METADATEN-MANAGEMENT IN CROSSMEDIALEN PRODUKTIONSUMGEBUNGEN

Semantikinvariante bidirektionale Transformation
typographischer Auszeichnungen auf Basis variabler Compiler

Vom Fachbereich 12 – Informatik und
Elektrotechnik der Universität Siegen
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
(Dr. rer. nat.)

genehmigte Dissertation
von
Dipl. Ing. Marcus von Harlessem

1. Gutachter: Prof. Dr. Wolfgang Merzenich
2. Gutachter: Prof. Dr. Udo Kelter
Vorsitzender: Prof. Dr. Hans Wojtkowiak

Tag der mündlichen Prüfung: 29. Oktober 2009
urn:nbn:de:hbz:467-4180

Gedruckt auf alterungsbeständigem holz- und säurefreiem Papier.

ABSTRACT DEUTSCH

Metadaten-Management in Crossmedialen Produktionsumgebungen

Im Bereich des Cross Media Publishing (CMP) ist die Integration der Prozesse zweier separater Anwendungsbereiche erforderlich: Erstens die Prozesse zur Online-Verarbeitung via Web Content Management Systemen (WCMS) und ähnlichen Anwendungen und zweitens, die Prozesse für die traditionellen gedruckten Publikationen, wie Layout und Formatierung mit Hilfe von Desktop-Publishing-Systemen (DTP). Nachdem diese Welten über viele Jahre isoliert nebeneinander existierten, wurde in den letzten Jahren viel Aufwand in die Übertragung der Inhalte zwischen diesen Medienarten investiert. Ein großer Fortschritt war die Entwicklung von XML als Transport-Protokoll für den Austausch der Daten. XML und die daraus abgeleiteten Sub-Sprachen behandeln mittlerweile viele der sich daraus ergebenden Probleme.

Im Zuge der Medienkonvergenz erfahren diese Integrationsprozesse eine schnell steigende Bedeutung. Medienunternehmen streben eine im Rahmen ihrer Diversifikationsstrategien notwendige Mehrfachverwertung von aufwändig erstellten Inhalten mit möglichst geringen Kosten an.

Ziel dieser Arbeit war zunächst, die neu entstehenden Prozesse in der medienneutralen Informationsverarbeitung zu erkennen, ihre Risiken und Kosten zu identifizieren und zu formalisieren. Während die Prozesse für die medienneutrale Verarbeitung von Bilddaten theoretisch bereits gelöst sind, gab es für die Behandlung von typographischen Metadaten bisher keine befriedigende Lösung, wenn man die industriellen Standards zur Bearbeitung von Inhalten beibehalten und den Anwendern weiterhin das Arbeiten in ihren gewohnten Umgebungen ermöglichen will.

Jede einzelne, medienspezifische Anwendung zur Bearbeitung und Gestaltung von Textinhalten besitzt ihre eigene Sprache mit typographischer Syntax und Semantik. Um diese typographischen Metadaten bei der Übertragung zwischen medienspezifischen Anwendungen und bei der mehrfachen Verwendung von Texten innerhalb einer Mediengattung nicht zu verlieren, wurde im Rahmen dieser Arbeit ein Verfahren zur Übersetzung dieser typographischen Metadaten auf Basis von grammatikgesteuerten Übersetzern entwickelt. Die Übersetzer verwenden als Drehscheibe für den Datenaustausch die Typographische Markup Sprache TML, ein XML-basierter Dialekt zur medienneutralen Datenhaltung der typographischen Metadaten. Die TML wurde ebenfalls im Rahmen dieser Arbeit entwickelt.

Die Anforderungen an die Gesamtarchitektur wurden für diese Arbeit in Zusammenarbeit mit der InterRed GmbH formuliert. InterRed entwickelt und vertreibt ein Redaktions- und Publishingsystem für die medienübergreifende Publikation von Inhalten wie Texte, Bilder und Video. Für die bidirektionale Übertragung von Textinhalten zwischen dem Redaktionssystem und Standards wie Quark XPress, Adobe Indesign, RTF oder HTML gab es bisher kein Verfahren, welches die typographische Semantik eines Textes berücksichtigt.

Marcus von Harlessem, Freudenberg, im Dezember 2008

ABSTRACT ENGLISH

Marcus von Harlessem:

In the field of Cross Media Publishing (CMP) the processes of two separate application suits require integration: first, the online information processing via Web Content Management Systems (WCMS) and similar applications, and, second, the traditional printed publications, such as the layout and formatting with Desktop-Publishing Systems (DTP). After many years of respective isolation, much effort has been dedicated to enable data transfers between these realms. One major achievement was XML as a data transport protocol. XML, and the sub-languages derived from it, addressed many of the resultant problems.

In the wake of media convergence, these integration processes get a fast-growing importance. In the context of their diversification strategies, media enterprises require the multiple utilization of their expensive content with the lowest possible costs.

The aim of this work was, first, to identify and formalize the risks and the costs of the new processes, that would occur in a media-neutral processing environment information. While the theory of the processes of media-neutral image processing has already been solved, so far there were no satisfactory solutions for the treatment of typographical metadata, if the industry standards for handling content are maintained and users would be allowed to work in their familiar application frameworks.

Every single media-specific application for editing and layout of textual content has its own internal language, with typographic syntax and semantics. In order not to lose this typographic metadata while transferring content between the media-specific applications or at the multiple use of texts within a media class, it was in the scope of this work to develop a translation-procedure for typographical metadata, based on translators controlled by grammar rules. For the exchange of textual data, the compilers uses the Typographic Markup Language (TML), an XML-based dialect of the media-neutral contents representation, as an intermediate hub format of the typographical metadata. Developing the TML was also an aspect of this work.

For this work, the demands for the overall architecture have been predefined in collaboration with the InterRed GmbH. InterRed is a software manufacturer, that develops and distributes an editing and publishing application for the media independent publishing of content such as text, images and video. For the bi-directional exchange of text content between the editorial system and industry standards such as QuarkXPress, Adobe InDesign, RTF or HTML there was no method, which takes the semantics of a typographical text into account.

Marcus von Harlessem, Freudenberg, at December 2008

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

DANKSAGUNG

Als ich gegen Ende des Jahres 2002 mit den ersten Recherchen zum Thema Cross Media Publishing begann, sah ich vor mir einen Zeitraum von 4-5 Jahren, in denen ich intensiv in die Thematik einsteigen und einen gewissen Beitrag zur deren Weiterentwicklung leisten würde. Es sollte eine Aufgabe sein, deren Umfang so gewählt würde, dass sie neben den weiteren Aufgaben als wissenschaftlicher Mitarbeiter an der Universität Siegen in diesem Zeitraum zu schaffen sein würde, vor allem – das ich sie alleine schaffen würde und müsste.

Jetzt, ziemlich genau sechs Jahre später, fällt die Retrospektive deutlich anders aus. Nur durch die Hilfe und Mitarbeit vieler Menschen konnte das Gesamtsystem eine anwendbare, praxistaugliche und effektive Stufe erreichen, konnte der Kontext der Arbeit in vielen Facetten beleuchtet werden.

An erster Stelle möchte ich drei Menschen meinen ganz besonderen Dank aussprechen: Meinem Doktorvater Prof. Dr. Wolfgang Merzenich danke ich für die unschätzbar wertvolle Möglichkeit, eigenverantwortlich und in freiem Geiste in der Fachgruppe Programmiersprachen der Universität Siegen geforscht und gelehrt haben zu können. In Zeiten einer beschleunigt fortschreitenden Ökonomisierung der Hochschulen ist das leider keine Selbstverständlichkeit mehr. Seine allzeit positive und Neuem gegenüber aufgeschlossene Art steckt nicht nur an, sie macht einfach Spaß. Er nahm sich für Gespräche und Diskussionen immer ausreichend Zeit und war dabei ein aktiver Ideengeber und Motivator. Einen solchen Chef wünsche ich jedem Jung-Wissenschaftler.

Großer Dank gebührt auch Dr. André Klahold, Geschäftsführer der InterRed GmbH und langjähriger Begleiter meiner beruflichen Karriere. In den vielen Jahren unserer Zusammenarbeit hat er sich immer als außergewöhnliche Kombination aus fachlicher und sozialer Kompetenz gezeigt. Das heute oft vermisste Paradigma des Unternehmens als große Familie wird durch ihn in vorbildlicher Weise gelebt. Er hat die Entwicklung der TML-Engine strategisch und operativ begleitet und mit guten Ideen angereichert.

Meiner lieben Frau Monika bin ich ebenfalls zu großem Dank verpflichtet. Sie war mir in der langen Zeit Stütze und Ruhepol zugleich, hat mich motiviert und unterstützt, wie ich es mir besser nicht vorstellen könnte und viele durchgearbeitete Wochenenden klaglos erduldet. Ohne sie würde es diese Arbeit nicht geben.

Dem zweiten Gutachter Hr. Prof. Dr. Udo Kelter danke ich für die investierte Zeit, die er trotz der umfangreichen Aufgaben an der Universität Siegen, für Gespräche und für die Begutachtung aufgebracht hat.

Des Weiteren danke ich allen Studenten, die zum Gelingen dieser Arbeit beigetragen haben, stellvertretend und besonders den Mitgliedern der Projektgruppe Transformation, welche den größten Teil der TML-Engine implementiert haben: Daniel Hüsch, Marc Nilius, Tobias Reichling, Magnus Roth, Adrian Schütz und Pascal Zeuner.

Für die allzeit gewährte Hilfe und Unterstützung und für ein wunderbares und unendlich unterhaltsames Arbeitsklima danke ich meinen ehemaligen Kollegen der Fachgruppe Pro-

grammiersprachen Birgit Berger-Bedarff, Simon Budig, Dr. Achim Hennings und Christoph Schlechtingen. Meinem Freund und Sportskamerad Dirk Vetter danke ich für das Lektorat.

ABKÜRZUNGSVERZEICHNIS

CALS:	Computer-Aided Acquisition and Life-Cycle Support
CML:	Chemical Markup Language
CMP:	Cross Media Publishing
CMS:	Content Management System
COST:	Copenhagen SGML Tool
CPAN:	Comprehensive Perl Archive Network
CRM:	Customer Relationship Management
CSS:	Cascading Style Sheets
CTP:	Computer-to-Plate
CvD:	Chef vom Dienst
DMS:	Dokumentenmanagement System
DSSSL:	Document Style Semantics and Specification Language
DTD:	Document Type Definition
ECMS:	Enterprise Content Management System
ERP:	Enterprise Ressource Planning
GXL:	Graph Exchange Language
KMU:	Klein- und mittelständische Unternehmen
NITF:	News Industry Text Format
MAM:	Media Asset Management
OASIS:	Organization for the Advancement of Structured Information Standards
PIM:	Product Information Management
RSS: (ab 2.0)	Really Simple Syndication

SGML:	Standard Generalized Markup Language
TCL:	Tool Command Language
WCMS:	Web Content Management System
XMP:	eXtensible Metadata Platform
XPS:	XML Paper Specification
XSL:	eXtensible Stylesheet Language
XSL-FO:	eXtensible Stylesheet Language Formatting Objects
XTG:	Quark Xpress TaG
VDZ:	Verband Deutscher Zeitschriftenverleger

EINLEITUNG

Im Vergleich zur historischen Entwicklung des Buchdrucks[1] erscheint die Ägide des Information Broadcasting and Publishing über digital produzierte Online- und Offline-Medien als Entwicklungsprozess, der in atemberaubender Geschwindigkeit die weltweit verfügbaren Prozesse verändert. Wir erleben eine rasante Bedeutungszunahme des World Wide Web für die Bereitstellung und Verteilung jeglicher Informationen. Zahlreiche Studien belegen den gleichzeitigen Bedeutungsschwund der klassischen Printmedien. Vor allem Produkte im B2C-Bereich mit Nachrichten und Informationen mit geringer Halbwertszeit mussten schon in der Vergangenheit eine Verschiebung der Nutzeranteile in Richtung der Online-Medien feststellen. Mittlerweile trifft diese Abwanderung auch fachkompetente Medienunternehmen im B2B-Bereich. Neben Migrationstendenzen der Bestandskunden ist vor allem das signifikant geänderte Mediennutzungsverhalten der jüngeren Generationen eine Herausforderung für die klassischen Medienunternehmen.

Die *Intermediäre Diversifikation*[2] in die Zielbranche Online-Medien ist in fast allen Unternehmen der Inhalte-Industrie[3] bereits erfolgt, mit den unterschiedlichsten Strategien, Geschäftsmodellen und technischen Lösungen. Die betriebswirtschaftlichen Erkenntnisse sind wie so oft ein wichtiger Motivator dieses ‚Zusammenwachsens'.

Nachdem sich nun dafür notwendige effektive Werkzeuge und Prozesse etabliert haben, folgt im nächsten Schritt die Frage nach der Effizienz. Das Ziel ist die Verschmelzung von Online- und Offline auf allen Ebenen der vertikalen Prozesskette. Eine grundlegende Forderung ist dabei die Abschaffung der redundanten Datenhaltung und damit die Publikation aus einer zentralen Datenquelle, allgemein als *Cross Media Publishing* bezeichnet.

Cross Media Publishing, die medienübergreifende elektronische Publikation, hat sich unter dem Aspekt der zweigleisigen Produktion von Online- und Offline-Medien innerhalb weniger Jahre zu einem strategischen Faktor für die Medienindustrie entwickelt. Die parallele, autonome Produktion in zwei Welten ist aus kostenmäßigen und qualitativen Gründen nicht mehr sinnvoll. Im Rahmen dieser Forschungsarbeit kooperierte die Universität Siegen mit der InterRed GmbH, einem traditionellen Hersteller von Content Management Software. Seit einigen Jahren entwickelt man dort Lösungen für das Cross Media Publishing im industriellen Umfeld. Die grundlegende Analyse der informationstechnischen Anforderungen mit dem Schwerpunkt auf der Realisierung von Medienbrüchen ohne den Verlust typographischer Metadaten waren die zentralen Ziele dieser Kooperation.

Der erste Teil dieser Arbeit betrachtet den Kontext dieses Gebietes und endet mit der Motivation für diese Forschungsarbeit. Auf informationstechnischer Ebene sind hier zwei Bereiche relevant, deren gewünschter Zusammenschluss den Auslöser dieser Arbeit bildeten: Die

---

1 siehe Bleisatz

auf Seite 6

2 siehe Seite 30

3 Verlage, Radio- und TV-Stationen

vergleichsweise junge Welt der Informationsbereitstellung in Online-Medien, ein Thema, welches erst mit der Erfindung des Internets geboren wurde. Und die eher traditionelle Welt des IT-gestützten Buch- und Zeitungsdrucks, des Electronic Publishings. Über die Jahre haben sich in beiden Welten Verfahren, Prozesse und Standards für Daten und Anwendungen entwickelt, deren Integration seit ungefähr der Jahrtausendwende Gegenstand der wissenschaftlichen Forschung sind und die als wesentliche Rahmenfaktoren in dieser Arbeit zu beachten waren.

Hinreichende Voraussetzung dafür ist die möglichst weit reichend medienneutrale Speicherung der zu integrierenden Daten und Metadaten in strukturierter Form. Ein wichtiger Schritt für diese Aufgabe war die ‚Erfindung' der Extensible Markup Language – XML. Seit dem 6. Oktober 2000 ist XML 1.0 vom W3C spezifiziert[4]. Auf XML basieren heute die meisten integrativen Verfahren. Letztendlich ist XML aber nur eine Technologie, die Informationen mit Typ und Struktur ausstattet.[5] Auf eine allgemeine Einführung in SGML und XML wird in dieser Arbeit verzichtet, die Grundlagen können als bekannt vorausgesetzt werden. Spezifische XML-Derivate, die der Transformation von Dokumenten und Texten zwischen verschiedenen Formaten und Medien dienen und die im Kontext der Arbeit stehen, werden in Teil III vorgestellt.

Für die Transformation oder den Austausch von Daten existieren bereits einige XML-basierte Verfahren, beispielsweise UXF für UML-Dokumente oder GXL für Graphen. Auch für die medien-unabhängige strukturierte Ablage technischer Dokumentationen existieren Standards wie DocBook, DITA, CALS, um die Metadaten kümmern sich Verfahren wie RDF, Dublin Core usw.

Auch für die zunehmende Vielfalt an Ausgabegeräten und Medienformen existieren Transformationstechniken, welche eine formatierte und mediengerechte Publikation strukturierter Daten ermöglichen. Der Verbreitungsgrad in der industriellen Anwendung ist bislang aber eher gering. Hauptsächlich sprechen zwei Gründe dagegen:

Verfahren wie XSL-FO verfolgen einen ganzheitlichen Ansatz und liefern fertig druckbare Dokumente. In den Medienunternehmen existieren allerdings über lange Zeit etablierte industrielle Standardsysteme, die insbesondere für Print-Objekte und Internetseiten umfangreiche, medienspezifische Funktionen beinhalten. Distribution und Publishing muss weiterhin aus diesen Systemen erfolgen. Ziel ist also der Austausch und die Transformation zwischen diesen Systemen ohne Verlust der teuer erstellten Metadaten.

Betrachtet man alle Zielobjekte in einem Cross-Media-Projekt als Knoten in einem Graphen und definiert man die zu programmierende Transformation zwischen zwei Objekten als Kante zwischen diesen Knoten, so ergibt sich im schlechtesten Fall ein vollständiger Graph. Die Anzahl der zu programmierenden Transformationen hängt also quadratisch von der Anzahl der Objekte ab. In Verlagen mit einer größeren Zahl von Produkten und Zielmedien bedeutet das einen enorm hohen Entwicklungsaufwand.

---

[4] siehe [W3C00]

[5] siehe[Box01 S. 15]

Die Behandlung der Metadaten unter dem Aspekt der Medienneutralität ist ein relativ neues Forschungsgebiet. Dabei ist zunächst eine Differenzierung notwendig: Welche Metadaten sind medienneutral und welche sind medienspezifisch? Der Begriff „Cross Media Publishing" wird bisher nur sehr ungenau verwendet.

Teil II dieser Arbeit analysiert Prozesse und Risiken im Cross Media Publishing und formalisiert die zu erwartenden Kostenarten. Anschließend werden mögliche Prozessarchitekturen zur medienübergreifenden Medienproduktion in drei Klassen aufgeteilt.

Die Ergebnisse dieser Forschungsarbeiten und Analysen bilden die Rahmenbedingungen und Anforderungen für die technische Konzeption und Implementierung der Architektur aus sprachspezifischen typographischen Übersetzern, der Datenhaltung der variablen Grammatiken, der typographischen Markup-Sprache TML und der umgebenden Gesamtarchitektur. Teil IV beschreibt genauer die Konzeption und Implementation dieser Bestandteile.

In Kapitel 14 folgt eine Interpretation der formalen Eigenschaften der TML und ihre Auswirkungen auf die Entwicklung der Übersetzer. Die Besonderheiten der externen Formate RTF und QuarkXPress und ihr Einfluss auf die Implementation werden vorgestellt, ebenso wie besondere semantische Eigenschaften der entwickelten Architektur. Weiter wird in Kapitel 15 erläutert, in wie weit die in Teil II identifizierten Problemstellungen mit Hilfe der TML-Architektur behandelt und Kosten reduziert werden konnten. Teil V schließt mit einem Bericht über erste Erfahrungen, die im praktischen Betrieb gesammelt werden konnten, einem Fazit zu dieser Forschungsarbeit und gibt einen Ausblick auf die weitere Entwicklung dieses sehr dynamischen Themas.

Teil I

Kontext

Themenumfeld und Motivation

K a p i t e l 1

ELECTRONIC PUBLISHING

Printing should be invisible. (Beatrice Warde)

Vom Blei zum Byte - Exkurs in die Historie des „klassischen Publishing“

Als Johannes Gensfleisch, später bekannt als Gutenberg[6] (geboren um 1400 in Mainz, gestorben 3. Februar 1468 ebenfalls in Mainz), aus verschiedenen, ihm bekannten Techniken – Glockengießen, Weinpresse, Holzschnittdruck – den „modernen“ Buchdruck erfand, ahnte er wohl noch nicht, welch revolutionierende Erfindung ihm gelungen war. In der bekannten Welt wurden damals Bücher durch Abschreiben vervielfältigt, und das vorwiegend durch Mönche, die einzige Bevölkerungsgruppe mit durchgängiger Schreibausbildung. Diese Art des Veröffentlichens[7] war mühsam und fehleranfällig, noch dazu blieben die dadurch erreichten Auflagen in sehr kleinem Rahmen, was eine geringe Verbreitung von Büchern zur Folge hatte.

ABBILDUNG I-1: LANGE ZEIT ARBEITETEN SCHRIFTSETZER NOCH ENG ANGELEHNT AN DIE ERFINDUNG DES BUCHDRUCKS DURCH GUTENBERG. (ARCHIVFOTO: SIEGENER ZEITUNG)

[6] Benannt nach dem Namen des Familienbesitzes, dem Hof der Eltern „zum Gutenberg“.

[7] Im weiteren Verlauf bezeichnen wir alle Formen von Veröffentlichung als „Publishing“.

Gutenbergs Errungenschaften änderten dies. Er kam als erster[8] auf die Idee, aus den bisher zum Druck benutzen Holzschnitten die einzelnen Buchstaben zu isolieren, sie separat als Lettern aus einer Blei-Zinn-Legierung zu gießen und die Druckplatten aus den einzelnen Buchstaben zusammenzusetzen (siehe folgender Abschnitt). Zusammen mit einer Handpresse und von ihm hergestellten Farben entstanden so die ersten Druckwerke, die sich einfach und für die damalige Zeit schnell vervielfältigen ließen.

Im Folgenden werden einige, in der Industrie wichtige Verfahren[9] zum Erstellen von Drucksachen (Büchern, Zeitungen, Zeitschriften, Broschüren, Flyer, Briefe, etc.) vorgestellt. Es werden dabei hauptsächlich Techniken aus der Druckvorstufe beschrieben. So bezeichnet man den Prozessabschnitt, der vom Erstellen der druckfertigen Vorlagen bis zur Druckplattenherstellung reicht. Neben dem geschichtlichen Aspekt und der grundsätzlichen Technik werden dabei auch die jeweiligen Vor- und Nachteile herausgestellt.

ABBILDUNG I-2: DER SETZKASTEN WAR EIN WICHTIGES HANDWERKSZEUG DES SETZERS. (ARCHIVFOTO: SIEGENER ZEITUNG)

Bleisatz

Der Bleisatz beschreibt die klassische Drucktechnik, bei der Druckplatten aus einzelnen Lettern (den Buchstaben und Zeichen, manchmal auch Zeichengruppen) zusammengesetzt werden. Die Lettern bestehen aus einer Metalllegierung, die zum größten Teil aus Blei be-

[8] Es existieren überlieferte Berichte, dass in China und Korea buddhistische Mönche bereits 800 Jahre vor Gutenberg ähnliche Techniken anwendeten. Allerdings haben keine ihrer Erzeugnisse die Zeiten überdauert, so dass der endgültige Beweis dieser Berichte wohl nicht erbracht werden kann. In Ostasien wurden einzeln geschnittene Lettern aus Metall in Korea bereits um 1232 entwickelt. Der Druck mit beweglichen Lettern lässt sich im Kaiserreich China sogar bereits im 11. Jahrhundert als Erfindung von Bi Sheng nachweisen; seine Arbeitsmethoden wurden von Shen Kuo in den Meng Xi Bi Tang („Traumstrom-Essays“) aufgezeichnet.

