margin-top="15mm" margin-left="30mm"

                                        margin-bottom="10mm" margin-right="30mm">

                                        <fo:region-body region-name="textkorpus"

                                        margin-top="20mm" margin-left="5mm"

                                                margin-bottom="20mm"

                                                margin-right="5mm"/>

                                        <fo:region-before region-name="kopfzeile"

                                                extent="10mm"/>

                                        <fo:region-after region-name="fusszeile"

                                                extent="10mm"/>

                                </fo:simple-page-master>

                                <!-- Bilder -->

                                <fo:simple-page-master master-name="bildseite"

                                        page-height="300mm" page-width="210mm"

                                        margin-top="15mm" margin-left="30mm"

                                        margin-bottom="10mm" margin-right="30mm">

                                        <fo:region-body region-name="bildkorpus"

                                        margin-top="5mm" margin-left="5mm"
```

```
                                        margin-bottom="5mm"
                    margin-right="5mm"/>
                            </fo:simple-page-master>
                        <!-- Seitenverlaufsvorlagen -->
                        <fo:page-sequence-master master-name="titelbild">
                            <fo:single-page-master-reference
                                master-reference="bildseite"/>
                        </fo:page-sequence-master>

                        <fo:page-sequence-master master-name="titelei">
                            <fo:single-page-master-reference
                                master-reference="standardseite">
                        </fo:single-page-master-reference>
                        </fo:page-sequence-master>
                        <fo:page-sequence-master
                master-name="inhaltsverzeichnis">
                            <fo:single-page-master-reference
                                master-reference="standardseite"/>
                        </fo:page-sequence-master>
                        <fo:page-sequence-master master-name="kapitel">
                            <fo:repeatable-page-master-alternatives>
                                <fo:conditional-page-
                      master-reference
                                master-reference="standardseite"/>
                            </fo:repeatable-page-master-alternatives>
                        </fo:page-sequence-master>
                        <fo:page-sequence-master master-name="autor">
                            <fo:single-page-master-reference
                                master-reference="bildseite"/>
                        </fo:page-sequence-master>
                    </fo:layout-master-set>
                    <!-- Konkreter Seitenverlauf -->
                    <!-- Titelbild -->
```

```
<fo:page-sequence master-reference="titelbild">
        <fo:flow flow-name="bildkorpus">
                <fo:block text-align="center">
                        <fo:external-graphic
                         src="C:/Users/Felix/Desktop/
  Daten Masterarbeit/
                         XSL-FO/Bilder XSL-
  FO/titelbild.jpg"
                         content-height="80%" content-
  width="75%"/>
                </fo:block>
        </fo:flow>
 </fo:page-sequence>
 <!-- Titelei -->
 <fo:page-sequence master-reference="titelei">
        <fo:flow flow-name="textkorpus">
                <fo:block>
                        <xsl:apply-templates
   select="//tst:verfasser |
                                //tst:titel"/>
                </fo:block>
                <fo:block font-family="Times"
font-size="10pt"
                        margin-top="150mm" text-
   align="center">
                        <fo:block>XSL-FO-Output;
    Masterarbeit Felix
    Hau&#223;er
                        </fo:block>
                    <fo:block>Das Dokument entstand
   im Zuge einer Masterarbeit
                     zum Thema XML-basierte
   Anreicherung von Textdateien
   am Institut f&#25r <fo:basic-
   link external-destination=
```

```
                    "http://www.buchwiss.uni-
                     erlangen.de/institut/"
                                color="gray" text-
                     decoration="underline">
                                                Buchwissenschaft
                     </fo:basic-link> der FAU
                     Erlangen-N&#252;rnberg.
                   </fo:block>
                                        <fo:block>Der verwendete Text
                    stammt vom <fo:basic-link
                    external destination=
                    "http://gutenberg.spiegel.de/
                    buch/165/1" color="gray"
                    text-decoration="underline">
                                             Projekt Gutenberg-DE!
                    </fo:basic-link>
                  </fo:block>
                                        <fo:block>Gestaltung und
                    Illustration:
                 <fo:basic-link external-
                     destination=
                     "http://www.friederikewolf.
                     com/"color="gray" text-
                     decoration="underline">
                     Friederike Wolf
                                                </fo:basic-link></fo:block>
                                    </fo:block>
                          </fo:flow>
                </fo:page-sequence>
                <!-- Inhaltsverzeichnis -->
                <fo:page-sequence master-reference="inhaltsverzeichnis">
                        <fo:flow flow-name="textkorpus">
                                <fo:block font-family="Times"
                                        font-size="12pt" font-weight="bold"
                                        margin-bottom="5mm">
```