[9] Auswahl in enger Anlehnung an [Fri01]. Es gibt noch einige weitere Verfahren, die aber aus Platzmangel hier nicht behandelt werden sollen.

steht. Sie sind an der Basis quadratisch, alle gleich hoch, und am wesentlichen Ende ist das zu druckende Zeichen spiegelbildlich herausgearbeitet. Der Drucker, ein ausgebildeter Meister seines Handwerks, setzt – daher der Begriff „Bleisatz“ – die Lettern Buchstaben für Buchstaben, Zeile für Zeile, auf die Druckplatte (auch Druckstock genannt), welche später in der Druckmaschine mit Farbe bestrichen wird und diese zu Papier bringt.

Wesentlich bei dieser Drucktechnologie, und das ist vor allen Dingen die Errungenschaft von Gutenberg, ist die Idee der Wiederverwendbarkeit der Lettern – hat man einmal eine Seite fertig gedruckt (und ist sich sicher, dass man sie nicht mehr nachdrucken muss), kann man die Druckplatte wieder auseinander nehmen und eine neue setzen. Nur eine begrenzte Menge an Lettern ist daher für die Produktion auch umfangreicher Druckwerke nötig.

ABBILDUNG I-3: DIE ZEILENGUSSMASCHINEN VON LINOTYPE ERLEICHTERTEN SPÄTER DIE ARBEIT DES SETZERS UND ERHÖHTEN DIE SETZLEISTUNG AUF 5.000 – 6.000 BUCHSTABEN PRO STUNDE. (ARCHIVFOTO: SIEGENER ZEITUNG)

Bis in die 80er Jahre des 19. Jahrhunderts war das Setzen der Druckplatten reine Handarbeit und entsprechend mühsam. Gute Setzer schafften im Handsatz bis zu 1500 Zeichen pro Stunde. [Gen78]

Das verbesserte sich radikal, als 1886 Ottmar Mergenthaler mit der „Linotype“ die erste Satzmaschine der Welt erfindet. Mit ihr werden aus einzelnen Lettern, welche der Setzer über eine Tastatur anforderte, die Zeilen zusammengestellt, die danach in Metall gegossen als ganze Zeile („line o’ type“) in den Satz kommen. Auf diese Weise lassen sich 5000 bis 6000 Buchstaben pro Stunde setzen. [Gen78]

Vorteile:

- Gute, gleich bleibende Qualität
- Grundlage des Handwerks

Nachteile:

- Hoher Aufwand bei Objektwechsel
- Langsam (allerdings verbessert durch Maschineneinsatz)
- Teuer
- Unflexibel

[Fri01], [Duß73], [Gen78]

Fotosatz

Um 1970 erlangt eine neue Technik in der Druckbranche große Bedeutung und löste aufgrund freierer Gestaltungsmöglichkeiten und höherer Produktivität den Handsatz vollständig ab: der Fotosatz wird eingeführt.

ABBILDUNG I-4: ANFANG DER 80ER JAHRE WURDE DIE BLEI-ÄRA BEI DER SIEGENER-ZEITUNG DURCH DEN DAMALS REVOLUTIONÄREN FOTODRUCK ABGELÖST. (ARCHIV-FOTO: SIEGENER ZEITUNG)

Seiten werden dazu vom Layouter auf lichtempfindlichem Film belichtet und danach zur Druckplattenerzeugung gegeben. Dies geschieht ebenfalls auf fototechnischem Weg: Die Druckplatten aus Metall, ebenfalls mit lichtempfindlichen Material beschichtet, werden durch die Filmvorlage hindurch belichtet und entwickelt. Je nach Plattentyp und Entwicklungsverfahren entstehen Druckplatten für Hoch-, Tief- oder Flachdruck. Sie unterscheiden sich durch die Ausprägung der zu druckenden Fläche – beim Hochdruck ist diese aus der Platte herausgearbeitet, beim Tiefdruck hingegen eingefräst. Beim Flachdruck schließlich gibt es kaum einen Höhenunterschied zwischen der zu druckenden und der nicht zu druckenden Fläche. Hier greifen später chemische Effekte, um die Farbe korrekt aufzubringen (siehe folgendes Kapitel Offsetdruck). [Fri01], [Duß73]

Offsetdruck

Der klassische Offsetdruck ist ein indirektes Druckverfahren, die Farbe wird also nicht direkt auf das zu bedruckende Material aufgetragen, sondern zuerst auf ein Übergangsmedium, bevor sie auf das Zielmedium gedruckt wird. Im Falle des Offsetdrucks sind dies vier Gummiwalzen, auf welche die vier Farben des CMYK-Farbmodells[10] aufgetragen werden. Das passiert mit Hilfe separater Druckplatten, die jeweils eine Farbe aufnehmen und das Druckbild aufgrund unterschiedlicher Benetzungsverhalten verschiedener Stoffe definieren. Eine Druckplatte, entwickelt aus einem belichteten Film, hat dabei zwei unterschiedliche Oberflächeneigenschaften: Eine, welche die Farbe anzieht, aber Wasser abstößt und der umgekehrte Fall. Beim Anfahren der Druckmaschine wird zuerst „befeuchtet", das heißt Wasser auf die Druckplatten gegeben, welches die wasserfreundliche Schicht vollständig aufnimmt. Danach wird die Farbe aufgetragen, die sich ausschließlich auf der farbfreundlichen Schicht ansammelt. Von dort wird sie auf die Gummiwalze übertragen. In der Vierfarb-Offsetdruckmaschine sind die vier Druckwerke hintereinander geschaltet und bringen nacheinander je eine Farbe auf das Medium auf. Man kann sich gut vorstellen, dass das mit einer hohen Präzision erfolgen muss, um ein exaktes Druckbild zu erhalten.

Dieses Druckverfahren, so kompliziert es auch klingt, hat lange Zeit die Drucktechnik beherrscht und wird noch heute von vielen Druckern als die einzig wahre Druckmethode bezeichnet. Zusätzlich ist es gegenwärtig für hochvoluminöse Produktionen noch ohne veritable Konkurrenz, da die Alternativen (siehe weiter unten) für große Mengen entweder zu teuer sind oder aber nicht die geforderte Qualität erreichen.

[10] CMYK: Cyan, Magenta, Yellow (Gelb), Key (Kontrast, Schwarzwert) ist das für den Papierdruck gebräuchliche, subtraktive Farbmodell. Es beruht auf der Tatsache, dass die Mischung aller drei Grundfarben schwarz ergibt. Der Schwarzwert wurde in den Farbraum aufgenommen, um Farben abzudunkeln und um den Druckprozess zu verbessern (in der Praxis wird 100% Schwarzwert benutzt, um wirkliches Schwarz zu erreichen).

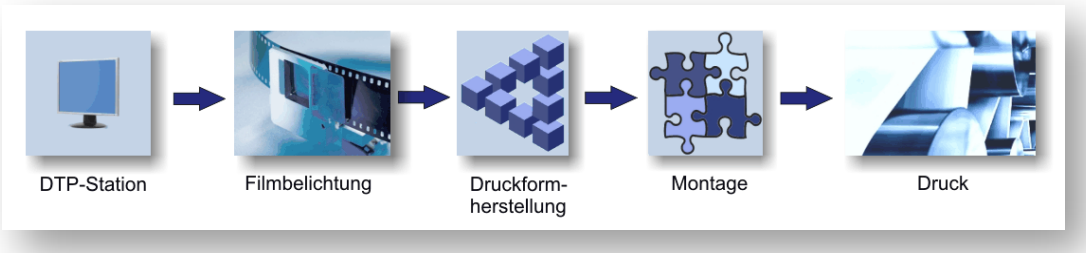

ABBILDUNG I-5: VEREINFACHTER OFFSETDRUCK-WORKFLOW

Vorteile:

- Hohe Geschwindigkeit
- Beste, gleich bleibende Qualität

Nachteile:

- Hoher bis moderater Aufwand bei Objektwechsel
- Teuer
- Unflexibel

[Fri01], [Pet98]

Desktop Publishing (DTP)

Mitte der 80er Jahre des letzten Jahrhunderts tritt unter der Bezeichnung „Desktop Publishing“ nicht ein neues Druckverfahren, sondern eine Änderung in der gewohnten Prozesskette der Druckindustrie auf den Plan. Wo bisher teuer bezahlte Spezialisten als Dienstleister die Texte, Bilder und andere Daten der Autoren zu Druckobjekten montieren und in aufwändigen Druckverfahren erzeugen mussten, nehmen jetzt die Autoren die Dinge selbst in die Hand. Möglich machen dies einige Hersteller aus der Computerindustrie, die es verstehen, die wichtigsten Zutaten des „Publishings“ vergleichsweise günstig für den Personal Computer zu realisieren:

- Das Startup-Unternehmen Adobe erfindet mit „PostScript“ die erste Druckbeschreibungssprache, die es ermöglicht, ein Dokument in einem skalierbaren Format zu speichern und unabhängig vom Ausgabegerät verlustfrei zu drucken.

- Nach den Prototypen aus dem Xerox Parc Research Center, dem Xerox Star und Apples Lisa, bringt Apple 1984 den ersten bezahlbaren Computer mit vollgrafischer Bedienoberfläche.
- Ebenfalls von Apple kommt der erste PostScript-fähige Laserdrucker, der ein annehmbares Druckbild dessen erzeugen konnte, was der Anwender auf seinem Bildschirm sieht.
- Die Softwarefirma Aldus steuert den „PageMaker“ bei, das erste DTP-Programm, das es dem Anwender ermöglicht, die Seiten seines Druckobjektes mit Texten und Bildern zu füllen und zu formatieren.

Von Linotype, einer Firma aus dem traditionellen Druckhandwerk, kommen die ersten PostScript-Schriften und der erste PostScript-Belichter für Offset-Druckplatten (siehe Offsetdruck S. 9).

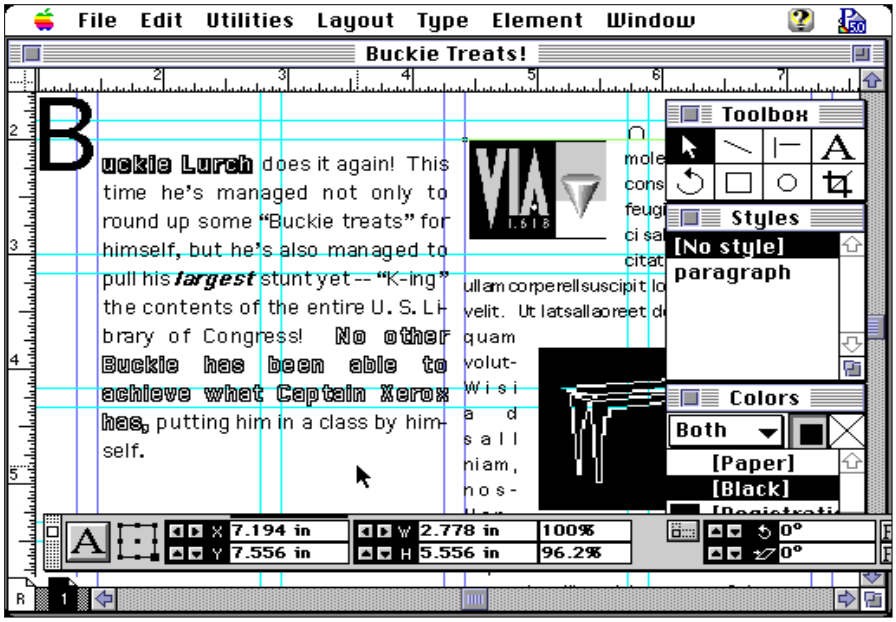

ABBILDUNG I-6: ALDUS PAGEMAKER AUF APPLE MACINTOSH

Die Kombination dieser Ereignisse löst eine Revolution in der Druckindustrie aus. Nachdem die Anwender an ihrem Apple-Computer mithilfe von PageMaker die Seiten selbst „setzen“ und Kleinserien davon in guter Schwarzweiß-Qualität mittels PostScript auf dem Laserwriter drucken können, bricht den Setzern und Druckereien ein Einnahmezweig weg. Schnell erhält das Desktop Publishing allerdings dadurch, dass es so einfach durchgeführt werden kann, das Stigma der Minderwertigkeit. Denn qualitativ hochwertiges Erzeugen von Zeitschriften, Büchern und Broschüren, ist im Wesentlichen nicht nur eine Frage der Machbarkeit, sondern basiert hauptsächlich auf Wissen und Erfahrung. Gelernte Setzer, Drucker und Mediengestalter haben eine Ausbildung hinter sich, unter Umständen viel Erfahrung in ihrem Job und ein Verständnis von Gestaltung und Design. Was der nun DTP-befähigte Autor in der Regel nicht hat.

Die Ergebnisse sind also unter Umständen wenig professionell, haben aber den großen Vorteil der deutlich geringeren Kosten. Das Drucken von einfarbigen Kleinserien ist preiswert wie nie, der Siegeszug von DTP in der Medienwelt unaufhaltsam. Was früher Setzer an ihren Satzmaschinen und Montagetischen erledigten, übernehmen jetzt „Layouter" an Computern. Diese Prozessmigration vernichtet auf der einen Seite Arbeitsplätze in der herkömmlichen Druckindustrie oder bringt sogar für einige Firmen den Ruin beziehungsweise die Übernahme durch die Fortschritts-orientierte Konkurrenz, auf der anderen Seite entstehen neue Jobs in den Zweigen der Branche, die sich nun intensiv mit DTP auseinander setzen. Neue PostScript-fähige Geräte werden gebaut, bessere Anwendungsprogramme geschrieben, Ausbildungsberufe werden angepasst. Betriebe stellen Schritt für Schritt ihre Technik und Dienstleistungen um. Heute ist DTP aus der Druckindustrie nicht mehr wegzudenken, ein fester Bestandteil der Prozesskette, an deren Ende das fertige Druckerzeugnis steht.

Vorteile:

- „What You See Is What You Get": so wie man es am Bildschirm gestaltet, wird es gedruckt
- Laserdrucker erlauben Anwendern und Firmen Vorabdrucke zur Qualitätssicherung („proof") und den preiswerten Druck von Kleinserien
- Große layout-technische Flexibilität auch von drucktechnischen Laien umsetzbar
- Beginn der Digitalisierung in der Druckindustrie

Nachteile:

- Große Flexibilität in Korrelation mit fehlendem Know-How führt potenziell zu qualitativ minderwertigen Ergebnissen

[Fri01], [Pet98], [Bau90], [Kre88], [Bae87]

Computer-to-Plate

Im Laufe der 90er Jahre des 20sten Jahrhunderts entsteht durch die Diversifikationsstrategien der Verlage ein steigender Bedarf an Kleinserien. Bekannte Verfahren sind dafür zu teuer. Um die Kunden zu binden, muss eine andere Lösung gefunden werden. Der Erfolg des DTP (siehe Desktop Publishing (DTP) S. 10) bringt ein Verfahren zurück in die Köpfe der Drucker, welches zwar schon 1968 erprobt und 1974 erstmals produktiv eingesetzt wurde, sich aber bisher wegen Lücken in der Prozesskette nicht durchsetzen konnte. „Computer-to-Plate" (CtP) ist eine Weiterentwicklung des Offsetdrucks. Bei diesem Verfahren wird die kostenintensive Belichtung des Films und die nachgeschaltete Übertragung des Druckbildes auf die Druckplatte zu einem Schritt zusammengefasst: Der Film fällt weg, die Druckplatte wird direkt belichtet. Dazu kommt im Plattenbelichter in der Regel ein Laser zum Einsatz,

welcher die Oberfläche behandelt und so das Bild herausarbeitet. Danach wird die Platte in den Offsetdruck übergeben und in der Druckmaschine montiert (siehe Offsetdruck S. 9). Während des Druckprozesses kontrollieren Mitarbeiter in den computergestützten Leitständen der Druckmaschinen in regelmäßigen Intervallen die genaue Ausrichtung der Druckplatten.

CtP ist gegenwärtig immer noch das vorherrschende Verfahren für kostengünstige Produktion von Großserien. Es bietet die Vorteile des Offsetdrucks (hohe Qualität und Geschwindigkeit) mit geringeren Kosten und schnellerer Adaptierbarkeit der Druckjobs. Neben Einsparungen in der Filmentwicklung – dieser Prozess kostet nicht nur Zeit und Geld, sondern auch die Lagerhaltung der dafür benötigten Materialien – ist es vorrangig die beschleunigte Abwicklung, welche die Verbreitung von CtP fördert. Die aus dem DTP kommenden digitalen Daten sind schnell und kurzfristig auf die Druckplatten aufgebracht, so dass man flexibel den Druckzeitpunkt bestimmen kann. Dadurch lassen sich teure Offsetdruckmaschinen besser auslasten. Ein weiterer Vorteil liegt in der Qualitätssteigerung, die durch den Wegfall des Films entsteht: Beim Belichten der Druckplatten mittels Film muss der Bearbeiter penibel auf Sauberkeit achten, damit auf der Druckplatte keine Verunreinigungen entstehen. Durch die direkte Übertragung der digitalen Daten auf die Druckplatte wird dieses Problem beseitigt.

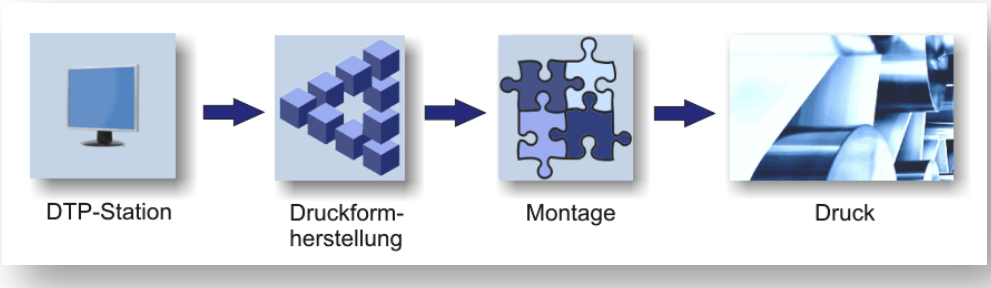

ABBILDUNG I-7: VEREINFACHTER CTP-WORKFLOW

Vorteile:

- Geringer Aufwand bei Objektwechsel
- Flexiblere und kostengünstigere Druckplattenherstellung
- Durch Beibehaltung des Offsetdrucks weiterhin hohe und gleichmäßige Druckqualität
- Beginn der Einbindung der Druckereien in den digitalen Workflow

Nachteile:

- Durch Beibehaltung des Offsetdrucks weiterhin nur bei Großserien lohnenswert

[Fri01], [Pet98], [Mal04]

Digitaldruck

Aktuell ist der Digitaldruck die modernste Technologie im Druckhandwerk. Das zu druckende Dokument wird in elektronischer Form direkt vom DTP-Arbeitsplatz beziehungsweise -Server an die Digitaldruckmaschine übertragen. Gedruckt wird mit Laser-, in größeren Maschinen auch mit Thermotransfer- oder Tintenstrahltechnik. Der Digitaldruck ist die erste Drucktechnologie mit professioneller Ergebnisqualität, die KMU direkt im eigenen Betrieb einsetzen können. Je nach Investitionsvolumen können Unternehmen in Qualität und Leistung Geräte erwerben, die Drucke erzeugen, welche vorher nur mit hohen Initialkosten in einer Druckerei machbar waren, was bei kleinen Auflagenzahlen zu sehr hohen Stückkosten führt. Beste Qualität und Druckprodukte im Premiumsegment mit modernen Verfahren wie Kaltfolientransfer[11], Heißfolienprägung[12] und Spotlackierung gibt es allerdings bis auf weiteres nur aus der Druckerei.

Der Digitaldruck ist aktuell die ideale Art Klein- und Kleinstserien zu drucken. Ohne jeden Umrüstaufwand kann schnell zwischen den Jobs gewechselt werden, was eine Grundvoraussetzung für kostengünstiges Produzieren ist. Die Qualität genügt oft unternehmerischen Ansprüchen, reicht aber noch nicht an die moderner Offsetmaschinen heran. Auch im Drucktempo hinken aktuelle Digitaldrucker noch hinterher[13], holen aber Zug um Zug auf. Der Digitaldruck eignet sich aufgrund seiner Flexibilität für Produkte, die aktuelle, sich häufig ändernde Informationen beinhalten: Flyer, Mailings, große Briefaussendungen, Handbücher, Sonder- und Teilkataloge und so weiter. Besonders spannend sind neue Möglichkeiten im Bereich Personalisierung und Print-on-Demand. Mit dem Digitaldruck ist der vollständig digitale Workflow im Druckhandwerk Realität geworden. Von der Texterfassung in der Textverarbeitung, der Bebilderung mit Hilfe von Digitalfotografie und Scannern sowie dem Layoutdesign per DTP bis zum Druck - die Daten durchlaufen ohne Medienbrüche (zum Beispiel über Filmbelichtung) alle Workflowstufen. Damit lässt sich ein definiertes Maß an Qualitätsverlust realisieren, beispielsweise durch Objekt-spezifische Rasterung von Bilddaten.

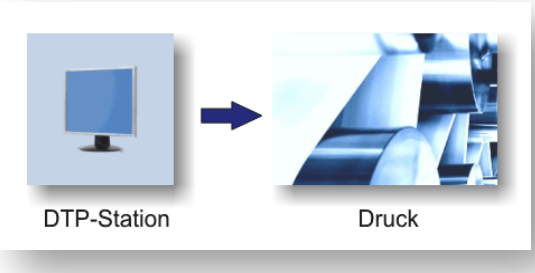

ABBILDUNG I-8: VEREINFACHTER DIGITALDRUCK-WORKFLOW

[11] Ein relativ neues Verfahren, um Druckprodukte mit Glanzeffekten zu veredeln. Dabei wird zunächst an den gewünschten Stellen Kleber auf das Medium aufgetragen, anschließend die Folie mit den Glanzeffekten aufgepresst. Da keine hohen Temperaturen nötig sind, kann ohne Verzögerung schnell und damit kostengünstig weiter gedruckt werden. Weitere Vorteile: Höhere Passgenauigkeit durch gleichmäßige Temperaturen im Druckprozess, auch für empfindliche Druckmedien geeignet.

[12] Bei der klassischen Heißfolienprägung wird die Folie mit den Glanzeffekten unter Hitzeeinwirkung mit dem Bedruckstoff verschweißt. Dies dauert bis zu zehn mal länger als der Kaltfolientransfer.

[13] Bezogen auf den professionellen Offsetdruck.

Vorteile:

- Kaum Aufwand bei Objektwechsel
- Bei kleinen bis mittleren Auflagen kostengünstig
- Sehr flexible Druckprozesse
- Datentransport vollständig digital, keine Medienbrüche

Nachteile:

- Bisher nur moderate Druckgeschwindigkeit
- Bisher nur bis mittlere Druckqualität
- Keine Veredelungen möglich

[Sch07], [Fri01]

Kapitel 2

CONTENT MANAGEMENT

Alles ist Content.

(Häufig gehörter Ausspruch in der Branche, Urheber unbekannt)

Content Management Systeme - Entstehungsgeschichte

Mitte der 90er Jahre offenbarte sich in den IT-Anwenderunternehmen eine neue Problemstellung, welche direkt mit der gleichzeitigen Popularisierung des Internet korrelierte. Immer häufiger mussten digital gespeicherte Informationen (=Inhalte, Content) von eventuell vorhandenen Struktur- und Layoutinformationen befreit und anschließend mit der Seitenbeschreibungssprache HTML neu codiert werden.

Diese Neucodierung war jedoch nur von teuren Fachkräften zu erledigen, sie war fehleranfällig und für ein Medium, welches Prozessgeschwindigkeiten geradezu revolutionierte, extrem langwierig. Für Einzelplatzanwendungen entstand zur Lösung dieser Problematik die Softwareklasse der komfortablen HTML-Editoren mit WYSIWYG-Funktionen, die es auch nicht in HTML versierten Anwendern ermöglichte – wenn auch mit Einschränkungen – Seiten für das Internet zu gestalten und mit Inhalten zu füllen. Für Mehrplatzsysteme mit Anforderungen an Teamwork, Rechtestrukturen und Workflow-Abläufe war jedoch eine eigene Softwaregattung notwendig.

Die grundlegende Systematik dieser Problemstellung ist vergleichbar mit der Entwicklung des Home Computers, später Personal Computers, zur elektronischen Schreibmaschine mit Hilfe von Textverarbeitungsprogrammen. Musste der Anwender anfangs noch Programme zur Ansteuerung des Druckers und zur Ausgabe von Zeichen in einer verfügbaren Programmiersprache oder Scriptsprache[14] verfassen – er musste also jedes Mal beispielsweise einen Brief ‚programmieren' -, so ermöglichten später Textverarbeitungsprogramme die Eingabe von Texten und deren Ausgabe auf einem Drucker ohne jegliche Programmierkenntnisse.

Ansätze zur Lösung dieser Probleme gibt es mittlerweile viele. Die zugehörigen Produkte laufen dann unter Bezeichnungen wie Redaktionssystem, Autorenwerkzeug, Content Management (auch Web Content Management), Database Publishing, Electronic Publishing (E-Publishing) oder ähnlichen Titeln.

[14] Tatsächlich erfreuen sich vor allem im wissenschaftlichen Bereich script-basierte Textverarbeitungssysteme auch heute noch großer Beliebtheit, bspw. TeX. Die ‚Programmierung' in Scriptform findet hierbei allerdings auf einer höheren, abstrakteren Ebene statt und definiert die Formatierung und Gestaltung eines Dokumentes, möglichst unabhängig vom verwendeten Ausgabegerät. Es ist nicht mehr notwendig, direkte, spezifische Befehle für die Ansteuerung eines Druckers oder Bildschirms anzugeben.

Nachteile herkömmlicher Verfahren

Mit der zunehmenden Anzahl von Unternehmen mit eigenem Internetauftritt entscheidet dessen bloßes Vorhandensein immer weniger über den Erfolg dieses Auftrittes als Marketing- oder Vertriebsinstrument. Faktoren wie Aktualität, interessanter Content, Kommunikation, Ergonomie oder ansprechendes Design werden zu den wirklichen Abgrenzungs- und Unterscheidungsmerkmalen.

In der „normalen" Vorgehensweise wird sowohl für das Design, für inhaltliche Erweiterungen oder Änderungen, als auch die Administration und Weiterentwicklung einer Internetpräsenz ein Mitarbeiter mit Erfahrung in diversen Internet-Technologien (HTML, FTP, etc.) benötigt, der so genannte „Webdesigner". Dies wird sich im Bereich Design, Administration und Weiterentwicklung in absehbarer Zeit auch nicht ändern. Die bereitzustellenden Informationen werden jedoch in der Regel von anderen Personen geliefert (Unternehmensleitung, PR-Abteilung, Marketing, Verwaltung, etc.).

Aufgrund der genannten technischen Hürden können Informationen nicht direkt, sondern nur über den Zwischenschritt des Webdesigners bereit gestellt werden.

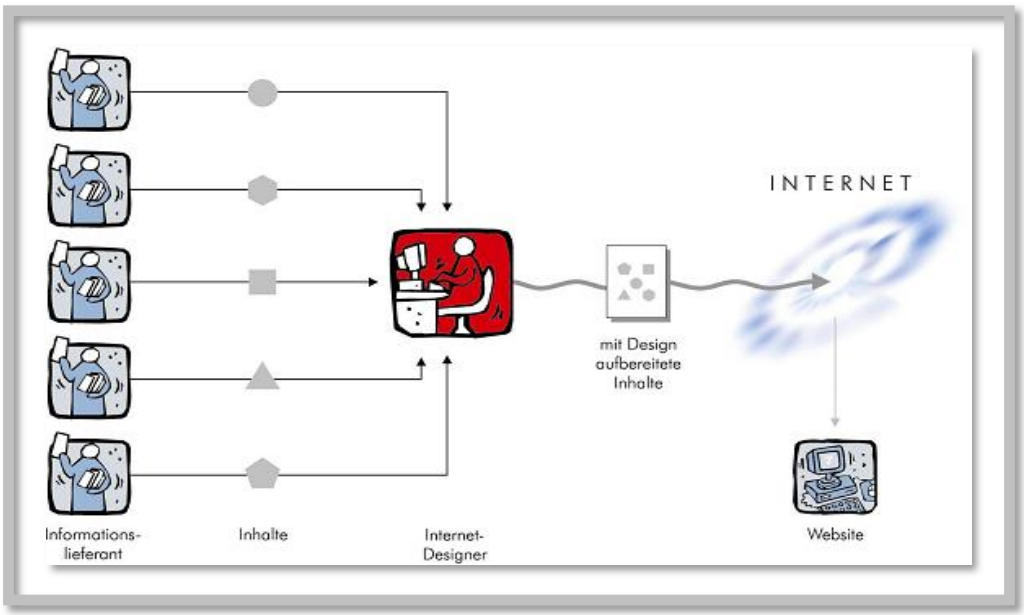

ABBILDUNG I-9: OHNE EIN CONTENT MANAGEMENT SYSTEM IST DIE WEBSITE-PFLEGE AUFWÄNDIG UND FEHLERANFÄLLIG UND DER WEBDESIGNER WIRD ZUM FLASCHENHALS.

Der kompetente, phasenweise knappe und teure Webdesigner bildet hier einen Engpass im Prozessablauf. Noch problematischer wird der Arbeitsablauf, wenn Informationen von verschiedenen geografischen Standorten einfließen sollen oder beispielsweise ein multilingualer Auftritt zu realisieren ist. Die Synchronisation und Wiederverwendung von Informationen bedarf in diesen Fällen einer komplexen und aufwändigen Organisation des Ablaufs.

Vorteile von Content Management Systemen

Ein Content Management System (CMS) entlastet den Webdesigner im Bereich der Contenteingabe und -pflege. In der Regel bieten CMS einfach zu verstehende Eingabemasken an, in denen die einzelnen Fachabteilungen Content selbständig einpflegen können. Der Webdesigner muss sich nur noch um das Design und die Erstellung der Layout-Templates kümmern. Sein Aufwand wird also drastisch reduziert. Die Templates bestimmen das spätere Aussehen eines Beitrages auf der Website. Das CMS füllt in den Templates des Webdesigners Platzhalter mit dem Content der Fachabteilungen und generiert daraus die fertigen (X)HTML-Seiten.

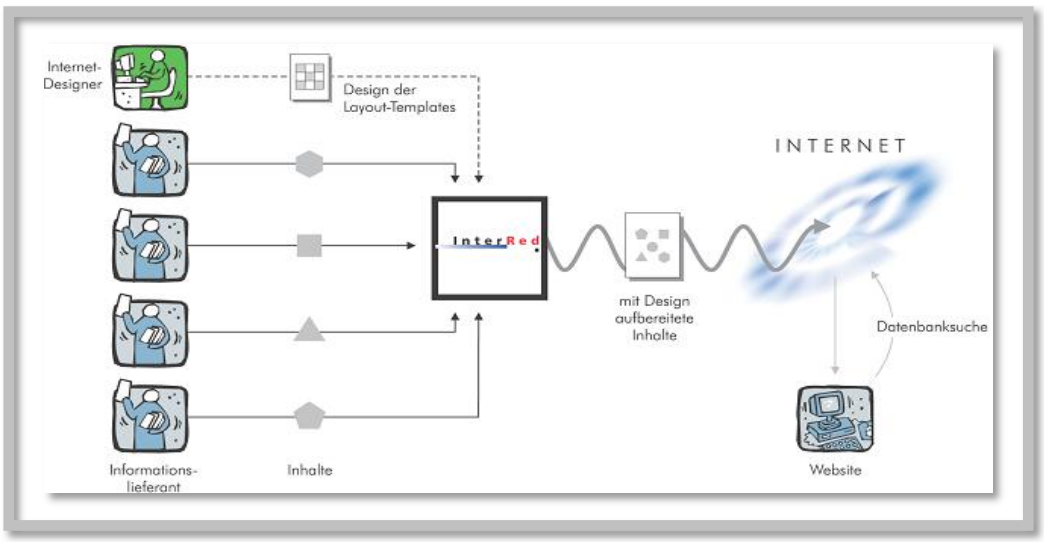

ABBILDUNG I-10: DER WEBMASTER IST NUR NOCH EIN GLIED IN DER INFORMATIONSKETTE.

Bei einem Content Management System mit offener Template-Technologie sind dem Webdesigner in der Gestaltung der Design-Templates kaum Grenzen gesetzt. Er kann sowohl seine gewohnten Design-Werkzeuge weiterhin benutzten, als auch alle verfügbaren Internet-Technologien verwenden.

Content Life Cycle

Durch die fortschreitende Integration lassen sich viele Systeme immer schwieriger voneinander abgrenzen. Lösungen für Content Management, Dokumenten Management, Knowledge Management, Cross Media Publishing sind funktional nicht disjunkt. Gemeinsam ist allen CMS-Ausprägungen die Realisierung des Content Life Cycle, also die Erstellung, Kontrolle, Freigabe, Publizierung und Archivierung von Webseiten. [Chr03 S. 15]

Zur Absicherung von redaktioneller Qualität und juristischer Legalität, ist es nötig, die Inhalte systematisch zu produzieren und zu publizieren. Jeder Inhalt durchläuft dabei von der initialen Erstellung bis zur Archivierung verschiedene Stadien innerhalb des Content Life Cycle.

Erstellung: In dieser Phase setzt der Autor seine eigene oder eine fremde Idee um. Der so erstellte Content kann normaler Text, aber auch eine Grafik oder ein anderer multimedialer Bestandteil eines Dokumentes oder einer Website sein und eventuell bereits bestehende Inhalte aus dem Archiv referenzieren.

Kontrolle und Freigabe: Der erstellte Content wird durch einen entsprechend autorisierten Mitarbeiter überprüft. Ist die Kontrolle positiv, wird der Content freigegeben und zur Publikation weitergereicht. Anderenfalls wird der Inhalt zur Korrektur an den Autor zurückgegeben.

Publikation: Freigegebene Inhalte werden im Web publiziert und damit für den jeweiligen Benutzerkreis zugänglich gemacht.

Archivierung: Ein erstellter Inhalt wird nach einer bestimmten Zeit archiviert. Dabei wird er entweder ganz aus dem Netz genommen und nur intern archiviert oder innerhalb der Website in einen speziellen Archiv-Bereich gestellt, um aktuelleren Beitragen Platz zu machen.

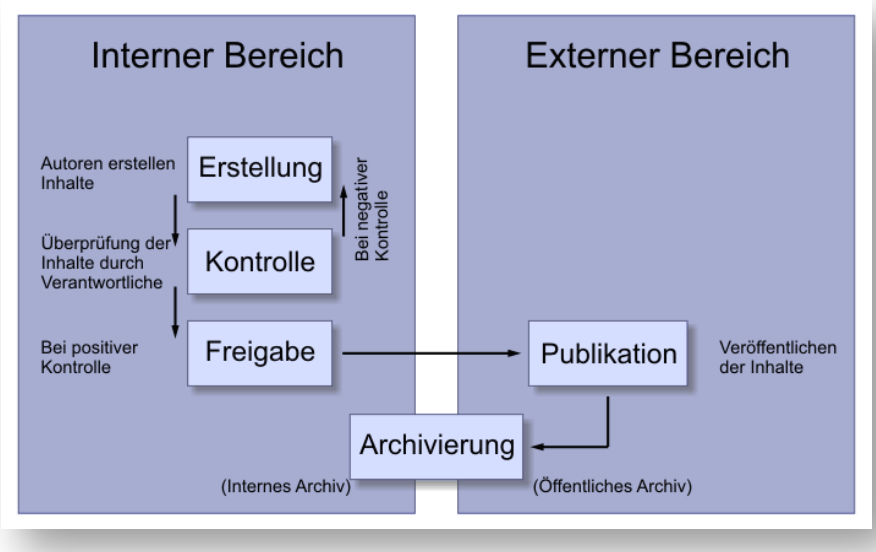

ABBILDUNG I-11: DER CONTENT-LIFE-CYCLE NACH [ZSC02 S. 56]

Asset und Templates

Mit einem einfach zu bedienenden Content Management System können einzelne Fachabteilungen ihre Inhalte für die Website eigenständig erzeugen und pflegen. Dabei muss sich der einzelne Anwender weder um das Design noch um technische Aspekte kümmern. Eingaben und Änderungen erfolgen über Eingabeformulare, die vom System für einzelne oder kombi-

nierte Assets[15] zur Verfügung gestellt werden. Definiert werden Assets bei kleineren Projekten meistens im Zuge der Erstellung der Templates, bei größeren im Vorfeld durch eine objektorientierte Analyse.

In Content Management Systemen treten jedoch neben statischen (Content-)Assets weitere Asset-Typen auf, die ebenfalls inhaltlicher Natur sind, deren Erstellungsprozess sich aber typischerweise nicht immer innerhalb des Content Life Cycle abdecken lässt. Die Gesamtheit dieser digitalen Assets lässt sich wie folgt klassifizieren [Zsc02 S. 40]:

Übliche Webinhalte: Texte, Bilder und Links, welche das Grundgerüst einer Website darstellen.

Multimediale Assets: Audio- und Videoformate, die teilweise vom Browser nativ wiedergegeben werden, teilweise aber Plug-Ins benötigen. Beispiele: Macromedia Flash, RealAudio, RealVideo, Apple Quicktime, Windows Media Video.

Applikationsgebundene Assets: Als Download angebotene Datei, welche zur Darstellung das Programm benötigen, mit dem sie erstellt wurden. Beispiele hierfür sind Office Dokumente, proprietäre Grafikformate oder gepackte Archive.

Transaktionelle Assets: Diese dienen als Container für Content, der im Zusammenhang mit Transaktionen und Benutzersitzungen entsteht. Inhalte dieser Art sind die Grundlage für Personalisierung und E-Commerce-Anwendungen jeder Art.

Community-Assets: Solche Inhalte entstehen durch die Mitglieder einer Community in den Bereichen, in denen ein Dialog zwischen den Teilnehmern oder zwischen Teilnehmern und Anbietern möglich ist, also beispielsweise in Diskussionsforen oder Chat Areas. Im Rahmen des Web 2.0-Hypes haben solche Inhalte eine enorme Aufwertung erfahren. Fachtermini für Community-Assets sind UGC (User Generated Content) und UGF (User Generated Feedback).

Ein weiteres zentrales Merkmal eines CMS ist die getrennte Ablage der einzelnen Komponenten. Da die digitalen Assets nur immer einmal abgelegt und mehrfach referenzierbar sind, können diese dann sehr effizient und schnell an einem zentralen Ort, dem Content Repository, verwaltet und bearbeitet werden. Technisch lässt sich das Content Repository auf verschiedene Arten umsetzen. So ist eine Speicherung der Assets im Dateisystem, gegebenenfalls ergänzt mit Dateien zur Aufnahme von Metadaten, ebenso möglich wie eine Ablage in einer relationalen oder objektorientierten Datenbank oder als XML-Dokumente. [Ehl03 S. 113]

In die Layout-Templates fügt der Entwickler für das Content Management System spezifische, proprietäre Tags ein. Sie definieren die Art und Weise, wie die erfassten Assets bei der späteren Erzeugung der HTML-Seiten in die Templates eingefügt werden. Die Tags von InterRed Online enthalten neben dem einfachen Anzeigen eines Content-Elementes auch

[15] In der Literatur wird ‚Asset' unterschiedlich definiert. Oft wird darunter die strukturierte Kombination einzelner Content-Bestandteile verstanden, manchmal in Form von Objekten mit Methoden und Eigenschaften. [Zsc02] definiert ein Asset als eine Kombination eines Inhaltsobjektes mit Verwendungsrechten. In Anlehnung an XML verwenden manche Hersteller auch den Begriff Dokumenttypdefinition (DTD). InterRed Online verwendet den eigenen Begriff ‚Beitragsart'.

Befehle für komplexere imperative und deklarative Anweisungen, mit denen beispielsweise Schleifen für automatische Inhaltsverzeichnisse oder Anweisungen für die konditionale Ausgabe von Inhalten realisiert werden können.

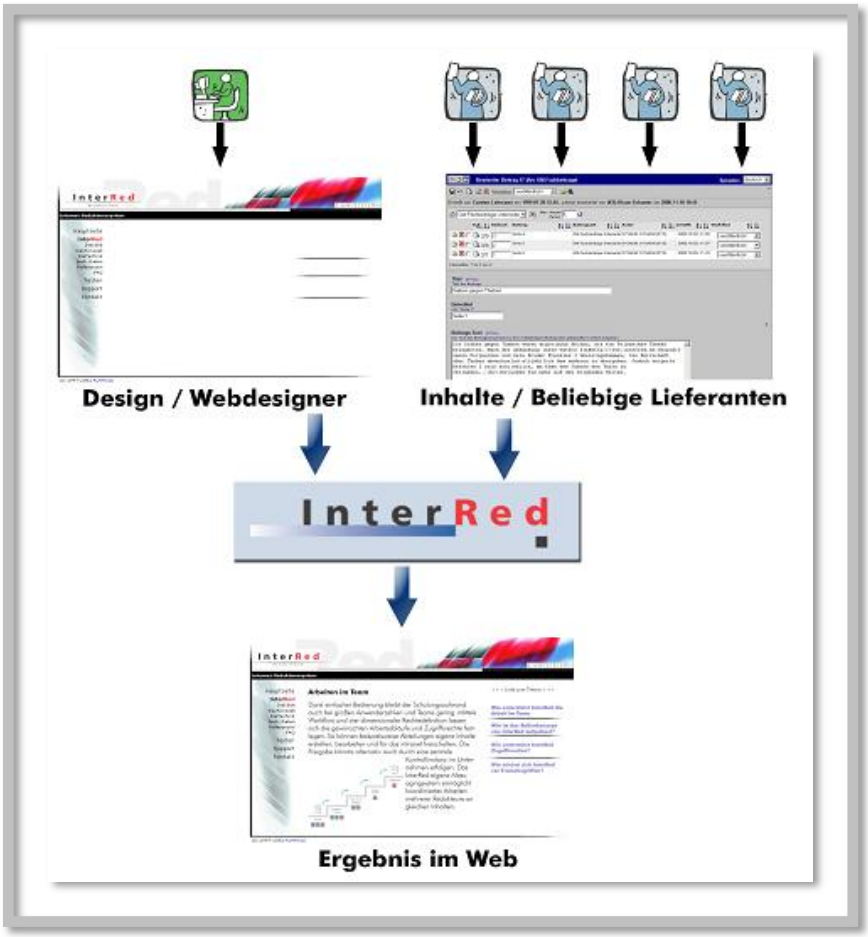

ABBILDUNG I-12: INTERRED ONLINE GENERIERT AUS DEM DESIGN UND DEM CONTENT DIE FERTIGE WEBSITE

Grundfunktionen

Von aktuellen Content Management Systemen erwarten die Anwenderunternehmen eine einfache, intuitive Bedienung mit einer durchgängigen Browseroberfläche. Eine Client-Installation kann bei rein browserbasierten Systemen komplett entfallen. Fast alle CMS erfordern heute am Arbeitsplatz für Anwender und Administratoren lediglich einen Internetzugang und einen normalen Internet-Browser mit aktiviertem Java Script. Mittlerweile zählt das zur Basisausstattung eines jeden PCs, unabhängig ob das Betriebssystem Windows, Mac oder Linux heißt. Damit lässt sich ein solches System weltweit ohne besondere Voraussetzungen nutzen.

Statische und dynamische Inhalte

Für das technische Verfahren zur Auslieferung der HTML-Seiten setzen Content Management Systeme im Wesentlichen zwei unterschiedliche Methoden ein:

1. *Dynamische CMS*

In den meisten Systemen werden die HTML-Seiten für jede Auslieferung durch den Webserver mit den aktuellen Inhalten aus der Datenbank des Content Management Systems generiert. Diese Content Management Systeme bezeichnet man auch als dynamische CMS. Hauptvorteile dieses Verfahrens sind die machbar höchste Aktualität der angezeigten Daten – Verzögerungen liegen hier je nach eingesetztem Cache-Mechanismus bei einer Größenordnung von wenigen Sekunden bis Minuten – und die einfachere Implementierung im Content Management System für die Generierung von Indexseiten, Inhaltsverzeichnissen, Querverlinkungen und Navigationen.

2. *Statische CMS*

Im anderen Verfahren generiert das Content Management System in einem gesonderten Prozess, dem sogenannten Publikationsprozess, statische HTML-Seiten und stellt diese dem Webserver zum Beispiel per FTP zur Verfügung. Eine Anfrage an die Datenbank des Content Management Systems ist bei der Auslieferung einer Seite an den Surfer nicht notwendig. Hauptvorteile dieses Verfahrens sind die höhere Verfügbarkeit des Webauftrittes, da ein Problem oder Wartungsarbeiten am Content Management System oder dessen Datenbank nicht zu einem Ausfall der Webseite führen, die höhere Geschwindigkeit bei vergleichbarer Hardware-Basis sowie die einfachere Aufbereitung der Inhalte für Internet-Suchmaschinen.

Ein Nachteil ist die geringere Aktualität - Verzögerungen liegen hier bei einer Größenordnung von mehreren Minuten bis zu Stunden – abhängig von Ort und Anzahl der zu beliefernden Webservern, dem Umfang der zu aktualisierenden Daten und der Leistungsfähigkeit des Publikationsmechanismus. Komplexe Auftritte statisch zu realisieren bedeutet außerdem einen hohen Entwicklungsaufwand für die Publikations- und Verteilungsmechanismen.

Medienneutrale Speicherung

Im Großteil der Content Management Systeme werden Inhalte und Layout getrennt gespeichert. Die Inhalte liegen also als abstrakte Information ohne Formatierungsanweisungen vor. Für die Ausgabe von Content im Internet werden neben reinen Formatierungsanweisungen für die Gestaltung einer HTML-Seite Metadaten für eine Strukturbildung benötigt. Geht man davon aus, dass nicht der gesamte Content in einer langen HTML-Seite ausgegeben werden soll, sondern auf einzelnen Seiten, die über Hyperlinks, beispielsweise in Form einer Navigation, erreichbar sind, so müssen die Inhalte in eine meist hierarchische Struktur eingefügt werden.

Sind Content Management Systeme in der Lage, diese Strukturinformationen getrennt von den Inhalten zu speichern, bringen sie eine notwendige Voraussetzung mit, um Inhalte medi-

enneutral zu speichern und auf unterschiedlichen Medien auszugeben. Zählen zu diesen Ziel-Medien auch professionell gestaltete Print-Objekte wie Zeitschriften, Tageszeitungen, Magazine, etc., dann wird ein Content Management System zu einem Cross Media Publishing-System. Damit steigt die Anzahl der Formatierungsoptionen und das Cross Media Publishing-System muss auch solche Metadaten verarbeiten, die für eine reine Online-Ausgabe gar nicht relevant sind. Eine ausführlichere Analyse dazu findet sich auf S. 35ff.

Das getrennte Speichern der Strukturinformation ist jedoch zwingende Voraussetzung, weil die Ausgabe von Content auf beispielsweise mobilen Geräten, CD/DVD oder PDF jeweils eigene strukturbildende Definitionen benötigt.