```
                                        Inhaltsverzeichnis
                                </fo:block>
                                <xsl:for-each select="//tst:kapitel">
                                        <xsl:call-template name="ivz"/>
                                </xsl:for-each>
                        </fo:flow>
                </fo:page-sequence>
                <!-- Kapitel -->
                <fo:page-sequence master-reference="kapitel">
                        <!-- Paratext -->
                        <fo:static-content flow-name="kopfzeile">
                                <fo:block text-align="right"
        font-family="Times"
                                 font-size="10pt" font-style="italic"
                                 margin-right="15pt"> <xsl:value-of select=
                                 "(//tst:verfasser, '-', //tst:titel)"/>
                                </fo:block>
                        </fo:static-content>
                        <fo:static-content flow-name="fusszeile">
                                <fo:block text-align="right"
         margin-right="15pt"
         font-family="Times" font-size="10pt">
                                        <fo:page-number/>
                                </fo:block>
                        </fo:static-content>
                        <!-- Fliesstext -->
                        <fo:flow flow-name="textkorpus">
                                <xsl:apply-templates
          select="//tst:kapitel"/>
                        </fo:flow>
                </fo:page-sequence>
                <!-- Autor -->
```

```
                        <fo:page-sequence master-reference="autor">
                                <fo:flow flow-name="bildkorpus">
                                        <fo:block text-align="center">
                                                <fo:external-graphic src=
                                                 "C:/Users/Felix/Desktop/Daten
                        Masterarbeit/XSL-FO/Bilder
                        XSL-FO/autor.jpg"
                                                  content-height="100%"
                        content-width="100%"/>
                                        </fo:block>
                                </fo:flow>
                        </fo:page-sequence>
                </fo:root>
        </xsl:template>


        <!-- Templates -->
        <!-- Template Kapitel -->
        <xsl:template match="tst:kapitel">
                <fo:block font-family="Amatic Bold" font-size="24pt" color="gray"
                        margin-bottom="10mm" id="{generate-id()}">Kapitel
                        <xsl:value-of select="./@tst:nr"/>
                </fo:block>
                <xsl:apply-templates select="element()"/>
                <xsl:if test="fn:not(fn:position() = fn:last())">
                        <fo:block page-break-before="always"/></xsl:if>
        </xsl:template>
        <!-- Template Absatz -->
        <xsl:template match="tst:absatz">
                <xsl:choose>
                        <!-- Erster Absatz - Kein Einzug, Initiale // F.O.P.
             untersuetzt weder Drop Caps (<fo:float>) noch fixierte
             Zeilenhoehe (line-stacking-) -->
                        <xsl:when test="fn:position() = 1">
```

```
                        <fo:block font-family="Times"
            font-size="10pt" text-align="justify"
                         hyphenate="true" language="de"
            line-stacking-strategy="line-height"
                         line-height="12pt">
                                <fo:inline font-size="14pt"><xsl:value-of
               select="substring(.,1,1)"/>
                                </fo:inline><xsl:value-of
               select="substring(.,2)"/></fo:block>
                   </xsl:when>
                   <!-- Zweiter Absatz, Einzug, keine Initiale -->
                   <xsl:otherwise>
                           <fo:block font-family="Times" font-size="10pt"
               line-height="12pt"
                               text-align="justify" hyphenate="true"
               language="de"
               widows="3" orphans="3" text-indent="10pt"
               space-after="1pt"
                                keep-with-next.within-page="always">
                                <xsl:value-of select="."/></fo:block>
                   </xsl:otherwise>
           </xsl:choose>
   </xsl:template>
   <!-- Template Bilder F.O.P. unterstuetzt keinen Textumfluss (float) -->
   <xsl:template match="tst:bild">
                <fo:block text-align="center" space-before="6pt"
        space-after="6pt"
                 float="left">
                        <fo:external-graphic src=
                        "C:/Users/Felix/Desktop/Daten Masterarbeit/
          XSL-FO/Bilder
                          XSL-FO/kapitel{../@tst:nr}.jpg"
          content-width="145mm"/>
                </fo:block>
   </xsl:template>
```

```
        <!-- Template Titelei -->
        <xsl:template match="tst:verfasser">
                <fo:block font-family="Times" font-size="22pt"
         text-align="center">
                        <xsl:value-of select="."/>
                </fo:block>
        </xsl:template>
        <xsl:template match="tst:titel">
                <fo:block font-family="Amatic Bold" font-size="50pt" color="gray"
                        margin-top="10mm" text-align="center">
                        <xsl:value-of select="."/>
                </fo:block>
        </xsl:template>
        <!-- Template Inhaltsverzeichnis -->
        <xsl:template name="ivz">
          <fo:block text-align-last="justify">
                <fo:table width="130mm">
                        <fo:table-column column-number="1" column-width="120mm"/>
                        <fo:table-column column-number="2" column-width="10mm"/>
                        <fo:table-body>
                                <fo:table-row>
                                        <fo:table-cell column-number="1">
                                                <fo:block color="gray"
                           text-decoration="underline">
                                                 <fo:basic-link
                           internal-
                           destination="{generate-id()}">
                                                        Kapitel<xsl:text>
                            </xsl:text><xsl:number
                                                        count="tst:kapitel"
                             format="1"/>
                                                        </fo:basic-link>
```