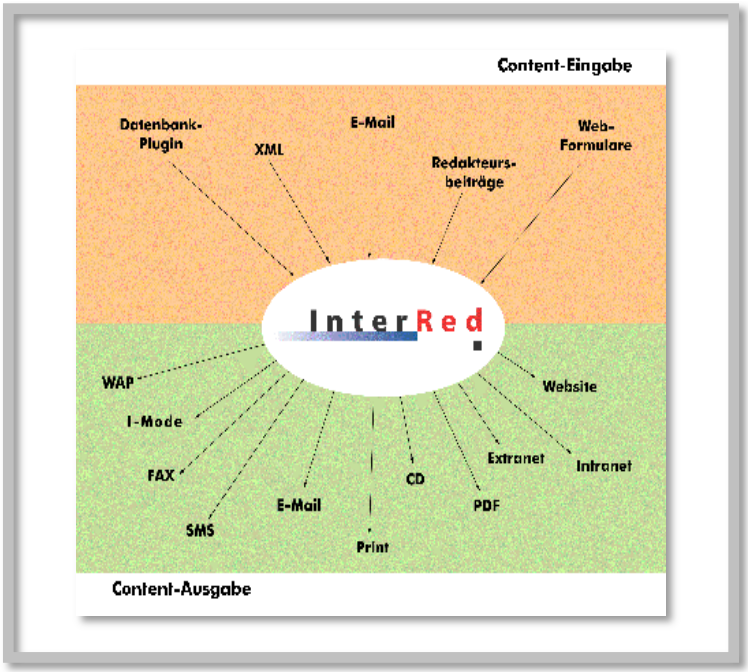

ABBILDUNG I-13: DAMIT EINE MEHRFACHVERWENDUNG VON INHALTEN MÖGLICH WIRD, IST EINE MEDIENNEUTRALE SPEICHERUNG IM CONTENT MANAGEMENT SYSTEM NOTWENDIG.

Workflow, Rechtesystem und Versionierung

Weitere wichtige Eigenschaften von Content Management Systemen sind die Unterstützung von Workflows, ein flexibles Rechteverwaltungssystem und die Versionierung von Inhalten.

Mit Hilfe von Workflowsystemen können vorhandene Arbeitsabläufe und Informationsflüsse im Unternehmen abgebildet werden. Es können automatisch Aufgaben auch an externe Mitarbeiter vergeben und weitere Services, Dienste, etc. eingebunden und gesteuert werden. Da diese Prozesse zwischen den Unternehmen divergieren, sollte das Workflowsystem genügend Flexibilität bieten, um die verschiedenen Anforderungen der Anwender abbilden zu können.

Ein skizzenhafter Workflow könnte so aussehen: Zunächst legt ein Mitarbeiter einen neuen Beitrag auf der Stufe „in Bearbeitung“ im Content Management System an. Sobald er seine Arbeiten abgeschlossen hat, stellt er seinen Beitrag auf die Stufe „abgeschlossen“. Nun sorgt das Workflowsystem dafür, dass dieser Beitrag beispielsweise dem Abteilungsleiter vorgelegt wird. Ist er mit der Arbeit einverstanden, so kann er ihn durch Setzen der Stufe „freigegeben“ zur Veröffentlichung freigeben. Falls nicht, setzt er ihn zurück auf „in Bearbeitung“, der Mitarbeiter bekommt ihn dann noch einmal zur Überarbeitung vorgelegt. Typische Rollen im Verlagsumfeld sind beispielsweise Redakteure, Grafiker, CvD oder Projektleiter.

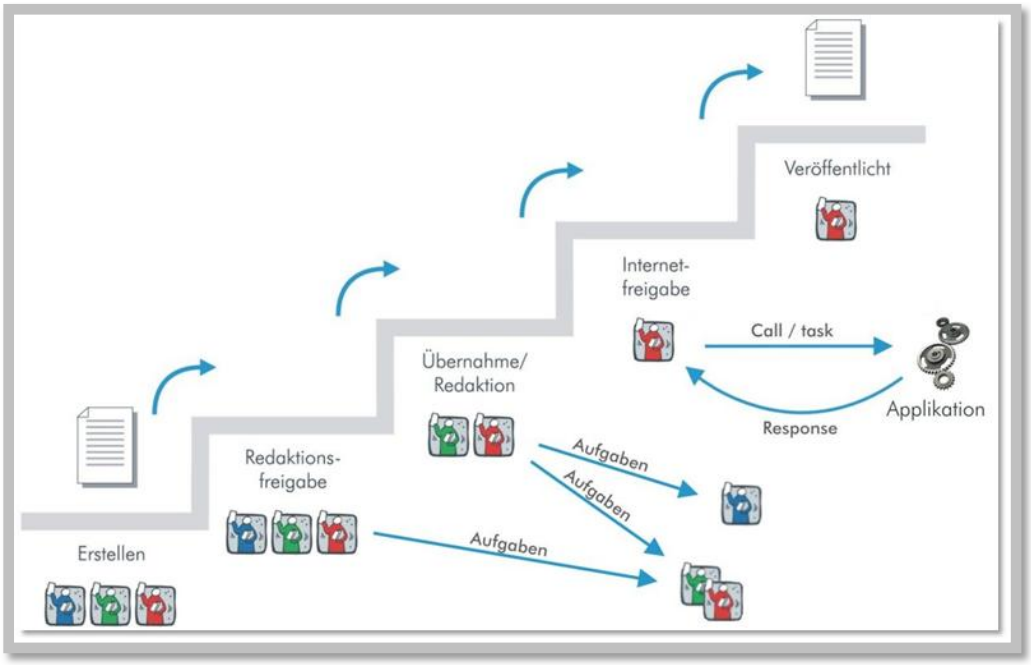

ABBILDUNG I-14: EIN FLEXIBLES WORKFLOWSYSTEM UNTERSTÜTZT DIE ARBEIT IN GRÖßEREN TEAMS UND FÖRDERT EINE HÖHERE QUALITÄT DER ERGEBNISSE.

Dieses Spektrum an Aufgaben und Verantwortungsbereichen muss so in einem CMS abgebildet werden können, dass möglichst wenig technologisch bedingte Eingriffe in die erprobten Arbeitsabläufe notwendig sind. Damit bei der Abarbeitung einer Workflowinstanz nur die dazu berechtigten Mitarbeiter die Arbeitsschritte ausführen, ist eine enge Verknüpfung der Komponenten Workflow und Benutzerverwaltung notwendig. [Ehl03 S. 118]

In einer konzeptionell durchdachten Ausgestaltung der Rechte- und Rollendefinition liegt ein wichtiges Basiselement für Funktion und Wartbarkeit einer CMS-Implementation. Bei sehr kleinen Teams kann es durchaus noch Sinn machen, eine subjektbezogene Rechtezuordnung an einzelnen Nutzern festzulegen. In größeren Teams mit entsprechender Fluktuation ist die Rechtezuordnung an Gruppen und Rollen flexibler. Damit lassen sich beispielsweise alle Mitarbeiter, die Texte bearbeiten dürfen, zu einer Gruppe ‚Redakteure‘ zusammenfassen. Innerhalb dieser Gruppe können Rollen wie Chefredakteur oder Ressortleiter definiert werden. Eine weitere Verbesserung der Prozessqualität wird durch automatisch generierte Nachrichten und Statusmeldungen erreicht, die in Abhängigkeit von Workflowstati beispielsweise Übersetzer über anstehende Aufgaben informieren. Die Nachrichten können per E-Mail oder über systeminterne Anzeigen verteilt werden. Technisch wird dies über mehrere Zu-

stände der eingebundenen Assets realisiert. Interaktionen mit diesen Assets können Zustandsänderungen auslösen, die wiederum sekundäre Verarbeitungsschritte an-stoßen können. [Rot03 S. 132]

Da bei der rollen- bzw. gruppenbezogenen Modellierung des Workflows potenziell mehr als eine Person die gleiche Aktion ausführen kann, ist im CMS neben Workflowsystem und Rechteverwaltung ein Verfahren für den konkurrierenden Zugriff auf Daten notwendig. Diese Kontrolle der Nebenläufigkeit ist ein bekanntes Problem in der Informatik und wird häufig in der Datenbanktheorie behandelt.[Kel03] Man unterscheidet dabei zwei Ansätze:

- **optimistische** Nebenläufigkeitskontrolle: Beispielsweise durch den Algorithmus von Kung & Robinson. Er basiert auf der Beobachtung, dass die Wahrscheinlichkeit für den Zugriff zweier Transaktionen auf ein Objekt relativ gering ist. Die Transaktionen werden bis zum Ende fortgesetzt und erst mit dem Commit erfolgt eine eventuelle nötige Konfliktbehandlung. Die Folge können Abbruch oder Neustart der Transaktion sein. Bei der optimistischen Nebenläufigkeitskontrolle können zudem Inkonsistenzen auftreten. Vorteil ist die geringere Blockadehäufigkeit im Arbeitsablauf.
- **pessimistische** Nebenläufigkeitskontrolle: Verfahren mit zentraler und dezentraler Kontrolle der Sperren. Datenkonsistenz wird gesichert. Verwaltung der Sperren kann aufwändig werden, die Sperrgranularität hat Einfluss auf den Grad der Nebenläufigkeit.

In einer typischen transaktionsorientierten Client/Server-Architektur arbeitet die Nebenläufigkeitskontrolle transparent für den Anwender im Hintergrund, im besten Fall nimmt der Benutzer sie bei der Arbeit überhaupt nicht wahr.[Kel02], [von93] Da ‚HTTP' als Request/Response-Protokoll Transaktionen nicht direkt unterstützt, muss der Benutzer von browserbasierten CMS Daten zur Bearbeitung explizit Ein- und Auschecken.[16] Ein ausgechecktes Datenelement ist bei pessimistischer Nebenläufigkeitskontrolle für die Bearbeitung durch alle anderen Personen gesperrt. Je nach Gestaltung des Workflows kann es sinnvoll sein, eine schreibgeschützte Version des aktuellen Bearbeitungsstandes bereit zu stellen.

[16] Im üblichen Sprachgebrauch als Check-In und Check-Out bezeichnet.

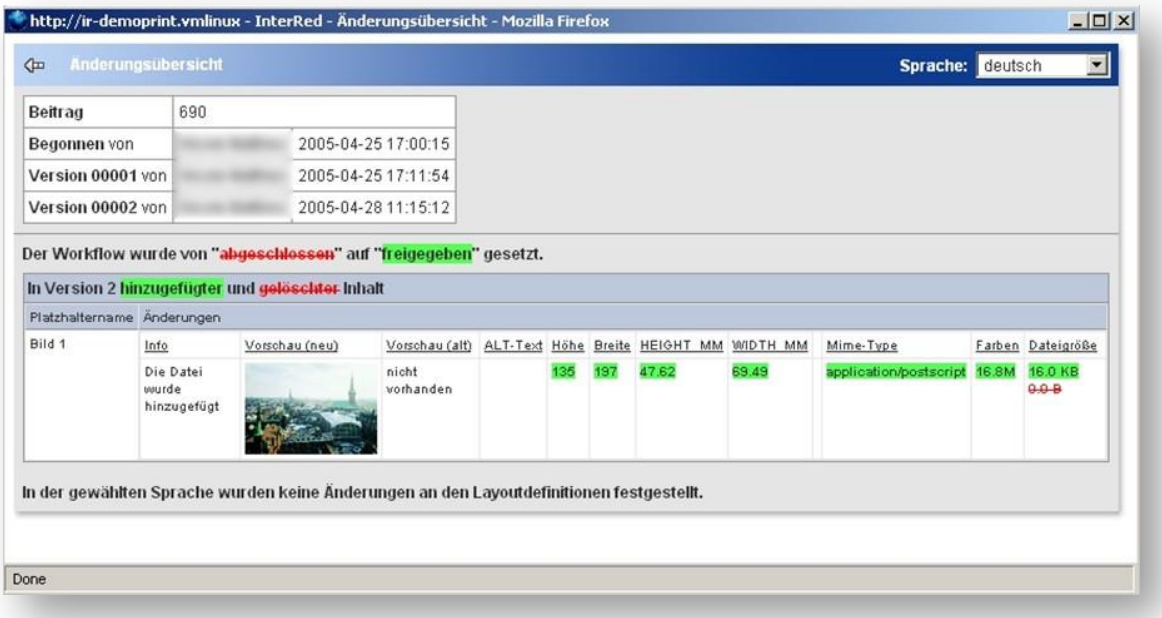

ABBILDUNG I-15: FARBLICH HINTERLEGTE ÄNDERUNGEN AN WORKFLOW UND INHALT IN DER VERSIONSHISTORIE.

Content Management Systeme sind zwar eine relativ junge Softwaregattung, sie haben sich aber vergleichsweise schnell und breit horizontal diversifiziert. Eine allgemein benutzte Klassifikation aller Gattungen liefert das Themen-Portal contentmanager.de mit einer aktuellen und umfangreichen Marktübersicht.[Fei07] Einige Vertreter der Gattung ‚Enterprise CMS' (ECMS) entwickelten sich in Richtung der traditionellen Dokumentenmanagement Systeme (DMS). Diese Entwicklung und das Inkrafttreten der GDPdU[17] im Jahre 2001 machte eine Erweiterung der Nebenläufigkeitskontrolle um eine Dokumenten-echte Versionierung erforderlich.

Auf die weiteren Funktionen und Merkmale professioneller CMS[18] soll an dieser Stelle nicht weiter eingegangen werden, es wird auf die Literaturangaben verwiesen. Besondere Eigenschaften, die unter dem Aspekt der medienneutralen Datenhaltung, Datenaustausch und Datenausgabe eine Rolle spielen, werden im weiteren Verlauf dieser Arbeit an geeigneter Stelle behandelt.

[17] Grundsätze zum Datenzugriff und zur Prüfbarkeit digitaler Unterlagen

[18] Beispielsweise Suchmaschinen für strukturierte oder intelligente Suche, Personalisierung, Besucherstatistiken, Business Process Management, Portalserver, Intranet-spezifische Funktionen, Newsletter-Generierung, Mandantenfähigkeit, Teasermanagement, Content Syndication, etc.

Kapitel 3

CROSS-MEDIA MANAGEMENT

Nach Jahren kontinuierlichen Wachstums steckt die gesamte Verlagsbranche derzeit in einer Krise, die bisher Verlagsmanager nur vom Hörensagen aus anderen Branchen bekannt war. Die Krise ist dabei sowohl auf die wirtschaftliche Stagnation und das eingetrübte Werbeklima zurückzuführen als auch auf ein strukturelles Problem, das sich mit fortschreitender Digitalisierung verstärkt. Die negativen Erfahrungen der zahlreichen von Verlagshäusern initiierten und wenig später gescheiterten Internet-Projekte sollten nicht das Blickfeld der Verlagsmanager auf eine „Back-to-the-roots"-Strategie und eine bloße Konzentration auf Print-Produkte verengen. Gefragt ist Flexibilität gegenüber den Medienpräferenzen der Leser, so dass Crossmedia zur alles entscheidenden Handlungsmaxime der nächsten Jahre für einen Verlag werden sollte. [19]*(Christoph Weger)*

Seit Beginn der Digitalisierung befindet sich die Medienbranche in einer Phase tiefgreifender Veränderungen. Aus der fortschreitenden Konvergenz der Bereiche Medien, Informations- und Telekommunikationstechnologie entstehen für die Unternehmen der Konvergenzbranchen neue strategische Chancen und Risiken.[Koc04] Wie die Erfahrungen mit dem Internet aufgezeigt haben, bedeutet jeder neue Distributionskanal zunächst eine höhere kaufmännische, konzeptionelle und technische Komplexität in der Wertschöpfungskette der Medienunternehmen, die fast immer auch höhere Kosten bedeuten. Neue gewinnbringende Geschäftsmodelle sind nur mit innovativen Konzepten und ausgewählten Inhalten bzw. Formaten realisierbar. Auf die neuen digitalen Kanäle zu verzichten, ist jedoch keine Option. Quer durch alle Altersschichten verhalten sich die Konsumenten immer mehr ‚crossmedial', besonders signifikant in der Zielgruppe der 18 bis 30-jährigen. Gestützt wird diese Entwicklung durch immer leistungsfähigere Endgeräte, die regelmäßig auch wieder neue Formate bedingen.[Mül02] In welcher Form und auf welchen Geräten Content zukünftig ausgegeben werden muss, ist also nicht abzusehen. Fast in jährlichem Turnus erhöhen sich die Auflösungen mobiler Endgeräte, die durchschnittlich benutzte Auflösung am Desktop-PC und am Laptop

[19] Aus dem Geleitwort von Dr. Christoph Weger zu Keuper F./Hans R. (2003): Multimedia-Management. Dr. Weger war zu dieser Zeit Managing Director Electronic Media bei der Financial Times Deutschland.

erhöht sich ebenfalls permanent. Davon betroffen ist die optimierte Ausgabe von Bildmaterial mit passender Qualität, auch Textinhalte müssen mit passendem Format und geeigneter Typographie ausgegeben werden können. Eine medienneutrale Datenhaltung ist also essentiell.

Das Thema Cross-Media ist aus strategischer Sicht für die Medienindustrie unausweichlich, erfolgreiches Cross-Media-Management ist für die Steigerung von Gewinn und Unternehmenswert kritisch.

Definition Cross-Media Management

> *Cross-Media Management ist die integrierte Planung, Implementierung und Steuerung medienübergreifender Vermarktungskonzepte mit dem Ziel, vorhandene Marken, Inhalte und Kundenbeziehungen wertsteigernd crossmedial zu nutzen.*

Vgl. [Mül02]

Cross-Media Management ist in dreifacher Hinsicht ein neuer Ansatz zur Steuerung von komplexen Vermarktungsprozessen im Medienumfeld:

Ausgeprägte Kundenorientierung: Durch den Bezug auf die Vermarktungsabsicht hat Cross-Media Management immer einen starken Kundenbezug, und zwar im Hinblick auf Content- als auch Werbekunden.

Integration: Die Besonderheit liegt in der integrierten Steuerung aller Prozesse, die wertrelevant sind. Dies betrifft neben der Integration der kaufmännischen, redaktionellen und technischen Perspektive insbesondere auch die übergreifende Steuerung auf Basis entsprechender Kennzahlen (*Cross-Media Measurement*).

Wertsteigerung: Cross-Media Management hat zum Ziel, vorhandene Kernkompetenzen und Marken wertsteigernd crossmedial zu nutzen. Damit ist das Cross-Media Management den integrierten horizontalen Steuerungsansätzen zuzurechnen, die auch nicht-finanzielle Kennzahlen explizit einbeziehen. [Mül02]

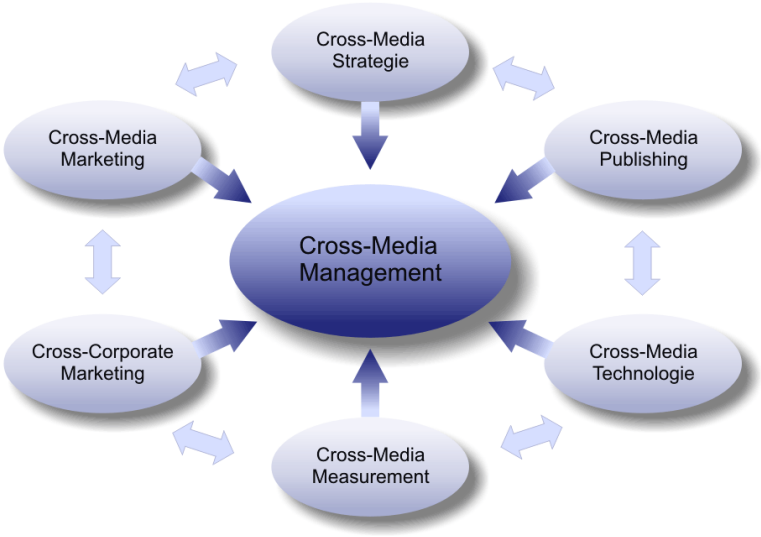

ABBILDUNG I-16: STEUERUNGSBEREICHE VON CROSS-MEDIA-MANAGEMENT (NACH [MÜL02])

Strategische Programme im Management von Medienunternehmen basieren allgemein auf der Analyse aus vier Perspektiven:[Sch06]

- Die produktorientierte Perspektive. Produkte sind das wesentliche Ergebnis der unternehmerischen Tätigkeit von Medienunternehmen. Ihrem Management kommt demnach eine besondere Bedeutung zu. Im Gegensatz zu anderen Produkten weisen Medienprodukte in der Regel für meist zwei Kundengruppen einen Nutzen auf. Rezipienten, als direkte Konsumenten von Medienprodukten, ziehen einen Nutzen aus dem Informations-, Bildungs- und Unterhaltungswert des Produktes, während die Werbewirtschaft einen Nutzen aus der durch das Medienprodukt generierten Konsumentenaufmerksamkeit zieht und diese für die Vermittlung von Werbebotschaften nutzt.
- Die ressourcenorientierte Perspektive. Mittel- und langfristig sind zwei Ressourcenarten von besonderer Bedeutung, die einen zentralen Einflussfaktor auf Produktion und Kosten darstellen: die Ressource Personal und die Ressource Anwendungssystem.
- Die kaufmännische Perspektive. Hierbei handelt es sich um die klassischen kaufmännischen Felder Rechnungswesen und Finanzwirtschaft.
- Die managementorientierte Perspektive. Sie soll sicher stellen, dass das Handeln aller Beteiligten auf die Unternehmensziele ausgerichtet ist und betrachtet dazu die Gestaltungskräfte und Steuerungsmaßnahmen des Managements, welches diese zielgerichtete Koordination bewirkt.

Unter dem Aspekt der stetigen Zunahme der Zielmedien leiten sich daraus ökonomische Erklärungen verschiedener Varianten von Cross-Media-Strategien ab. [Sju05] definiert den Begriff „Cross-Media Strategien“ als Diversifikationsentscheidungen von Medienunternehmen, die als Zielbranchen andere Medienteilmärkte fokussieren, also cross-mediär sind.[20]

Zur Kennzeichnung von Diversifikationsstrategien werden zwei Kriterien herangezogen. Zum einen unterscheidet man den Verwandtschaftsgrad von Ressourcen, Technologie und Risiko von Ausgangs- und Zielbranche. Eine Diversifikation, die auf ein Geschäftsfeld innerhalb der eigenen Wertschöpfungskette zielt, wird als „related diversification“ bezeichnet, eine „unrelated diversification“ ist eine Diversifikation in eine unverwandte Branche.

Zum anderen ist das Verhältnis von Ausgangs- und Zielbranche im Hinblick auf ihre Position in der Wertschöpfungskette relevant. Eine Diversifikation, welche die gleiche Wertschöpfungskette fokussiert, wird als „horizontal“, eine Diversifikation, die auf eine vor- oder nachgelagerte Wertschöpfungskette zielt, wird als „vertikal“ bezeichnet.

Auf Basis dieser Kriterien entwickelt Sjurts eine Typologie von Diversifikationsstrategien und identifiziert daraus die relevanten Cross-Media Strategien.

Es lassen sich drei grundlegende Diversifikationsvarianten wie folgt definieren:

Intramediäre Diversifikation: Im Falle einer intramediären Diversifikation wird ein Medienunternehmen aktiv in einer vor- oder nachgelagerten Branche der brancheneigenen Wertschöpfungskette.

Intermediäre Diversifikation: Bei der intermediären Diversifikation tritt ein Medienunternehmen in eine andere Medienteilbranche ein. Folglich handelt es sich um eine Cross-Media Strategie. Nach dem Verwandtschaftsgrad der Ausgangs- und Zielbranche kann hier weiter in eine verwandte, unverwandte und eine konvergenzgetriebene intermediäre Diversifikation unterschieden werden.

Eine verwandte intermediäre Diversifikation liegt vor, wenn ein Medienunternehmen in eine technologisch verwandte Medienteilbranche eintritt. Beispiele sind der Markteintritt eines Zeitungsverlags in die Zeitschriftenbranche oder die Präsenz eines Hörfunksenders auch im Fernsehbereich.

[20] Einer Sichtweise, die unter Cross-Media Strategien die kombinierte, gekoppelte Werbung in verschiedenen Medien durch beispielsweise Werbeagenturen versteht, kann dagegen nicht gefolgt werden. Hier wird der Strategiebegriff zur Bezeichnung eines Sachverhaltes gebraucht, der ein strategisches Programm, aber keine Strategie darstellt.

Bei einer unverwandten intermediären Diversifikation werden Medienunternehmen in einer klassischen Medienteilbranche tätig, die zu ihrer eigenen Teilbranche in keiner technologischen Verwandtschaft steht. Dies ist beispielsweise der Fall, wenn Verlage im Rundfunk aktiv werden oder Fernsehsender sich im Printbereich engagieren.

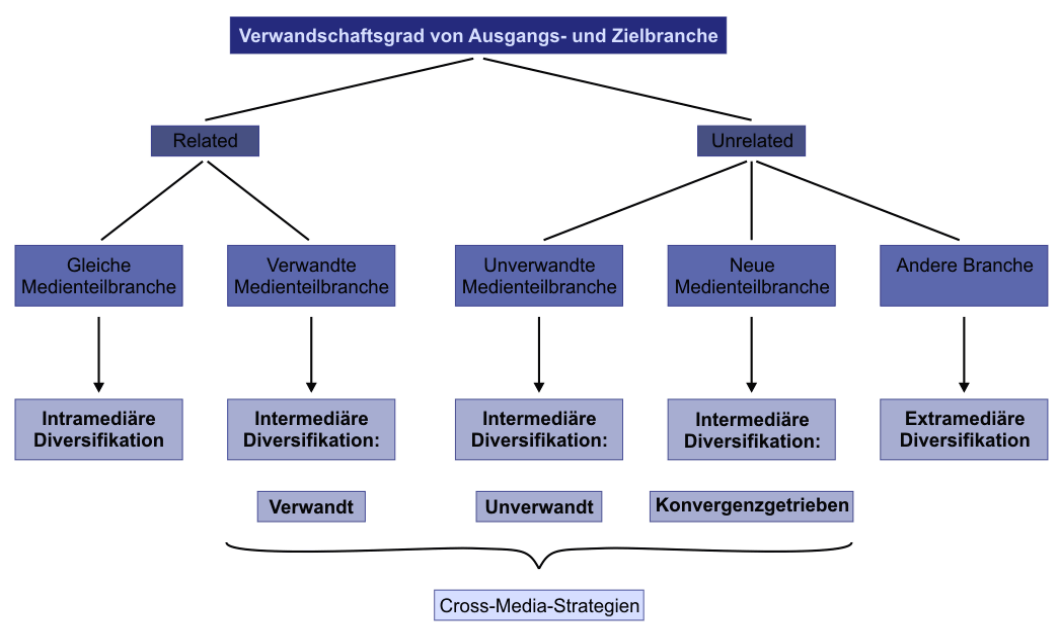

ABBILDUNG I-17: DIVERSIFIKATIONSSTRATEGIEN IN MEDIENUNTERNEHMEN (NACH [MÜL02 S. 6])

Die konvergenzgetriebene intermediäre Diversifikation bezeichnet schließlich den Eintritt von Medienunternehmen in die neuen Medienmärkte Internet und Mobile Kommunikation. Diese Diversifikationsoption ist konvergenzgetrieben, da sie erst im Zuge der technologischen Konvergenz für die Medienunternehmen verfügbar wurde. Beispiele bilden die Websites von Verlagen oder Web 2.0-Aktivitäten von Fernsehsendern. Für potenzielle Nutzer der in dieser Arbeit erstellten Ergebnisse, mit der Absicht Crossmedialer Aktivitäten, ist dieses die häufigste Art der Diversifikationsstrategie.

Extramediäre Diversifikation: Eine extramediäre Diversifikation liegt schließlich vor, wenn ein Medienunternehmen außerhalb von Medien- und Konvergenzbranchen aktiv wird. Beispielsweise die Beteiligung an Gastronomiebetrieben durch die Verlagsgruppe Milchstraße („Fit for Fun Restaurant“).

(Vgl. [Mül02], [Hei01], [Gei02])

Diversifikation folgt häufig dem Ziel, neue Märkte zu erobern und das eigene Wachstum zu sichern. Für Medienunternehmen sind Diversifikationsstrategien jedoch eine Notwendigkeit, um auf das geänderte Mediennutzungsverhalten der Nutzer zu reagieren und bestehende Marktanteile zu sichern. Die dabei zwangsläufig steigenden Kosten erfordern eine genaue Analyse der Strategien und geeignete Maßnahmen, um den Anstieg bestmöglich zu dämpfen.

Insbesondere Zeitungs- und Zeitschriftenverlage unterliegen einer im Vergleich zu anderen Branchen einzigartigen Kosten- und Erlösstruktur.

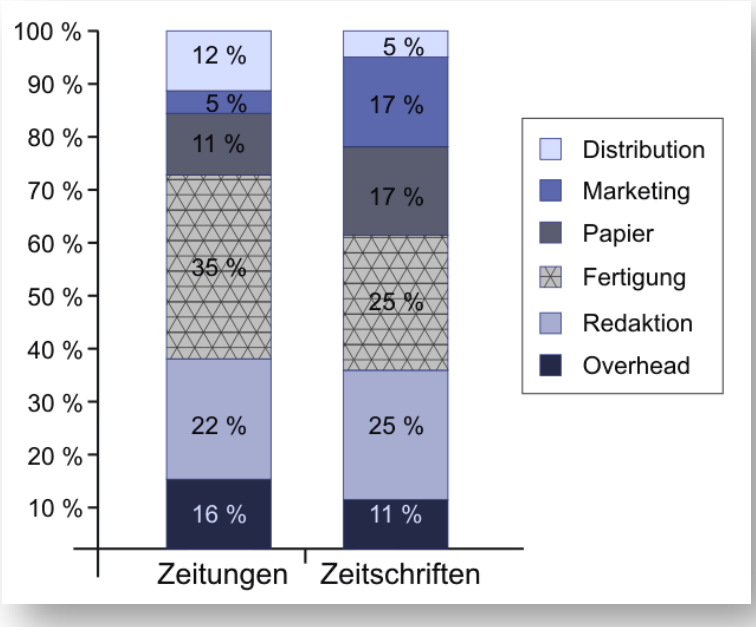

ABBILDUNG I-18: CHARAKTERISTISCHE PRODUKTIONSKOSTENSTRUKTUR VON ZEITUNGS- UND ZEITSCHRIFTENVERLAGEN (VGL. [KEU03])

First-Copy-Cost-Charakter

Jede Produktion eines Informationsgutes ist eine „Blaupausen-Produktion“. Unabhängig von der späteren Anzahl der Rezipienten muss die Redaktion für jede Ausgabe zunächst einen Prototyp konzipieren. Daraus resultiert das bestimmende ökonomische Grundproblem des Verlags in Form einer überdurchschnittlich hohen Fixkostenintensität. In Zeitschriftenverlagen entfallen ca. 65 Prozent aller Kosten auf Fixkosten (first copy costs). Einen etwas geringeren, im Branchenvergleich aber dennoch hohen Fixkostenanteil von 50 Prozent finden sich im Zeitungsverlag.[Sch06]

Abbildung I-18: Charakteristische Produktionskostenstruktur von Zeitungs- und Zeitschriftenverlagen (Vgl.) zeigt die zwei typischen Produktionskostenstrukturen von Zeitungs- und Zeitschriftenverlagen. Marketing- und Papierkosten sind bei der Zeitschriftenherstellung deutlich höher, wohingegen insbesondere im Tageszeitungsbereich höhere Beiträge für ein meist täglich in Anspruch genommenes Distributionssystem aufgewendet werden. Augenfällig an dem hohen Fixkostenanteil ist der Personalkostenanteil von ca. 40 Prozent, der damit deutlich über dem gesamtwirtschaftlichen Durchschnitt von ca. 20 Prozent liegt. Ursache ist die personelle Mindestausstattung in den Redaktionen, ohne die der Prototyp einer Ausgabe überhaupt nicht erst entstehen könnte.[Keu03]

Aus dieser hohen Personalkostenintensität resultiert das grundlegende Paradigma des Cross Media Publishing: Mehrfachverwendung ohne Mehrkosten (siehe S. 36). Diversifikationsstrategien über Medien- und Objektgrenzen hinweg können unter Kostenaspekten nur funktionieren, wenn die resultierenden Verarbeitungsprozesse für Inhalte weitgehend automatisiert werden können.

Teil II

Aufgabenstellung

Identifikation der Prozesse und Formalisierung der Aufgabenstellung

Kapitel 4

CROSS MEDIA PUBLISHING

Jetzt wächst zusammen, was zusammen gehört.

(Willy Brandt, Berlin, 10. November 1989)

In Kapitel 3 wurde der ökonomisch-strategische Hintergrund aufgezeigt, der das Thema „Cross Media Publishing" für Medienunternehmen zu einem wichtigen Punkt in der eigenen Weiterentwicklung gemacht hat. Während der Literaturrecherchen zu dieser Arbeit hat sich gezeigt, dass bereits zahlreiche wirtschaftswissenschaftliche Publikationen und Forschungsergebnisse auf diesem Gebiet existieren, die Frage nach einer informationstechnischen Definition aber eher selten und abstrakt beantwortet wird. Im Folgenden sind zwei Definitionen von „Cross Media Publishing" aufgeführt:

Definition Cross Media Publishing (nach Fritsche)

> *Seit ein paar Jahren macht der Begriff Cross Media Publishing (CMP) in der Medienwelt von sich Reden. Man versteht darunter vereinfacht gesagt ein erweitertes Database Publishing, das es ermöglicht, die Inhalte einer (Medien-)Datenbank wahlweise auf Print- und/oder digitalen Medien auszugeben. […] Konsequent umgesetzt, wird CMP zum Dreh- und Angelpunkt der gesamten Unternehmenskommunikation und zu einem der wichtigsten Instrumente für kundenorientiertes Marketing.*

Vgl. [Fri01]

Definition Cross Media Publishing (nach Becker/Bramann)

> *Cross Media Publishing: Die Veröffentlichung von Texten, Bildern oder Datenbankinhalten in unterschiedlichen Medien, z.B. in Printform, als Online-Medium oder als elektronisches Offline-Medium. Eine ökonomisch sinnvolle Mehrfachverwertung setzt voraus, dass die Daten bei der Erfassung medienneutral strukturiert werden.*

Vgl. [Bec02]

Einigkeit herrscht im Wesentlichen darüber, dass Daten aus einer Quelle auf verschiedene Medien beziehungsweise Kanäle ausgegeben werden sollen: *Single Source - Multi Channel Publishing*. Damit das gelingt, ist die Speicherung in einem ‚medienneutralen' Format notwendig. Eine genauere Definition dessen, was ein Persistenzformat erfüllen muss, um medienneutral zu sein, ist bisher nicht erfolgt. Sie dürfte auch nur kurzzeitig Bestand haben, da das gesamte Forschungsgebiet der medienübergreifenden Publikation enorm schnellen Entwicklungen und Veränderungen unterworfen ist. Ein Beispiel dafür ist das Thema „2D-Code" zur Verknüpfung von Printobjekten mit Online-Ressourcen, auf das am Ende dieses Kapitels noch einmal genauer eingegangen wird.

Neben der grundsätzlichen Idee des Single Source – Multi Channel Publishing herrscht auch weitgehend Einigkeit über eine XML-basierte Auszeichnungssprache als Formatbasis für die Übertragung und Transformation von Daten zwischen beteiligten Systemen. Ob es aber auch sinnvoll ist, die dauerhafte Ablage in einem XML-Format oder in einer XML-Datenbank zu realisieren, ist allerdings umstritten. [Gor06]

Man kann das zentrale Ziel der medienneutralen Datenhaltung vereinfacht unter folgender Aussage zusammenfassen:

Mehrfachverwendung ohne Mehrkosten

Inhalte, die also automatisiert für die Ausgabe auf verschiedenen Medien und/oder verschiedenen Objekten[21] transformiert und bereit gestellt werden können, sind per definitionem medienneutral. Für Bildmaterial ist diese Forderung heute weitgehend erfüllt. Wenn Bilder in der besten verfügbaren Auflösung gespeichert werden, kann man automatisiert in die passende Auflösung für Print, Online etc. umwandeln.[22] Die Transformation in das Ausgabeme-

[21] Mit Objekten sind hier und im weiteren Verlauf der Arbeit Gruppierungen von Inhalten gemeint, die unter einem Titel adressiert werden können. Also beispielsweise verschiedenen Zeitschriften als Print-Objekte oder verschiedene Websites als Online-Objekte.

[22] Durch unterschiedliche Farbräume bei Medienbrüchen oder durch Anpassungen an verschiedene Druckmedien kann manueller oder teilautomatisierter Zusatzaufwand für Bilder entstehen.

dium gestaltet sich in Form der Adaption der Bildgröße als vergleichsweise einfach. Deutlich komplexer wird es bei Texten. Was entspricht hier „der besten Auflösung“?

Die strukturierte Publikation großer Datenbestände mit Hilfe von SGML und TeX reicht ca. 30 Jahre zurück. Es basiert auf zwei Grundannahmen, je eine zum Content und zum Bearbeitungsverfahren. Erstens behandelt es Struktur und Gestalt als getrennte Bestandteile einer Publikation, desweiteren müssen Struktur- und Layoutinformationen sowohl im Publikationsprozess als auch bei der Speicherung der Daten streng getrennt werden. Zweitens sollte die Prozesskette folgende klar unterscheidbare Zustände durchlaufen: Content wird erst beschafft, dann inhaltlich und strukturell veredelt, dann gestaltet und schließlich publiziert und distributiert.[Rot03]

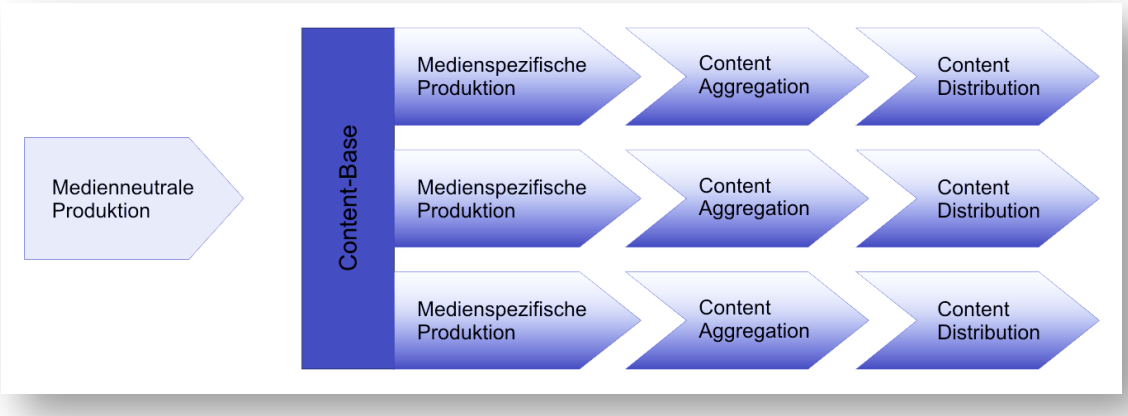

ABBILDUNG II-1: CONTENT-WERTSCHÖPFUNGSKETTE BEIM CROSS MEDIA PUBLISHING (NACH STAMER [MÜL02 S. 94])

Das Konzept der Trennung von Semantik[23] und Format ist also eine wichtige Voraussetzung für die medienneutrale Datenhaltung von Texten. Doch was ist mit typographischen Metadaten? Aufgrund der Inkompatibilität der beteiligten Systeme bei der Interpretation der Metadaten werden im klassischen SGML/XML-Publishing Formatierungsinformationen strikt weggelassen. Da aber Attribute nicht attributierbar sind, kann man bisher nirgendwo in strukturell verbindlicher Form sagen, dass beispielsweise eine Absatzdefinition mit `<p></p>` der Gestaltung dient und nicht der Strukturierung.

In der Regel bringen Texte aber wichtige mikrotypographische Information mit. Das sind beispielsweise Absatz- und Zeichenstile, Hoch- oder Tiefstellung oder Schriftschnitte wie Fettung. Diese Informationen in den proprietären Formaten der beteiligten DTP-Programme zu speichern wäre falsch. Nicht nur, weil damit der Medienbruch in Richtung Online mit Aufwand verbunden wäre, sondern auch, weil die Wiederverwendung über Print-Objekte hinweg dadurch aufwändiger würde. Der Grund dafür liegt darin, dass die typographischen Informationen für Artikel über Objekte hinweg und auch innerhalb von Objekten variieren.

[23] Mit Semantik ist hier die inhaltliche Struktur im Sinne der Interpretation einzelner Textelemente als Titel, Überschrift, Zwischenzeile, Fließtext, etc. gemeint. Im Gegensatz zum Format eines Dokumentes mit mikro- oder makrotypographischen Definitionen.

Der Titel eines einspaltigen Artikels hat beispielsweise die Schriftgröße „18“ und die Schriftart „Times“ bei Flattersatz im Fließtext. Der gleiche Artikel als zweispaltiger Kasten verwendet, hat dann beispielsweise einen Titel in Schriftgröße „24“, der Schriftart „Times bold“ und einem Fließtext im Blocksatz.

Ein manuelles Anpassen – Artikel für Artikel – schmälert die Wertschöpfung der Wiederverwendung. Bei Sonderheften, die aus zahlreichen Ausgaben zusammengestellt werden, verstärkt sich dieser Effekt noch. Eine alternative Lösung für das automatisierte Cross-Media Publishing ist die Transformation der Metadaten (*Schriftart, Schriftgröße, Satzart*) mit Hilfe eines Programms (Compilers), das – um beim oben angeführten Beispiel zu bleiben – aus einem „Artikel-Einspalter“ einen „Kasten-Zweispalter“ erzeugt.

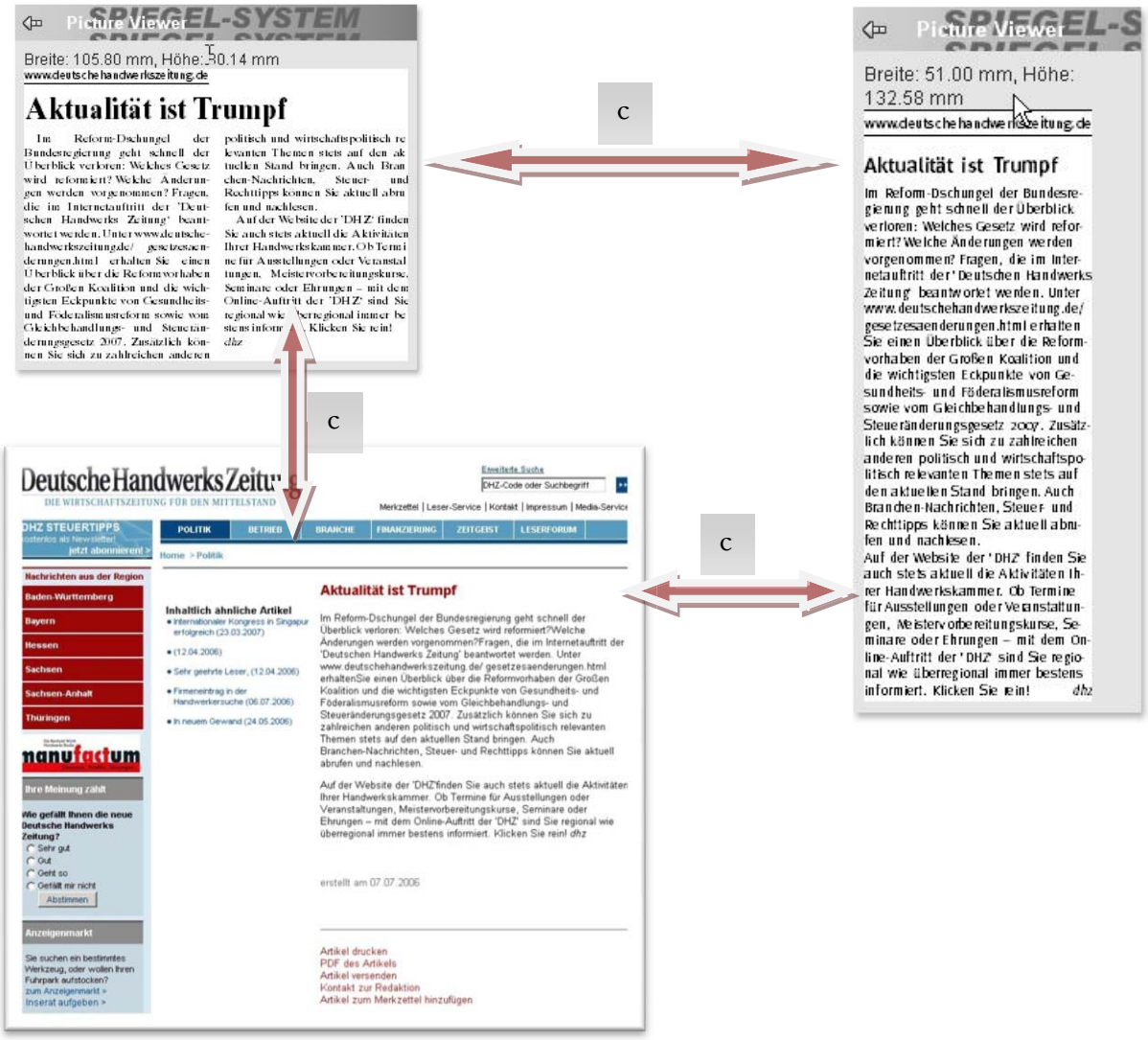

ABBILDUNG II-2: ALGORITHMEN *c* TRANSFORMIEREN DIE TYPOGRAPHISCHEN METADATEN AUTOMATISCH ZWISCHEN DEN VARIANTEN PRINT-1-SPALTIG, PRINT-2-SPALTIG UND ONLINE.

Der Aufwand für die Automatisierung ist aber nicht zu unterschätzen. Mit Hilfe graphentheoretischer Überlegungen ergibt sich die folgende Komplexität in der Umsetzung.

Komplexität der Texttransformationen

Sei $o = |O|$ die Anzahl der Objekte und $p_i :=$ Anzahl der Absatz- und Zeichenstile im Objekt i, dann ist die Summe aller Stile:

$$ps = |PS| = \sum_{i=1}^{o} p_i$$

Desweiteren gilt: Eine Programmiersprache ist ein 4-Tupel $\underline{L} = (L, I, O, sem)$ mit

$L :=$ Menge der zulässigen Programme

$I :=$ Eingabemenge

$O :=$ Ausgabemenge

$sem : L \quad I \rightharpoonup O$ ist eine partielle Abbildung (vgl. [Mer97])

Betrachten wir nun allgemein die Programme zur Transformation c als Compiler, die ein Programm $p \in L_A$ in ein Programm $p' \in L_B$ transformieren (siehe Abbildung II-3[24]).

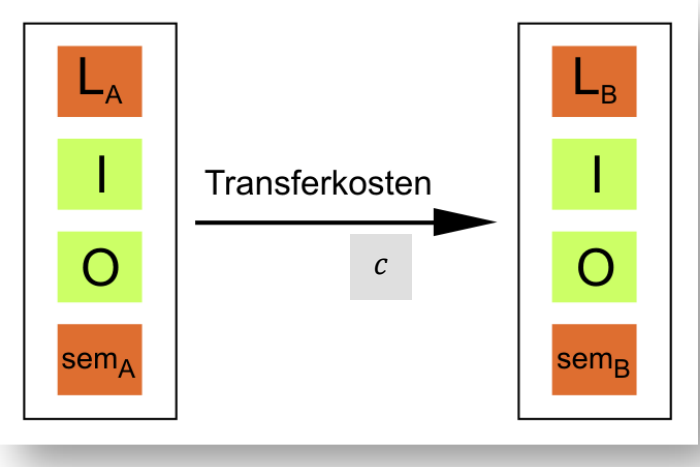

ABBILDUNG II-3: TRANSFERKOSTEN ENTSTEHEN BEI DER ÜBERSETZUNG VON PROGRAMMEN AUS L_A NACH L_B MIT HILFE EINES COMPILERS c.