```
                                                    <fo:inline color="black">
                                                      <fo:leader leader-
                                  pattern="dots"/>
                                                       </fo:inline>
                                                    </fo:block>
                                               </fo:table-cell>
                                               <fo:table-cell column-number="2">
                                                  <fo:block>
                                                        <fo:page-number-citation
                             ref-id="{generate-id()}"/>
                                                  </fo:block>
                                               </fo:table-cell>
                                          </fo:table-row>
                                     </fo:table-body>
                                </fo:table>
                           </fo:block>
                      </xsl:template>
</xsl:stylesheet>
```

kafka_xsl-fo.xsl

6. EPUB

```
<?xml version="1.0" ?>
<container xmlns="urn:oasis:names:tc:opendocument:xmlns:container" version="1.0">
 <rootfiles>
  <rootfile media-type="application/oebps-package+xml" full-path="OPS/kafka.opf"/>
 </rootfiles>
</container>
```

container.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<package version="2.0" unique-identifier="kafka_ma"
xmlns:opf="http://www.idpf.org/2007/opf"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.idpf.org/2007/opf" >
 <metadata>
        <!-- Metadaten des E-Books -->
        <dc:title>Die Verwandlung</dc:title>
        <dc:creator opf:role="aut" opf:file-as="Kafka, Franz">
    Franz Kafka</dc:creator>
        <dc:identifier id="kafka_ma">KafkaFHBuwi</dc:identifier>
        <dc:language>de</dc:language>
        <dc:publisher>Felix Haußer</dc:publisher>
        <dc:subject>Surrealismus</dc:subject>
        <dc:subject>Novelle</dc:subject>
        <dc:type>Text</dc:type>
        <dc:rights>gemeinfrei</dc:rights>
```

```
</metadata>

<manifest>

        <!-- Einband vorne -->

        <item id="titelseite" href="Texte/titelseite.xhtml"

      media-type="application/xhtml+xml"/>

        <!-- Einband hinten -->

        <item id="rueckseite" href="Texte/rueckseite.xhtml"

      media-type="application/xhtml+xml"/>

        <!-- Impressum -->

        <item id="impressum" href="Texte/impressum.xhtml"

    media-type="application/xhtml+xml"/>

        <!-- Inhalt des E-Books -->

        <item id="inhalt" href="Texte/kafka_verwandlung.xhtml"

      media-type="application/xhtml+xml"/>

        <!-- NCX-Datei -->

        <item id="ncx" href="nav.ncx" media-type="application/x-dtbncx+xml"/>

        <!-- CSS-Stylesheets -->

        <item id="css1" href="Typo/text.css" media-type="text/css"/>

        <item id="css2" href="Typo/impressum.css" media-type="text/css"/>

        <!-- Bilder -->

        <item id="bild1" href="Bilder/titelbild.jpg" media-type="image/jpeg"/>

        <item id="bild2" href="Bilder/kapitel3.jpg" media-type="image/jpeg"/>

        <item id="bild3" href="Bilder/kapitel7.jpg" media-type="image/jpeg"/>

        <item id="bild4" href="Bilder/kapitel12.jpg" media-type="image/jpeg"/>

        <item id="bild5" href="Bilder/kapitel13.jpg" media-type="image/jpeg"/>

        <item id="bild6" href="Bilder/kapitel15.jpg" media-type="image/jpeg"/>

        <item id="bild7" href="Bilder/autor.jpg" media-type="image/jpeg"/>

        <!-- Schriftart -->

        <item id="schrift_amatic1" href="Typo/AmaticSC-Regular.ttf"

      media-type="application/x-font-opentype"/>

        <item id="schrift_amatic2" href="Typo/Amatic-Bold.ttf"
```

```
       media-type="application/x-font-opentype"/>

        </manifest>

 <spine toc="ncx">

        <itemref idref="titelseite" linear="no"/>

        <itemref idref="impressum" linear="yes"/>

        <itemref idref="inhalt" linear="yes"/>

        <itemref idref="ruecksеite" linear="yes"/>

 </spine>

 <guide>

  <reference href="Texte/titelseite.xhtml" type="cover" title="Cover"/>

  <reference href="Texte/impressum.xhtml" type="impressum" title="impressum"/>

  <reference href="Texte/kafka_verwandlung.xhtml" type="inhalt" title="inhalt"/>

  <reference href="Texte/ruecksеite.xhtml" type="autor" title="autor"/>

 </guide>

</package>
```

kafka.opf

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

<!DOCTYPE ncx PUBLIC '-//NISO//DTD ncx 2005-1//EN'
'http://www.daisy.org/z3986/2005/ncx-2005-1.dtd'>