[24] Transfer- und Nutzungskosten beschreiben hier die Kostenaufwände für die Erstellung der jeweiligen Compiler c. In einem Cross Media Publishing-Gesamtsystem ergäben sich zwar identische Komplexitätsklassen für die Laufzeiten durch die Anwendung der Compiler c, praktisch ist aber selbst bei größten Redaktionen mit zeitgenauer Publikation auszuschließen, dass alle möglichen Übersetzungswege gleichzeitig ausgeführt werden.

Wenn die Abbildung $L_A \xrightarrow{c} L_B$ semantikerhaltend ist, dann gilt $sem_B \circ c = sem_A$, und ein Algorithmus für c heißt ein Übersetzer oder Compiler. Der Compiler (für) c transformiert also jedes Programm $p \in L_A$ in ein Programm $c(p) \in L_B$, so dass gilt:

$$sem_B\ c(p) = sem\ p$$

Nutzungskosten

Betrachtet man nun die Elemente der Menge der Absatz- und Zeichenstile PS als Knoten eines gerichteten Graphen und die zu programmierende Transformation zwischen zwei Objekten $c \in C$ als Kante zwischen diesen Knoten, so ergibt sich im schlechtesten Fall ein vollständiger, gerichteter Graph mit jeweils zwei Kanten zwischen jedem Paar zweier verschiedener Knoten.

Für eine endliche Menge PS mit $|PS| = ps$ und $k \in \mathbb{N}$ gilt: (vgl. [Mer02])

$$|P_k(PS)| = \binom{ps}{k} = \binom{|PS|}{k}, P_k(PS) \text{sind die } k\text{-elementigen Teilmengen von } PS$$

Praktisch sind hier nur Transformationen zwischen zwei Formaten relevant, also ist $k = 2$.

Zusammen mit $\binom{n}{k} \stackrel{\text{def}}{=} \begin{cases} \frac{n!}{k!(n-k)!}, & 0 \le k \le n \\ 0, & 0 \le n < k \end{cases}$

folgt daraus die Komplexitätsklasse der Formattransformationen:

$$ft = |FT| = \binom{ps}{2} * 2 = \frac{ps^2 - ps}{2} * 2 = ps^2 - ps$$
$$\Rightarrow ft \in O(ps^2)$$

Die Anzahl der zu programmierenden Transformationen wächst also im ungünstigsten Fall quadratisch mit der Anzahl der Stile über alle Objekte. Erfahrungen aus der Praxis haben gezeigt, dass dies ein signifikantes Problem ist. Beispielsweise gibt es in Verlagen Vorgaben an die Autoren und Layouter, nur Absatzstile aus einer vorgegebenen Menge zu verwenden, um den individuellen Wildwuchs an Formatierungen zu vermeiden und den Objekten des Verlages ein einheitliches Erscheinungsbild zu geben. Die Anzahl der vorgegebenen Stile kann dabei leicht >100 sein. Eine vollständige Cross-Media-Strategie kann also zu einer nicht mehr praktikablen Anzahl von zu programmierenden Transformationen führen. Damit hätte man die Kosteneinsparung als Treiber für Cross-Media konterkariert.

Wie in Kapitel 3 aufgezeigt wurde, ist die erstmalige Erstellung von Inhalten ein beträchtlicher Kostenfaktor in Medienunternehmen. Das Ziel muss es also sein, Content mehrfach zu verwerten. Führt die Mehrfachverwendung dazu, dass Inhalte in Objekten mit unterschiedlicher Geometrie und Typographie platziert werden, so entstehen Nutzungskosten (Platzierungskosten). Ein wesentliches Ziel dieser Arbeit war es, diese Nutzungskosten zu senken.

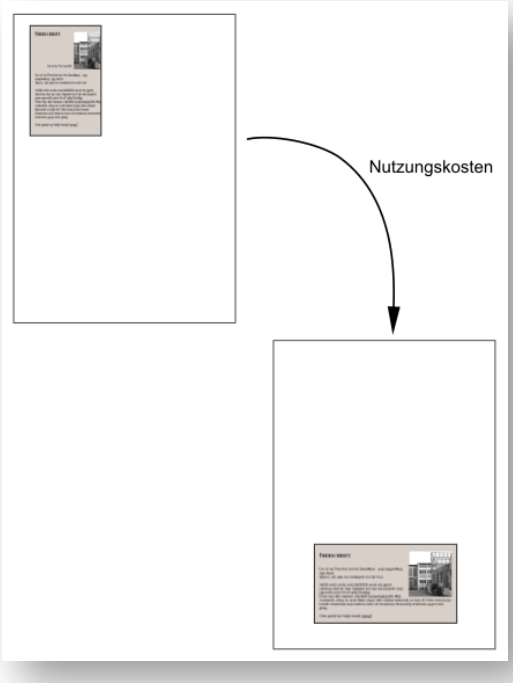

ABBILDUNG II-4: NUTZUNGSKOSTEN (=PLATZIERUNGSKOSTEN) ENTSTEHEN BEI FORMATÄNDERUNGEN DURCH UMPLATZIEREN VON CONTENT-ELEMENTEN.

Definition Nutzungskosten

> *Um Content zu nutzen, muss er in einem Ausgabemedium platziert werden. Die Platzierung entscheidet über die Geometrie und über die Typographie der Inhalte. Folgen aus der Platzierung direkt Änderungen an der Typographie oder führen Änderungen an der Geometrie zu notwendigen Anpassungen an der Typographie des Contents, so sind die Kosten für die Anpassungen der Content-Typographie Nutzungskosten.*

Transferkosten

Nach dem allgemeinen Verständnis bedeutet Cross Media Publishing in erster Linie die Ausgabe auf unterschiedlichen Medientypen. Bisherige XML-basierte Ansätze verfolgen dabei den Weg, eine attributier- und steuerbare Zentraleinheit für die Publikation in die verschiedenen Medien zu realisieren. In der praktischen industriellen Anwendung ist dieser Weg aber nicht gangbar. In den Medienunternehmen finden Anbieter von Cross Media Publishing-Systemen medienspezifische Standardsysteme vor, beispielsweise CMS für die Online-Publikation und DTP für die Print-Produktion. Insbesondere der DTP-Bereich wird derzeit von zwei Systemen dominiert: XPress von der Firma Quark und InDesign von Adobe[25]. Ein Ersatz dieser Systeme durch ein zentrales Cross Media Publishing-System erscheint mittelfristig nicht erreichbar. Neben dem Standardargument der Kosten für Mitarbeiterschulungen und Prozessänderungen scheint es insbesondere derzeit nicht realistisch, dass irgendein Software-Unternehmen in der Lage wäre, die sehr umfangreichen Funktionen dieser Systeme in einem eigenen Cross Media Publishing-System zu implementieren. Außer in kleineren Projekten, wo nur rudimentäre Ansprüche an die Prozesskontrolle und die Qualität der Ausgabe gestellt werden, muss ein solches System also in der Lage sein, mit bestehenden Standardsystemen Daten unidirektional oder bidirektional auszutauschen. Für Medienunternehmen ist es dabei umso wichtiger, dass Metadaten ohne Verlust der Semantik ebenfalls ausgetauscht beziehungsweise übertragen werden können, desto mehr Aufwand in die Erstellung dieser Metadaten investiert wurde. Distribution und Publishing muss also weiterhin über die etablierten Systeme erfolgen.

[25] Es existieren noch einige weitere DTP-Anwendungen wie Microsoft Publisher, Ragtime, Macromedia Freehand uvm., die im industriellen Einsatz aber praktisch bedeutungslos sind.

Oberhalb der Ebene der Nutzungskosten, die Aufwände bei der Platzierung innerhalb einer Medienkategorie beinhalten, bedeutet die Transformation der Inhalte zwischen den beteiligten Systemen die Entstehung von Transferkosten.

Seien nun die beteiligten Software-Systeme (CMS, DTP, etc.[26]) Elemente einer Menge SA und betrachten wir die Systeme $sa \in SA$ als Knoten eines (gerichteten[27]) Graphen und die zu programmierende Transformation zwischen zwei Software-Systemen $c \in C$ als Kante zwischen diesen Knoten, so ergibt sich in dem Fall, dass kein zentrales Cross Media Publishing-System in der Lage ist, zwischen den bestehenden Formaten zu konvertieren, wiederum ein vollständiger gerichteter Graph.

Wie schon bei den Nutzungskosten folgt daraus die Komplexitätsklasse der Systemtransformationen:

$$st = |ST| = \binom{sa}{2} * 2 = \frac{sa^2 - sa}{2} * 2 = sa^2 - sa$$
$$\Rightarrow st \in O(sa^2)$$

Die Anzahl der zu programmierenden Transformationen wächst also auch hier im ungünstigsten Fall quadratisch mit der Anzahl der beteiligten Systeme. Eine Systemtransformation zu programmieren, ist nach bisherigen Erkenntnissen aber mit großem Aufwand verbunden (siehe Teil III). Die Lösung liegt in einem zentralen System, über das alle Content-Übertragungen und –Transformationen abgewickelt werden.

Definition Transferkosten

> *Transferkosten entstehen, wenn Content und zugehörige Metadaten von einem Quellsystem A in ein Zielsystem B transformiert werden müssen. Die Transformation kann für jede Content-Instanz manuell erfolgen oder automatisiert durch eine programmierte Transformation von A nach B. Die Transferkosten erhöhen sich, wenn die Transformation ohne Verlust der Semantik erfolgen soll.*

Aus dem Druck des diversifizierten Nutzerverhaltens und der Konvergenz der Medien entstehen aktuell laufend neue Verfahren, welche die Reduktion der Transfer- und Nutzungskosten bei der crossmedialen Verarbeitung von Inhalten adressieren. Alleine während der

[26] Praktisch ist der Datenaustausch mit weiteren Systemen wie Anzeigenmanagement, CRM, Farbraumprüfung etc. notwendig.

[27] Bei einer unidirektionalen Kommunikation ist der Graph ungerichtet, bei der bidirektionalen Kommunikation gerichtet.

Erstellung dieser Arbeit erschienen wochen- bzw. monatsweise Meldungen über ‚neue' Cross Media Publishing-Funktionalitäten. Ziele, Wirkungsweise und Ansätze dieser Verfahren sind dabei sehr unterschiedlich, so dass der Begriff „Cross Media Publishing" alleine keine Einordnung über die jeweiligen Möglichkeiten erlaubt.

Im Folgenden wird versucht, die Ansätze bestehender Cross Media Publishing-Verfahren in drei Stufen zu klassifizieren. Dabei ist eine höhere Stufe nicht per se ‚besser' als eine niedrigerere. Je nach Anwendungsfall kann ein Ansatz aus einer kleineren Stufe sinnvoller sein, als eine aufwändigere Lösung, deren Mehrleistung nicht in Mehrwert umgesetzt wird. Zudem kann diese Klassifikation keinen Anspruch auf Vollständigkeit und Dauerhaftigkeit erheben, da es sich wie schon angesprochen um einen aktuell sehr dynamischen Technologiebereich handelt.

Vorab sollen noch zwei Begriffe definiert werden, welche die Prozessabläufe in den Phasen der Erstellung der Inhalte bis zur Imprimatur[28] charakterisieren. Im Zusammenspiel der Redakteure, welche für die Inhalte verantwortlich sind und der Grafiker, welche das Layout erstellen, ist die Grenze der kreativen Prozesse neben der Definition von Verantwortlichkeiten auch von den Möglichkeiten der eingesetzten Werkzeuge abhängig.

Content vor Layout

Der Redakteur erstellt seine Inhalte zunächst unabhängig vom späteren Layout der Druckausgabe. Er nutzt zum Erfassen von Texten klassische Werkzeuge wie Textverarbeitungsprogramme, Editoren oder Redaktionssysteme mit eigener Eingabeoberfläche. Nach der redaktionellen Freigabe übernimmt der Grafiker die Texte und evtl. beigefügte Bilder in sein Layoutsystem, üblicherweise ein DTP-Programm. Er ‚setzt' die Inhalte in zu druckenden Seiten ein und achtet auf das korrekte Layout.

Das Layout passt sich dem Content an. Dieses Verfahren kommt üblicherweise bei Fachzeitschriften, Magazinen, Fachliteratur etc. zum Einsatz und ist das Standardverfahren bei crossmedialer Produktion, da Inhalte und Metadaten gemeinsam in einer medienneutralen Oberfläche erfasst werden können. Ist der Platz nicht ausreichend oder der Text zu kurz, kann der Workflow wieder zum Redakteur laufen, der einen Probeausdruck mit Anweisungen erhält. In modernen Redaktionssystemen kann der Auftrag auch automatisiert elektronisch erfolgen. Da der Redakteur seine Inhalte nun an das Layout anpasst, befinden wir uns in einem nicht abgrenzbaren Bereich zum umgekehrten Prozessablauf, der im Folgenden als „Layout vor Content" erläutert und definiert wird.

[28] Imprimatur ist die Druckerlaubnis. Sie wird nach dem Korrekturlesen und Prüfen des Probeausdrucks erteilt. Die Imprimatur war ursprünglich der Segen des Papstes bzw. der oberen Kirchenleitung zu Drucksachen, in denen es auch nur im entferntesten um Kirche, Religion und Glauben ging.

Definition Content vor Layout

> *Seitenproduktionsverfahren, bei dem Inhalte zunächst ohne Rücksicht auf das Layout eines Ausgabemediums oder Objekts erfasst werden. Das Layout der Zielobjekte passt sich automatisch durch geeignete Anwendungen dem Inhalt an oder wird manuell an den Inhalt angepasst.*

Layout vor Content

In diesem Fall steht das Layout bereits fest, bevor Inhalte wie Texte und Bilder von einem Redakteur erstellt werden. Je nach Produktionsumgebung schreibt der Redakteur in einem autonomen Texteditor genau auf Zeichen- oder Wortlänge oder er schreibt in eine vorgefertigte Maske „auf Zeile“. Bei crossmedialen Produktionsverfahren potenziell problematisch, da entschieden werden muss, welches Medium und welches Objekt die Vorgabe für das Erfassen im Layout sein soll. Moderne DTP-Systeme enthalten Programme, die es dem Redakteur erlauben, direkt in das DTP-Layout einer Seite zu schreiben, ohne jedoch das Layout verändern zu können. Beispiele dafür sind Adobe InCopy oder Quark CopyDesk.

Der Inhalt passt sich dem Layout an. Dieses Verfahren kommt üblicherweise bei Tageszeitungen, Consumer-Magazinen, Werbematerialien etc. zum Einsatz. Ist der Platz nicht ausreichend oder der Text zu kurz, kann der Workflow wieder zum Grafiker zurück laufen, der einen auf beliebigem Weg überbrachten Kommentar des Redakteurs enthält. Da der Grafiker sein Layout nun an die Inhalte anpasst, befinden wir uns in einem nicht abgrenzbaren Bereich zum o.a. umgekehrten Prozessablauf.

Definition Layout vor Content

> *Seitenproduktionsverfahren, bei dem das Layout der fertigen Seite bereits vor dem Erfassen der Inhalte festgelegt ist. Inhalte werden spezifisch für ein Zielobjekt so erfasst, dass sie das Layout für dieses Objekt möglichst gut ausfüllen.*

Cross Media Publishing Level 1

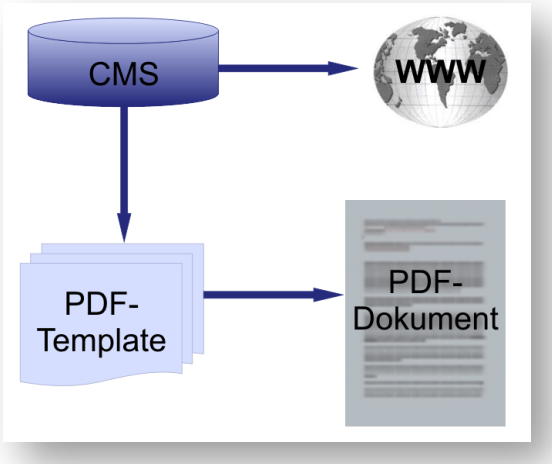

ABBILDUNG II-5: CROSS MEDIA PUBLISHING - LEVEL 1

Level 1 beschreibt Lösungen für einfaches Cross Media Publishing, wie sie heute bereits häufig zu finden sind. Bestehende Lösungen[29], die Internetseiten produzieren und verwalten können, werden so erweitert, dass sie die Inhalte in gestaltete PDF-Dokumente ausgeben können. Dabei kommen häufig Tools von Drittanbietern oder Lösungen auf XSL-Basis zum Einsatz.

Merkmale:

- Single Source - Multi Channel Publishing
- Existierende Content Management Systeme exportieren ihre strukturierten Inhalte in dedizierte PDF-Vorlagen (Templates).
- Transfer häufig über XSL oder über Erweiterungen zur direkten Erzeugung von PDF.
- Formatierungen und Typographieangaben per XSLT oder direkt im PDF-Template.
- Content Life Cycle:

[29] Es ist natürlich auch der umgekehrte Fall denkbar, dass vorhandene Printsysteme in Richtung Online-Publikation erweitert werden. Tatsächlich gibt es diverse Ansätze. Entweder wird dabei nur ein dediziertes Inhaltselement eines Online-Auftritts produziert, beispielsweise E-Paper, blätterbares Flash, etc. und damit auf die Vorteile und Eigenheiten des Mediums Internet verzichtet. Oder es wird aufgrund der funktionalen Komplexität (siehe Kap. Content Management Systeme) auf eine eigene Entwicklung verzichtet und die Integration professioneller Content Management Systeme angestrebt, was prinzipiell auf Cross Media Publishing Level 3 hinausläuft.

- Content-Erstellung im CMS →
- Vorschau der Druckausgabe über PDF →
- Korrekturen am Content im CMS →
- Vorschau über PDF →
- Druckausgabe des PDF

- Alternativ: Etablierte Print-Redaktionssysteme exportieren Inhalte in HTML-Templates.
- Content Life Cycle:
  - Content-Erstellung im PRS →
  - Vorschau im Browser (HTML) →
  - Korrekturen am Content im PRS →
  - Vorschau HTML →
  - Freigabe und Veröffentlichung HTML

Cross Media Publishing Level 2

Als Erweiterung zu Level 1 ist hier vor allem die applikative oder prozessuale Integration bestehender Systeme zu sehen, welche Daten verwalten und Prozesse steuern, die im Kontext von Publikationen in größeren Unternehmen verwendet werden. Ein weiterer wichtiger Unterschied ist die Art des Medienbruchs. Er erfolgt in der Art, dass die Daten in einem offenen Format ins Zielformat konvertiert werden, so dass diese anschließend weiter bearbeitet werden können.

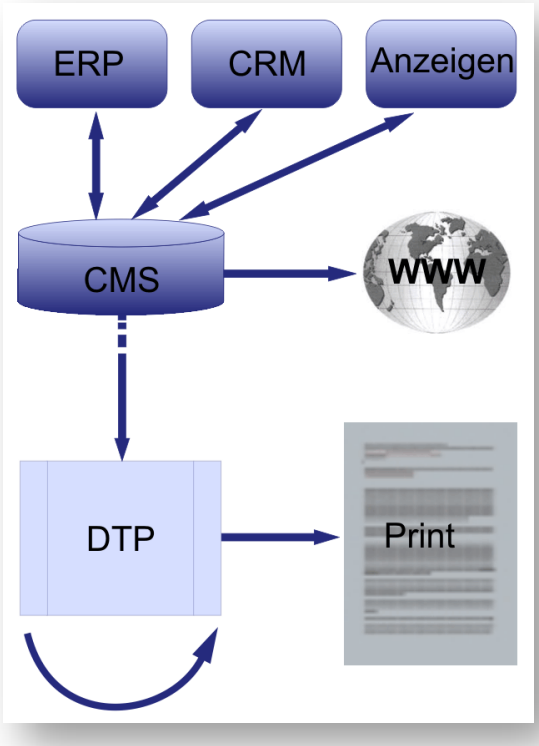

ABBILDUNG II-6: CROSS MEDIA PUBLISHING LEVEL 2

Merkmale:

- Single Source - Multi Channel Publishing
- Etablierte Content Management Systeme mit Workflow- & Rechtemanagement
- Export der Inhalte über Plugins/XML/eigene Schnittstellen an DTP- oder Print-Redaktionssysteme (eigene Lösungen oder Drittanbieter).
- Formatierungen und Typographieangaben im DTP bzw. PRS
- Content Life Cycle:
  - Content-Erstellung im CMS →
  - Export nach DTP →
  - Vorschau der Druckausgabe über DTP-Preview oder PDF →
  - Korrekturen am Content im DTP →
  - Druckausgabe aus DTP
- Anbindung ERP (Kataloge, PIM)
- Anbindung CRM und Anzeigenverwaltung (Zeitschriften, Zeitungen)

Cross Media Publishing Level 3

Neben der grundsätzlichen Verarbeitung der typographischen Metadaten wurde zu Beginn der Forschungsarbeiten ein weiteres übergreifendes Ziel definiert: Aus allen Zielmedien, die im klassischen Prozesskontext als Quellmedien verwendet werden, müssen vom Benutzer durchgeführte Änderungen an Inhalten und an Metadaten in das zentrale, medienneutrale Format zurückgeführt werden können. In Kapitel 6 (S. 53ff) werden die Motive näher erläutert. Die in Abbildung II-8 skizzierte Transformationseinheit musste also bidirektional ausgelegt werden, vergleichbar mit einem Compiler, der Programmtexte in beide Richtungen übersetzen kann. Wie sich im Laufe der Arbeiten herausstellte, fügte dieses Ziel dem Projekt eine eigene Komplexitätsebene hinzu. Neben den oben beschriebenen Architekturansätzen folgt daraus für das Gesamtsystem eine neue Ebene der medienneutralen Informationsverarbeitung, definiert als Cross Media Publishing Level 3:

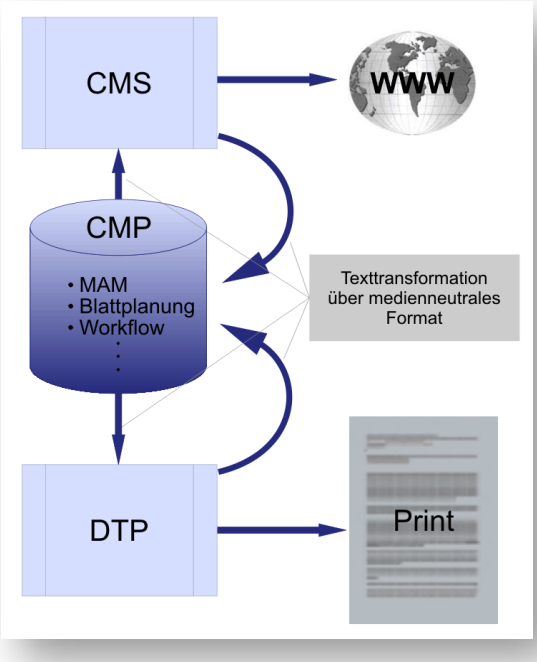

ABBILDUNG II-7: CROSS MEDIA PUBLISHING LEVEL 3

Merkmale:

- Multi Source - Multi Channel Publishing
- Dezidierte Cross Media Publishing-Systeme
- Bidirektionale Kommunikation mit CMS, DTP, Offline-Editoren etc.
- Medienneutrale Speicherung formatierter Texte mit Trennung von Inhalt, Struktur und Typographie
- Formatierungen und Typographieangaben im DTP, CMS oder im zentralen System
- Verlustlose Transformation aller Metadaten ohne Veränderung der Semantik
- Content Life Cycle:
  - Content-Erstellung im CMP, CMS, DTP oder in Offline-Editoren →
  - beliebiger Import und Export zwischen den beteiligten Systemen →
  - Vorschau aller Medienspezifischen Ausgabe im zentralen CMP-System oder separat im jeweiligen externen System →
  - Korrekturen am Content an beliebiger Stelle →
  - Druckausgabe aus DTP, Online-Ausgabe über CMS

- Schnittstelle MAM/integriertes MAM
- Medienabhängige Workflows
- Blattplanung, Themenplanung, Anzeigenplanung
- Anbindung ERP (Kataloge, PIM)
- Anbindung CRM und Anzeigenverwaltung (Zeitschriften, Zeitungen)

Kapitel 5

DESKTOP PUBLISHING UND TYPOGRAPHIE

Das Desktop Publishing ist entstanden als eine konzertante Aktion von Genies, Gurus und Gauklern, gewachsen auf dem Humus des Silicon Valley. (Joachim Peters)

Nach der kurzen historischen und allgemeinen Einführung zum Thema DTP auf Seite 10, geht es in diesem Kapitel um die spezifischen typographischen Besonderheiten des klassischen Mediums Print im Vergleich zu aktuellen Online- und Offline-Medien wie Internet, DVD, Mobile Geräte, etc., die im allgemeinen unter dem Begriff „Neue Medien" zusammengefasst werden.

Eine gedruckte Fassung stellt höhere Anforderungen an die formalen Strukturen und typografischen Auszeichnungen, als eine Online-Version. Es war ein wichtiges Ziel dieser Arbeit, diese unterschiedliche Mächtigkeit der Sprachklassen ohne Verlust der Auszeichnungs-Semantik zu realisieren.

Typographie

Seit dem Ende des materiellen Schriftsatzes existiert keine allgemein eindeutige Definition für Typographie mehr. Typographie wird heute nicht mehr alleine mit dem Buchdruck in Verbindung gebracht, sondern mit dem materiell und digital, also durch einen mathematischen Algorithmus kodiertem, reproduzierbarem Schriftbild als solchem.[30]

Zur allgemeinen Anforderung in dieser Arbeit gehört es, formatierten Text zu speichern, zu konvertieren und zu publizieren. Text, also eine Aneinanderreihung von Buchstaben und Zeichen, zu formatieren, ist Aufgabe der Typographie. Man kann sagen, Text wird von typographischen Funktionen verändert. Diese Funktionen sind bijektiv und weisen Mengen von Zeichen[31] Attribute zu. Die Textmenge im Bildbereich ist identisch mit der des Definitionsbereiches nur mit hinzugefügten, veränderten oder gelöschten Attributen. Eine genauere Betrachtung folgt in Kapitel 14. [Bei07] definiert Typographie als visuelle Gestaltung eines Druckerzeugnisses, einer Multi-Media-Arbeit oder einer dreidimensionalen Oberfläche in der Art, dass Inhalt und Schrift sowie die Anordnung von Text und Bild ein optisch und didak-

[30] Eine Übersicht der theoretischen als auch praktischen Disziplinen der Typographie sowie unterschiedliche kulturtheoretische und gestalterische Betrachtungsweisen findet sich auf [Bei07]

[31] Zeichen, Wörter, Sätze, Absätze, etc.

tisch befriedigendes Ganzes ergeben. Im weiteren Verlauf dieser Arbeit wird Typographie als Gestaltung mittels Schrift, Linien, Flächen und typographischem Raum[32] bezeichnet.

Dem Redakteur bzw. Anwender bieten sich im Desktop Publishing eine ganze Reihe Gestaltungsmöglichkeiten mit Hilfe typographischer Funktionen. Die Mächtigkeit dieser Funktionen im Desktop Publishing übersteigt im Allgemeinen die Mächtigkeit der bei der Ausgabe von HTML anwendbaren typographischen Funktionen.[33]

Zu den wichtigsten typographischen Funktionen gehören:

- Schriftfamilie, Schriftart
- Schriftgröße (auch „Schriftgrad")
- Auszeichnungsart[34]
- Zeilenlänge (auch „Satzbreite")
- Zeilenfall (auch „Schriftsatzart")
- Satzspiegel

Hinzu kommen dann noch die Anordnung von Bildern und Grafiken, das Benutzen von Linien und Kästen, sowie der Einsatz von Farben.

Die gestalterische Typographie wird in Mikrotypographie und Makrotypographie segmentiert. Die Mikrotypographie (auch Detailtypographie) beschreibt die Schriftgestaltung eines Satzes und der Zeichen[35] selbst, den Zeichen-, Wort- und Zeilenabständen, Trennungen und Laufweiten.

Die Makrotypographie hingegen beschäftigt sich mit dem optischen Gesamtkomplex einer Setzarbeit, beginnend mit der Wahl des Trägermaterials, der Herstellungsverfahren, bis zum Format, dem Seitenaufbau, der Auswahl der Schrift und des Schriftstils, der Anordnung der einzelnen Textboxen, Grafiken und dem Verhältnis der Elemente zueinander auf einer Seite.

Im DTP werden Texte und Grafiken mit Hilfe von umgebenden Boxen auf einer Seite[36] platziert, eine Seite kann dabei mit einer oder mit mehreren Boxen belegt werden. Diese makrotypographische Formatierungstätigkeit ist eine Objektspezifische Aufgabe, die entweder in einem zentralen System (Cross Media Publishing Level 1) oder in einem externen DTP-Client ausgeführt werden kann (Cross Media Publishing Level 2 und 3).

---

[32] optisch wirksamer Abstand

[33] Mit dem Einsatz von CSS ab der Version 2 bieten sich dem Anwender vergleichbar mächtige typographische Gestaltungsmöglichkeiten bei der Erstellung von (X)HTML-Seiten. Da die aktuellen Internet-Browser bei der Interpretation zu höchst unterschiedlichen Ergebnissen kommen und keine dem Komfort von DTP-Systemen vergleichbar komfortablen Editoren existieren, beschränkt sich die aktuelle Verwendung hauptsächlich auf die von HTML bekannten Möglichkeiten.

[34] beispielsweise **fett** (laute Auszeichnung), *kursiv* (leise Auszeichnung)

[35] Buchstaben, Ziffern, Sonderzeichen

[36] Textboxen können sich auch über mehrere Seiten erstrecken, der Text kann dann über Seiten hinweg ‚fließen'.

Kapitel 6

LASTEN UND PFLICHTEN

Der ans Ziel getragen wurde, darf nicht glauben, es erreicht zu haben.

(Marie von Ebner-Eschenbach)

Um möglichst effiziente Arbeitsabläufe in den Medienunternehmen zu gewährleisten, muss es das Ziel sein, die in den Unternehmen vorhandenen Systeme so einzubinden, dass die Anwender weiter in ihren gewohnten Oberflächen arbeiten können. Dazu ist eine bidirektionale Kommunikation mit diesen Systemen notwendig, damit Änderungen an den Inhalten in das zentrale Cross Media Publishing-System zurückgeführt werden können und damit wieder allen anderen Ausgabekanälen zur Verfügung stehen. Die Menge der dabei ausgetauschten Daten lässt sich zunächst in vier Klassen einteilen:

1. Von Struktur und Format befreite Textinhalte

2. BLOBS. Grafiken, Multimedia-Dateien, Downloads, etc.

3. Metadaten für die Formatierung von Inhalten

4. Sonstige Metadaten. Beispielsweise strukturelle Metadaten für die Generierung von Inhaltsverzeichnissen, Indizes, Navigationen oder für die Prozesskommunikation innerhalb der relevanten, beteiligten Systeme[37]

Fast alle professionell eingesetzten Publikationssysteme besitzen heute Schnittstellen für den Im- und Export von Daten, entweder nativ oder durch Erweiterungen von Drittanbietern. Als Transportprotokoll kommt mittlerweile fast ausschließlich XML zum Einsatz. Mit Hilfe von XML-Parsern lassen sich Inhalte und Metadaten relativ leicht extrahieren und im Zielsystem weiter verarbeiten. Für den Anbieter eines zentral steuerenden Gesamtsystems liegt die Herausforderung zunächst darin, sich den festgelegten Struktur- bzw. Typdefinitionen per Document Type Definition (DTD) oder XML Schema der externen Systeme anzupassen und für eine korrekte Übersetzung der Syntax zwischen den Formatspezifikationen zu sorgen. Je nach den Gegebenheiten der beteiligten Systeme variiert dabei der zu treibende Aufwand.

Die Interpretation der Semantik ist für Daten der Klassen 1 und 2 nicht erforderlich, da keine medienspezifische Unterscheidung existiert.

Innerhalb der Klasse 4 ist zwischen Medien-abhängiger und Medien-unabhängiger Semantik zu unterscheiden. Die Semantik struktureller Metadaten kann medienspezifisch sein. Zwischen dem Inhaltsverzeichnis einer Print-Publikation und der Navigation auf einer Website

[37] siehe Cross Media Publishing Level 2 und 3 ab S. 40

existieren häufig strukturelle Unterschiede. Metadaten für eine strukturelle Zuordnung eines Artikels oder eines Dokumentes müssen also medienspezifisch interpretiert werden. Auch innerhalb eines Medientyps können strukturelle Unterschiede für identische Dokumente auftreten. Beispielsweise kann die Pressemitteilung eines Unternehmens im Intranet unter einem anderen Navigationspunkt eingehängt werden wie auf der öffentlichen Website des Unternehmens. Für den Inhaltstyp „Pressemitteilungen" müssen die strukturellen Metadaten also medien- und objektspezifisch übersetzt werden. Hier entstehen bei der Übersetzung Kosten für Nutzung und Transfer (siehe S. 40 und S. 42). Andere Metadaten, welche die Struktur innerhalb eines Artikels oder Dokumentes definieren, wie Titel, Einleitung, Text, Bild, etc. können per se medienneutral sein.

Metadaten mit Medien-unabhängiger Semantik sind zum Beispiel Autorenname, Erstellungsdatum, Verlag, Sprache, Kundendaten und so weiter.

Für Daten der Klasse 3 wurde im Rahmen dieser Forschungsarbeit ein Verfahren entworfen und prototypisch umgesetzt, welches die Transformation typographischer Metadaten zwischen einem neutralen, zentralen Format und den Formaten der beteiligten Systeme realisiert. Diese Transformationseinheit soll konzeptionell zwischen einem Cross Media Publishing-System und den externen Anwendungen angeordnet sein (siehe Abbildung II-8).

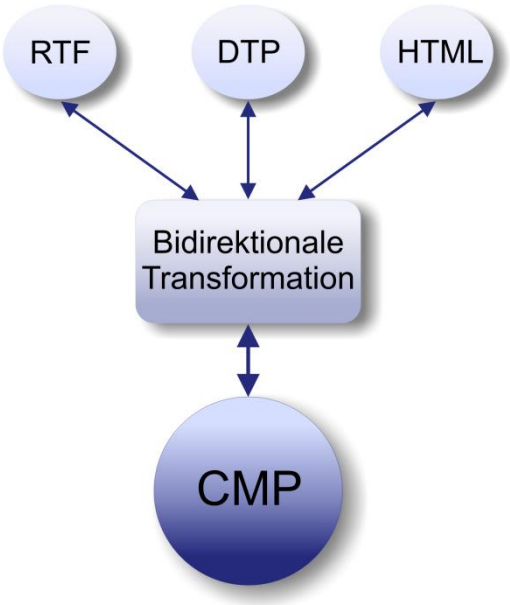

ABBILDUNG II-8 DIE TRANSFORMATIONSEINHEIT SOLL ZWISCHEN DEN ZENTRALEN CROSS MEDIA PUBLISHING-SYSTEM UND DEN EXTERNEN ANWENDUNGEN LIEGEN.

Es sprachen zwei Gründe für den grundlegenden Ansatz, ein eigenes, XML-basiertes und zentrales Format zu entwickeln, so dass sich eine „Hub-Struktur“ für den gesamten crossmedialen Content-Verwaltungsprozess ergibt:

Keines der bestehenden Standardverfahren konnte die gegebenen Kriterien (s.u.) erfüllen. Teil III dieser Arbeit stellt die wichtigsten Verfahren, die im Vorfeld betrachtet wurden, ausführlicherer vor. Für ein effizientes Cross Media Publishing schien XSL-FO das am ehesten geeignete Verfahren zu sein. Nach einer ausführlichen Analyse sprach eine Reihe von Gründen gegen die Anwendung von XSL-FO:

1. Zunächst ist für den industriellen Einsatz die Unterstützung der (wenigen) Standardsysteme für den Medien-spezifischen Publishingprozess essentiell. Um die Komplexität zu begrenzen, sollten zudem nur transformierte ‚Content-Stücke[38]‘ in die bidirektionale Kommunikation einbezogen werden. Die eigentliche Montage eines kompletten Artikels oder Dokumentes sollte weiterhin im Standardsystem erfolgen. Das Konzept der Seitenbeschreibung mit XSL-FO sieht dagegen die Beschreibung des druckfertigen, kompletten Dokumentes vor (siehe Abbildung III-4).

2. Eine Anforderung an das Gesamtsystem bestand darin, dass Content aus beliebigen Quellen in alle beteiligten Medienformate und -kanäle publiziert werden können soll. In diesem Prozess müssen Redakteure, je nach Anzeige der Inhalte im aktuellen Template, häufig Änderungen am Content und an der Typographie vornehmen. Diese Änderungen finden in verschiedenen Anwendungen statt und müssen beim Speichern bzw. Check-In möglichst schnell über das zentrale Format an alle Zielformate verteilt werden. Es zeigte sich, dass die Transformation ganzer Dokumente mit den bekannten XSL-FO-Prozessoren für einen interaktiven Prozess über eine Benutzerschnittstelle deutlich zu langsam war.

3. XSL-FO ist zu mächtig. Um die Komplexität in Cross Media Publishing-Systemen möglichst gering zu halten, werden die meisten makrotypographischen Formatierungen für Zeitschriften, Zeitungen, Kataloge, Berichte, Webseiten, etc. in den jeweiligen Ausgabetemplates festgelegt. Änderungen an diesen Templates finden vergleichsweise selten statt. Es war also sinnvoll, diesen kompletten Bereich nicht mit in das Transformationssystem zu integrieren. Um ein einheitliches Erscheinungsbild zu gewährleisten, arbeiten Verlage mit einem festgelegten Satz von Absatz- und Zeichenstilen, aus denen die Anwender wählen dürfen. Individualformatierungen sind selten erlaubt. Um das System schnell und schlank zu halten, sollten nur Absatz- und Zeichenstile, Sonderzeichen und Code Pages geparst und transformiert werden.

4. Für die Eingabe von Texten im WYSIWYG-Editor im zentralen Cross Media Publishing-System hätte eine weitere Meta-Sprache entwickelt werden müssen, da das Sprachkonzept von XSL-FO für die Eingabe durch den Anwender zu kompliziert ist.

Um die quadratischen Laufzeitklassen der Nutzungs- und Transferkosten zu vermeiden, sollten alle Transformationen über ein zentrales „Hub“-Metaformat laufen.

---

[38] Absätze erwiesen sich als geeignete Content-Stücke. Sie werden in allen Metaformaten durch ein dediziertes Tag geklammert.

Ein weiteres Format, welches zunächst in Frage kam, waren die bereits im Content Management System von InterRed verwendeten *IR-Deftags*[39]. Sie wurden ebenfalls verworfen, da sie eine von der XML abweichende Syntax verwendeten und damit statt dem Rückgriff auf die Vielzahl der existierenden Standardwerkzeuge für XML eigene Werkzeuge hätten entwickelt werden müssen. Außerdem waren mit ihnen keine überlappenden Auszeichnungen im WYSWYIG-Editor möglich (siehe entsprechende Anforderung im Folgenden).

Allgemeine Anforderung an das zentrale XML-Format

Das allgemeine Ziel für die Entwicklung des zentralen XML-Formates lässt sich wie folgt zusammenfassen:

> *Realisierung einer medienneutralen Speicherung textueller Daten*
>
> *ohne Verlust der typographischen Metadaten*

Vor Beginn der Implementationsarbeiten wurde in Gruppensitzungen mit allen Beteiligten versucht, zwingende, optionale und abgrenzende Anforderungen zu formulieren. Neben der abstrakten Zielvorstellung sollten sie die bisherigen Projekterfahrungen der InterRed GmbH reflektieren, Vorschläge und Ideen von First-Adaptor-Kunden berücksichtigen, sowie die Grundsätze einer flexiblen, wartungsarmen und allgemeintauglichen Verfahrensentwicklung beachten.

Konkrete Anforderungen

- Die für das Transformationssystem relevanten typographischen Formatangaben waren alle Definitionen, die einem Textinhalt unabhängig von seiner Platzierung im Ausgabemedium zugewiesen und somit als medienneutral angesehen werden können. Somit sollte die für das Projekt relevante Typographie zu großen Teilen die Mikrotypographie umfassen. Hinzu kamen einige Aspekte der Makrotypographie, so etwa die Angabe von Schriftart- und schnitt[40], Absatzausrichtungen, Absatzlinien oder seitlichen Abständen.
- Besonders zu beachten waren zwei Klassen von Textauszeichnungen: Absatzstile und Zeichenstile. Absatzstile bezeichnen verschiedene typographische Auszeichnungen, die auf einen gesamten Absatz angewendet werden, Zeichenstile bezeichnen Auszeichnungen,

[39] Beispiel für ein (selbsterklärendes) Deftag: …der normale Text und $(fett:hier beginnt fett und hört hier)$ wieder auf…

[40] Schriftschnitte gibt es streng genommen nur im physischen Druck, bei digitalen Fonts spricht man von Schriftstilen.

die nur auf eine bestimmte Teilmenge eines Absatzes angewandt werden. Diese Stile werden in Systemen, welche den Austausch der Metainformation „Name und Definition des Stils xxx“ unterstützen, unter einem eindeutigen Namen gespeichert. Damit wird die Wiederverwendung schneller und einfacher und sie sind für den Anwender so einfacher zu merken. Dies erlaubt die Formatierung unabhängig von den konkreten typographischen Auszeichnungen sowie das einfache globale Ändern von Auszeichnungen allein in den Formatdefinitionen.

- Nicht zu transformierende Metadaten waren Informationen wie Seitenformat, Textboxgröße, Anordnung auf der Seite, Anordnung von Grafiken, Umbruch-Bedingungen und Ähnliches. Diese zur Makrotypographie gehörenden Elemente werden in einem Redaktionssystem über medien- und objektspezifische Ausgabetemplates gesteuert, in welche die einzeln formatierten Content-Stücke (siehe Fußnote) importiert werden. Damit sollten die Forschungsarbeiten fokussiert und die Komplexität um die weitere Dimension reduziert werden, die Dokumentheaderinformationen ebenfalls generieren zu müssen. Je nach Format enthalten die Kopfdaten unterschiedliche Bereiche von Metadaten. Eine Vereinigungsmenge zu finden, die über alle Formate abbildbar ist, hätte die Komplexität massiv erhöht, ohne aus dieser Perspektive einen nennenswerten Nutzen zu erkennen.

- Die Semantik der typographischen Metadaten sollte bei allen Transformationen, die potenziell bidirektional ablaufen können, erhalten bleiben. Wegen der unterschiedlichen Mächtigkeit der beteiligten Formate musste ein Weg gefunden werden, die Informationen, die vom mächtigsten beteiligten Format verarbeitet werden können, auch in allen anderen Formaten vorzuhalten. Die meisten weniger mächtigen Formate bieten in Form von Kommentaren oder kommentarähnlichen Konstrukten die Möglichkeit, externe, nicht verwertbare Information zu konservieren. Indem man Daten, die ein Format nicht verarbeiten kann, in solchen Containern aufbewahrt, ermöglicht man bidirektionale Umwandlungen zwischen mächtigeren und schwächeren Formaten. Daraus ergeben sich die Vorteile, für die verschiedenen Prozessschritte weiter die jeweils darauf zugeschnittenen Programme[41] einsetzen zu können, sowie bestehende und gewohnte Arbeitsabläufe weitgehend beibehalten zu können.

- Im ersten Schritt sollten folgende externe Sprachen unterstützt werden: RTF, XTG[42] und HTML. Bei letzterem wird zusätzlich zwischen einer unidirektionalen Exportfunktionalität für die reine Ausgabe in Online-Medien und einer HTML-Darstellung in einem WYSIWYG-Editor unterschieden, welcher den Redakteuren als be-

[41] Beispielsweise Textverarbeitungsprogramme zum Schreiben und für die Rechtschreibprüfung, DTP-Programme für Arbeiten am Layout, etc.

[42] Abkürzung für „Xpress TaG“, Exportformat von Quark XPress

vorzugtes Werkzeug für die Bearbeitung dienen soll. Da die exportierten Daten lediglich als Präsentationsebene im Browser dienen sollten, können für die Typographie in HTML nicht verwertbare Metadaten gelöscht werden, da die Daten anschließend nicht mehr reimportiert werden. Im Gegensatz hierzu müssen Daten, welche im WYSIWYG-Editor bearbeitet wurden, ohne Verlust der Semantik wieder zurück in das zentrale Cross Media Publishing-System gespeichert werden. Somit dürfen für die Darstellung im HTML-Editor unbekannte Tags nicht verworfen werden. Diese müssen geeignet maskiert werden. RTF soll ähnlich wie WYSIWYG-HTML als Format genutzt werden, in dem Autoren Texte bearbeiten können. Auch hier müssen Übersetzungen für beide Richtungen entwickelt werden. Die Menge der zulässigen Auszeichnungsinformationen muss auf solche beschränkt werden, die von den entsprechenden Übersetzern behandelt werden können. Falls ein Benutzer im Editor nicht unterstützte RTF-Auszeichnungen einfügt, müssen diese unter Ausgabe einer entsprechenden Warnung verworfen werden.

- Zur internen Repräsentation der typographischen Metadaten war es das Ziel, ein XML-basiertes Format mit dem Arbeitstitel TypoML[43] zu entwickeln, dessen Dokumentstrukturen mit *XML Schema*[44] beschrieben werden sollten. Die Anlehnung an XML sollte es zukünftigen Entwicklern erlauben, sich schnell in TML zurechtzufinden. XML-Dokumente können vage zwischen datenorientierten und textorientierten Formaten unterschieden werden. Ziel war die Orientierung an einem textorientierten Format, damit bei Eingriffen durch Administratoren und bei Supportleistungen eine möglichst gute Identifikation der Inhalte gewährleistet ist. Letztendlich ergab sich wie in fast allen Fällen eine Mischform mit Schwerpunkt auf ein von Menschen noch lesbares Format. Kapitel 14 dieser Arbeit beschreibt diese Differenzierung detaillierter.
- Den Kern des zu entwickelnden Transformationssystems sollte eine Gruppe von sprachspezifischen Compilern bilden, welche die bidirektionale Transformation zwischen TML und den externen Fremdformaten ohne Verlust der Metadaten-Semantik ermöglichen.
- Die Transformationsregeln werden als Produktionen in sprachspezifischen Grammatiken gespeichert. Diese enthalten die Regeln für Transformationen von Absatz-, Zeichen- und Spezialformatierungen. Letztere umfassen zu transformierende Informationen, welche weder unter Absatz- noch Zeichenformate fallen. Pro externem Sprachformat können mehrere Grammatiken existieren.