<ncx xmlns="http://www.daisy.org/z3986/2005/ncx/" version="2005-1" xml:lang="de"
dir="ltr">

<head>

        <meta content="KafkaFHBuwi" name="dtb:uid"/>

        <meta content="1" name="dtb:depth"/>

        <meta content="0" name="dtb:totalPageCount"/>

        <meta content="0" name="dtb:maxPageNumber"/>

</head>

<docTitle>

        <text>Franz Kafka</text>

</docTitle>
```

```
<navMap>
        <navPoint id="np-1" playOrder="1">
                <navLabel>
                        <text>Die Verwandlung</text>
                </navLabel>
                <content src="Texte/titelseite.xhtml"></content>
        </navPoint>
        <navPoint id="np-2" playOrder="2">
                <navLabel>
                        <text>Impressum</text>
                </navLabel>
                <content src="Texte/impressum.xhtml"></content>
        </navPoint>
        <navPoint id="np-3" playOrder="3">
                <navLabel>
                        <text>Titel</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e7"></content>
        </navPoint>
        <navPoint id="np-4" playOrder="4">
                <navLabel>
                        <text>Kapitel 1</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e9"></content>
        </navPoint>
        <navPoint id="np-5" playOrder="5">
                <navLabel>
                        <text>Kapitel 2</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e40"></content>
        </navPoint>
```

```
        <navPoint id="np-6" playOrder="6">
                <navLabel>
                        <text>Kapitel 3</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e121"></content>
        </navPoint>
        <navPoint id="np-7" playOrder="7">
                <navLabel>
                        <text>Kapitel 4</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e264"></content>
        </navPoint>
        <navPoint id="np-8" playOrder="8">
                <navLabel>
                        <text>Kapitel 5</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e370"></content>
        </navPoint>
        <navPoint id="np-9" playOrder="9">
                <navLabel>
                        <text>Kapitel 6</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e461"></content>
        </navPoint>
        <navPoint id="np-10" playOrder="10">
                <navLabel>
                        <text>Kapitel 7</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e615"></content>
        </navPoint>
        <navPoint id="np-11" playOrder="11">
```

```
                <navLabel>
                        <text>Kapitel 8</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e709"></content>
        </navPoint>
        <navPoint id="np-12" playOrder="12">
                <navLabel>
                        <text>Kapitel 9</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e807"></content>
        </navPoint>
        <navPoint id="np-13" playOrder="13">
                <navLabel>
                        <text>Kapitel 10</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e928"></content>
        </navPoint>
        <navPoint id="np-14" playOrder="14">
                <navLabel>
                        <text>Kapitel 11</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e1064"></content>
        </navPoint>
        <navPoint id="np-15" playOrder="15">
                <navLabel>
                        <text>Kapitel 12</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e1248"></content>
        </navPoint>
        <navPoint id="np-16" playOrder="16">
                <navLabel>
```

```
                <text>Kapitel 13</text>
            </navLabel>
            <content src="Texte/kafka_verwandlung.xhtml#d13e1414"></content>
        </navPoint>
        <navPoint id="np-17" playOrder="17">
            <navLabel>
                <text>Kapitel 14</text>
            </navLabel>
            <content src="Texte/kafka_verwandlung.xhtml#d13e1561"></content>
        </navPoint>
        <navPoint id="np-18" playOrder="18">
            <navLabel>
                <text>Kapitel 15</text>
            </navLabel>
            <content src="Texte/kafka_verwandlung.xhtml#d13e1762"></content>
        </navPoint>
        <navPoint id="np-19" playOrder="19">
            <navLabel>
                <text>Kapitel 16</text>
            </navLabel>
            <content src="Texte/kafka_verwandlung.xhtml#d13e1866"></content>
        </navPoint>
        <navPoint id="np-20" playOrder="20">
            <navLabel>
                <text>Kapitel 17</text>
            </navLabel>
            <content src="Texte/kafka_verwandlung.xhtml#d13e2003"></content>
        </navPoint>
        <navPoint id="np-21" playOrder="21">
            <navLabel>
                <text>Kapitel 18</text>
```

```
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e2158"></content>
        </navPoint>
        <navPoint id="np-22" playOrder="22">
                <navLabel>
                        <text>Kapitel 19</text>
                </navLabel>
                <content src="Texte/kafka_verwandlung.xhtml#d13e2292"></content>
        </navPoint>
        <navPoint id="np-23" playOrder="23">
                <navLabel>
                        <text>Autor</text>
                </navLabel>
                <content src="Texte/rueckseite.xhtml"></content>
        </navPoint>
</navMap>
</ncx>
```

nav.ncx