---

[43] kurz: TML

[44] Bereits zu Beginn des Projektes war klar, dass eine Strukturdefinition per DTD nicht mächtig genug sein würde. Zudem sprachen eine Reihe weitere Gründe gegen die Verwendung von DTDs, bspw. die Zukunftssicherheit, die Werkzeug-Kompatibilität zu XML. Nachteilig im Vergleich zu DTDs ist die höhere Komplexität von XSDs (XML Schema Definitions) und die damit verbundene schlechtere Lesbarkeit.

- Vermutlich würden die Nutzer des TML-Systems später im Produktionsprozess regelmäßig die Produktionen der Grammatiken, also die eigentlichen Transformationsregeln, anpassen wollen, beispielsweise um die Generierung der Tags für die Online-Typographie von HTML nach XHTML umzustellen oder um die Definition eines Absatzstiles anzupassen. Damit dafür kein direktes Arbeiten in der Datenbank nötig ist, war bereits für den Prototyp eine browserbasierte Pflegeoberfläche geplant. Neben dem Interface für die Regelhaltung stehen Interfaces für die Manipulation der verschiedenen Mapping-Tabellen in der Datenbank zur Verfügung, beispielsweise Interfaces für die Codepage-, Entity- und Stylebehandlung[45].

- Für eine korrekte Zeichendarstellung müssen Codepageinformationen vom Transformationssystem in Abhängigkeit von Schriftarten und dem verwendeten Betriebssystem verwaltet und verarbeitet werden. An den Arbeitsplätzen der beteiligten Systeme kommen die unterschiedlichsten Betriebssysteme und Schriften zum Einsatz. Es musste sichergestellt werden, dass im zentralen Datenhaltungsformat alle Zeichen der verschiedenen existenten Zeichensätze gespeichert werden können. Dazu ist es notwendig, bei der Konvertierung der externen Daten in TML auch die reinen Textinhalte nach Unicode zu transferieren. Ebenso notwendig ist eine Konvertierung von Unicode in den entsprechenden Zeichensatz bei einer Ausgabe der Daten in einem der externen Dateiformate. Der gewählte Zeichensatz ist zum einen entweder global für das gesamte Dokument festgelegt oder mit den verwendeten Schriften definiert, da diese Schriften auf einem bestimmten Zeichensatz aufbauen. Um also die Daten korrekt konvertieren zu können, musste zuerst einmal bekannt sein, um welchen Zeichensatz es sich handelt. Diese Informationen sind nicht ohne weiteres aus den gelieferten Dokumenten auslesbar, müssen aber den zu konvertierenden Dokumenten zugeordnet werden können.

- Ähnlich der Codepage-Behandlung müssen auch schriftartabhängige Sonderzeichen für die interne Repräsentation in geeignete Unicode-Entities umgewandelt werden. Für nicht vorhandene Übersetzungsanweisungen muss ein fehlertoleranter Mechanismus entwickelt werden. Fehlt ein Konvertierungseintrag wird lediglich die reine Schrift- und Zeicheninformation gespeichert.

- Maskiert werden die Metadaten in einem Format nach XML-Spezifikation. Das führt natürlich zu Problemen, wenn eine Sprache mit vergleichbarer Syntax inhaltlich beschrieben werden soll. Um die Interpretation als Auszeichnungsinformation und damit ein Fehlverhalten des Transformationssystems zu verhindern, müssen die Auszeichnungsinformationen der unterstützten externen Sprachen, die inhaltlich im Text des zu verarbeitenden Dokumentes

[45] Diese Teilaspekte werden später ausführlich erläutert.

vorkommen, bei der Transformation nach TML geeignet maskiert werden[46].

- Den Entwicklern und Administratoren sollte der Umgang mit dieser Technologie möglichst einfach gemacht werden. Als Entwicklungsziel für die interne Repräsentation der Tags wurde eine möglichst große Nähe zu den typografischen HTML-Tags angestrebt.
- HTML ist eine SGML-Applikation und erbt daraus einige Standardregeln für die Dokumentsyntax. Die exakten Spezifikationen sind auf den Seiten des W3C veröffentlicht, für HTML aktuell in der Version 4.01 mit drei DTDs: Strict, Transitional und Frameset. Historisch haben sich die User Agents für das WWW sehr fehlertolerant entwickelt. Verstöße gegen die DTDs führen im Allgemeinen nicht zu Fehlermeldungen. Unsauberer HTML-Text wird weitgehend fehlertolerant interpretiert oder ignoriert. Es ist ohne große Probleme für die Ausgabe im Browser möglich, schließende Tags einfach wegzulassen oder Tags zu schachteln. In XML sind verschachtelte Tags nicht erlaubt, ein solches XML-Dokument wäre nicht wohlgeformt. Um solche in der Praxis vorkommenden HTML-Unsauberkeiten in TML zu speichern, müssen diese vorher in eine überlappungsfreie Form umgewandelt werden.
- Das Beispiel:

```
Test von überlappenden Tags:<b>Hier beginnt
fett und <i>hier kursiv. Hier endet fett</b>
und hier kursiv.</i>Das war´s.
```

wird in den getesteten Standard Internet-Browsern (Mozilla Firefox und Microsoft Internet Explorer) wie beabsichtigt dargestellt (siehe Abbildung II-9), führt aber in allen Validatoren des W3C zu entsprechenden Fehlermeldungen.

[46] Beispiel: Ein RTF-Dokument, welches inhaltlich die Sprache HTML behandelt.

ABBILDUNG II-9: IN ALLEN HTML-DTDS DES W3C IST DIE EBENENTREU PAARIG VERSCHACHTELTE TAG-ORDNUNG ZWAR VORGESCHRIEBEN, VERSTÖßE WERDEN VON DEN AKTUELLEN INTERNET-BROWSERN TROTZDEM FEHLERTOLERANT INTERPRETIERT.

- Die Umformung in ein überlappungsfreies Format mit identischer Semantik könnte zu dem Problem führen, dass mit der Zeit das Dokument ‚zumüllt', also viele redundante Typographie-Daten dauerhaft im Dokument verbleiben. Dieser Effekt kann durch einen entsprechenden „Optimierungscompiler/Code-Optimierer" gemindert werden. Dieser Compiler ist vorerst uninteressant, da es sich vermutlich nur um ein theoretisches Problem handelt. Beim Aufbrechen der Überlappung muss beachtet werden, ob die DTP-Systeme den neuen Code genauso interpretieren wie vor dem Aufbrechen, um semantische Gleichheit zu gewährleisten.
- Eine Persistenzschicht war innerhalb des Transformationssystems nicht vorgesehen, diese wird allein vom zentralen Cross Media Publishing-System realisiert. Das Transformationssystem speichert die einkommenden Daten während der Übersetzungsprozesse nur temporär zwischen.
- In den beteiligten Medienwelten und Standardformaten haben sich verschiedene Maßeinheiten für Schriftgrößen, Abstände, etc. etabliert. Es wurde erwartet, dass zum Vorteil des im Onlinebereich vorherrschenden Punktformates andere Maßeinheiten verblassen. Außerdem wurde angenommen, dass alle beteiligten externen Anwendungssysteme die Maßeinheit Punkt im- und exportieren können.
- Die von den beteiligten Zielsystemen generierten Dokumentrahmen sollten nicht geparst oder verändert werden.

Optionale Anforderungen

- Es wurde ein Grundstock an Auszeichnungen definiert, welcher ein Maß an typographischen Gestaltungen erlaubt, die einen Großteil aktueller Publikationen abdeckt. Beispielsweise wurde die Unter-

stützung von Tabellen und Formeln zunächst als optional definiert, da bereits zu Beginn ersichtlich war, dass diese Punkte einen erheblichen Implementierungsaufwand bedeuten würden. Abhängig vom zeitlichen Fortschritt könnten diese Elemente zusätzlich in den Prototypen einfließen.

- Um während der Entwicklungsphase den erzeugten TML-Code zu prüfen, sollte ein auf Xerces-P[47] basiertes Konsolenwerkzeug entwickelt werden, welches im Debug-Modus der Transformationsunit die Validität bezüglich des XML-Standards überprüft.
- Neben der Unterstützung von Quark XPress wurde eine Anbindung an das zweite marktbedeutende Desktop Publishing System „Adobe InDesign“ angestrebt. Eine Unterstützung des Exportformates dieses Produktes unterliegt aufgrund der vergleichbaren Aufgabenstellung der beiden Softwareprodukte denselben Rahmenbedingungen. Die Architekturspezifikationen des Transformationssystems sollen die Anbindung von InDesign und eventuell zukünftig relevanter DTP-Systeme berücksichtigen.

[47] Xerces-P ist die Perl-Schnittstelle des Xerces XML-Parsers, mit dessen Hilfe man u.a. XML validieren kann

Teil III

State of the Art

Content Processing und Medienbrüche

K a p i t e l 7

XML-BASIERTE FORMATIONS- UND TRANSFORMATIONSSPRACHEN

Falls Gott die Welt geschaffen hat, war seine Hauptsorge sicher

nicht, sie so zu machen, dass wir sie verstehen können.

(Albert Einstein)

Vor der Entscheidung, ein neues, auf XML basierendes Format für den Austausch typographischer Metadaten zu entwickeln, stand die Analyse bereits bestehender Technologien. Analysekriterien waren die konzeptionelle Ausrichtung der Verfahren, die Praxistauglichkeit im Sinne von Installation, Konfiguration und Dokumentation, sowie die Praxistauglichkeit im Sinne von Laufzeitverhalten und Stabilität.

Teil III dieser Arbeit beschreibt die Grundlagen der relevanten Verfahren sowie die wesentlichen Erkenntnisse der Analysephase. Dabei erhebt dieser Teil keinen Anspruch auf Vollständigkeit. Neue Technologien zur Bewältigung von Medienbrüchen entwickeln sich seit Beginn dieser Arbeit mit enormem Tempo, zahllose Einzellösungen und Ansätze zur meist unidirektionalen Transformation zwischen dedizierten Formaten wurden in dieser Zeit veröffentlicht.

XML-Daten lassen sich zwar mittels CSS formatieren, allerdings kann CSS nur einen Teil der von XML - neben der eigentlichen Formatierung - geforderten Möglichkeiten bieten. Die Lösung ist XSL. XSL bedeutet „Extensible Stylesheet Language“ und ist die XML-eigene formatting engine. XSL Stylesheets enthalten im Gegensatz zu CSS nicht nur einfache Formatierungsregeln, sondern einfache Programmierkonstrukte wie Schleifen, Bedingungen und Verzweigungen, welche die bedingte Verarbeitung von Daten ermöglichen.

Aktuell teilt sich XSL in drei verschiedene Teilstandards auf:

1. XSL Transformation (XSLT) für die Übersetzung von XML-Strukturbäumen

2. XSL Formatting Objects (XSL-FO) für die typographische Beschreibung druckfähiger Dokumente

3. XML Path Language (XPath) für die Adressierung einzelner Teile in XML-Dokumenten

Den Zusammenhang der im Weiteren beschriebenen Technologien verdeutlicht Abbildung III-2.

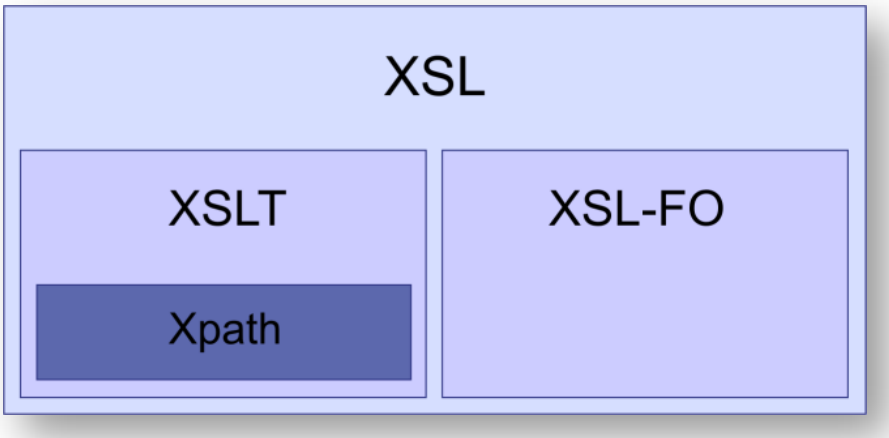

ABBILDUNG III-1: DIE DREI TEILSTANDARDS VON XSL.

Mit XSLT und XSL-FO werden im Folgenden die wichtigsten Technologien zur Transformation strukturierter XML-Daten beschrieben. Es folgen zwei weniger populäre Technologien, welche ursprünglich für SGML entwickelt wurden und die deshalb auch auf XML-Dokumente angewendet werden können: DSSSL und Cost/TCL.

Weitere relevante Formate, auf die im Rahmen dieser Arbeit nicht weiter eingegangen wird, sind beispielsweise PDF/X [Sch08], als Format für die verlässliche, digitale Druckvorlage und die für die Archivierung vorgesehen Variante PDF/A [PDF08].

Um Workflows und Prozesse bei der Verarbeitung von Mediendokumenten verbessern zu können, hat Adobe seinen Standard „Extensible Metadata Platform“ (XMP) [Ado08] für die Integration von Metadaten in Dokumentformate wie PDF oder in die meisten bekannten Bildformate etabliert.

Microsoft hat als direkte Konkurrenz zu PDF die „XML Paper Specification“ (XPS) [Mic09] entwickelt. Ein Plugin zur direkten Ausgabe kann direkt in Microsoft Office installiert werden. In der Praxis hat dieses Format bisher nur geringe Bedeutung erlangt.

Für den automatisierten Austausch von hierarchisch gegliederten Dokumenten wurde die Outline Processor Markup Language (OPML)[48] entwickelt, sie wird beispielsweise beim Austausch von RSS-Feeds verwendet. Umfangreiche Transformations- und Aggregationsmöglichkeiten im Kontext von RSS und OPML bietet der Dienst „XFruits“[49].

[48] http://www.opml.org/

[49] http://www.xfruits.com/

Kapitel 8

TRANSFORMATION MIT XSLT

Eine wichtige Technologie im Rahmen von Transformationen von XML-Dokumenten ist XSLT. XSLT steht für *XSL Transformation*. XSL wiederum steht für *Extensible Stylesheet Language*.

XML ist wie bereits erwähnt keine eigene Sprache im formalen Sinne, sondern ‚nur' eine Sammlung von Syntaxregeln. XML-Dokumente sind daher immer Dokumente in einer Sprache, die aus XML abgeleitet ist. XML-Sprachen sind beispielsweise Docbook, XHTML, SVG oder MathML. Die syntaktischen Ausprägungen aller dieser Sprachen sind durch ihre jeweilige DTD oder XML Schema-Definition beschrieben. Die Verwendung von DTDs oder Schemata ist nicht beschränkt oder vordefiniert, jeder Anwender kann eigene XML-basierte Sprachen mit eigenen Grammatiken definieren.

Diese Vielfalt von offiziellen und proprietären XML-Definitionen und der Grundgedanke, XML als Standardformat für den Datenaustausch und –transport einzusetzen, führt schnell zu der Anforderung, Daten, die in einer XML-Sprache vorliegen, auszuwerten und als Ergebnis Daten in einer anderen XML-Sprache zu erhalten. Hat man beispielsweise eine Menge von Messwerten in einem eigenen Format als Datenbestand vorliegen, möchte man vielleicht eine graphische Darstellung dieser Daten erzeugen, die man mit SVG beschreiben könnte. Dieser Sachverhalt war die Motivation zu XSLT. XSLT ist also in der Lage, eine Transformation von Daten aus einer XML-basierten Sprache in eine andere (nicht notwendigerweise auf XML-basierende) Sprache vorzunehmen.

XSLT selbst ist ebenfalls eine XML-Sprache, die also in der Lage ist, die Baumstruktur von XML-Dokumenten zu verändern. XSL enthält neben Formatierungsregeln auch einfache Programmierkonstrukte. Diese ermöglichen die Definition von einfachen Verarbeitungsregeln (wie beispielsweise bedingte Formatierung oder Sortierung von Daten) und das ist einer der Gründe warum XSL mächtiger ist als CSS. Ein weiterer Vorteil der XSL ist, dass sie selbst – anders als CSS – XML-basiert ist. Somit lassen sich viele Applikationen für XSL benutzen, die eigentlich für XML entwickelt wurden.

> *XSLT ist die Abkürzung von XSL Transformation und kann XML-Dateien in andere XML-Dateien, strukturierte oder textbasierte Formate transformieren. XSLT wird hierbei von XPath unterstützt (XML Path Language)*

XSLT ist die Transformationskomponente von XSL. XSL ist u.a. aus DSSSL (siehe Kapitel 10) hervorgegangen und selbst XML-basiert. Die Transformationsanweisungen

werden in einem sogenannten Stylesheet (oder auch Transformation) festgelegt. Ein Programm, das diese Stylesheets verarbeiten kann, heißt XSLT-Prozessor. Mit Hilfe eines XSLT-Prozessors lässt sich also eine XML-Datei in andere XML-Dateien, strukturierte oder textbasierte Formate umwandeln. Unterstützt wird XSLT hierbei von XPath. XPath ermöglicht es, einzelne Teile in einem XML-Dokument zu adressieren oder auf diese zu verweisen.

Die serialisierte Darstellung von Datenstrukturen in XML lässt sich auch als Baumstruktur darstellen. Dabei spricht man vom Quellbaum für die Ausgangsdaten und vom Ergebnisbaum für die transformierten Zieldaten. Jede der Transformationsregeln besteht aus einem mittels XPath erzeugten Muster, welches beschreibt, für welche Knoten im Quellbaum die Regel angewendet werden soll und einem Inhalt (auch Template oder Schablone genannt), der bestimmt, wie die Regel den neuen Teil des Ergebnisbaums erzeugen soll. Da ein typisches Stylesheet sowohl aus XSL-eigenen als auch aus Elementen und Attributen der Sprache besteht, in die transformiert werden soll, müssen vorher die benötigten Namensräume in das Stylesheet importiert werden.

Hauptanwendungsgebiete für XSLT sind Transformationen zwischen XML-Dialekten oder die Wandlung von XML-Daten in eine Internetbrowser-fähige Darstellung in HTML bzw. XHTML.

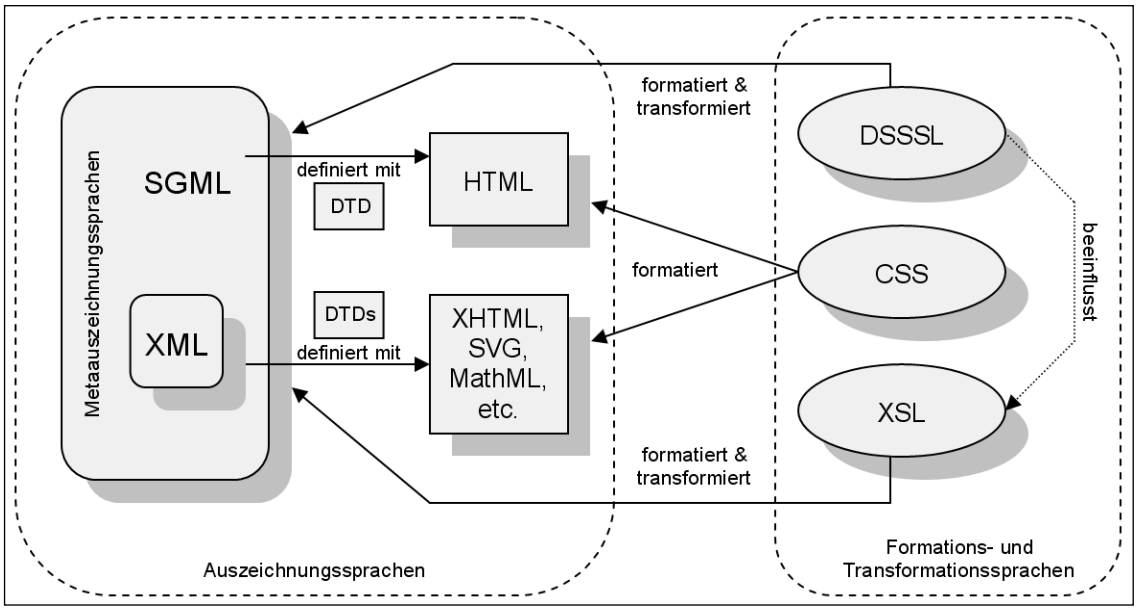

ABBILDUNG III-2: *ZUSAMMENHÄNGE ZWISCHEN XML, SGML, DTD, HTML, XHTML, CSS, XSL, ETC.* (NACH [MIN03])

Eine besondere Rolle spielen XML und XSLT bei Anwendungen, deren Ziel die Darstellung im WWW oder der Datenaustausch auf Basis von Internet-Technologien ist. Webseitenanbieter können ihre Daten in XML sauber speichern und verwalten und trotzdem ihre Webseiten in HTML oder XHTML präsentieren. Auch JavaScript oder CSS lassen sich mit XSLT einbauen. Beispielsweise können mit XSLT Daten serverseitig gewandelt und anschließend mit CSS clientseitig formatiert werden. Aus zwei Elementen ‚vorname' und ‚nachname' in einer XML-Datei könnte man zum Beispiel eine HTML-

Tabellenzeile erstellen, in der der Inhalt von ‚vorname' und ‚nachname' jeweils in einer eigenen Tabellenzelle abgelegt wird. Auch komplexere Transformationen sind möglich. So lässt sich eine in XML als Text gespeicherte E-Mail-Adresse zur Ausgabe in HTML mit einem Link hinterlegen, indem man im HTML-Ergebnisbaum dem href-Attribut des Elementes ‚a' diese E-Mail-Adresse zuweist.

Grundsätzlich ist jede Programmiersprache wie Java, C++ oder Perl eine Alternative zu XSLT, jedoch erstellt XSLT unter Einhaltung bestimmter Voraussetzungen garantiert wohlgeformte und eventuell sogar gültige Zieldokumente. Außerdem sind die Templates in XSLT oft einfacher zu entwickeln als in einer klassischen Programmiersprache.

Eigenschaften und Einsatzgebiete von XSLT

Ein XSLT-Prozessor verarbeitet ein XML-Dokument auf der Ebene seiner Knoten. Ein XSLT-Stylesheet besteht aus einer Reihe von Transformationsregeln, zusammengefasst in Templates. Jedes Template besitzt einen mittels XPath gebildeten Ausdruck, mit welchem die Knoten des Quellbaumes identifiziert und adressiert werden, auf die das jeweilige Template anzuwenden ist. Zudem beinhaltet ein Template die Angaben, wie der zugehörige Teil des Zielbaumes zu gestalten ist.

Die Ausgabetypen von XSLT-Transformationen werden grob in zwei Kategorien gegliedert:

POP – Presentation Oriented Publishing

POP bezeichnet die Transformationen, die zum Zwecke der Darstellung vorgenommen werden. Ein semantisches Markup wird also durch ein stilistisches Markup ersetzt.

Ein XML-Datenbestand kann so mit unterschiedlichen Templates in verschiedene Ausgabeformate gebracht werden. So etwa die oben erwähnte graphische Darstellung mit SVG, eine Ausgabe als XHTML, als DocBook oder in andere darstellungsbezogene XML-Sprachen.

MOM – Message Oriented Middleware

MOM hingegen bezeichnet Transformationen, die primär dem Austausch von Daten dienen. Zum Austausch von XML-Daten genügt es nicht, dass die beteiligten Systeme XML-basiert arbeiten, da XML nur ein Konzept für konkrete Sprachen darstellt. Es ist vielmehr notwendig, dass die Systeme die gleiche XML-basierte Sprache verwenden. Da dies häufig nicht unmittelbar der Fall ist, ist eine Transformation der Daten von einer DTD bzw. von einem XML Schema in die andere notwendig.

Eine solche Transformation kann beispielsweise in einem Importfilter eines Textverarbeitungsprogrammes integriert ein, der das XML-Format eines anderen Textverarbeitungsprogramm in das eigene übersetzt.

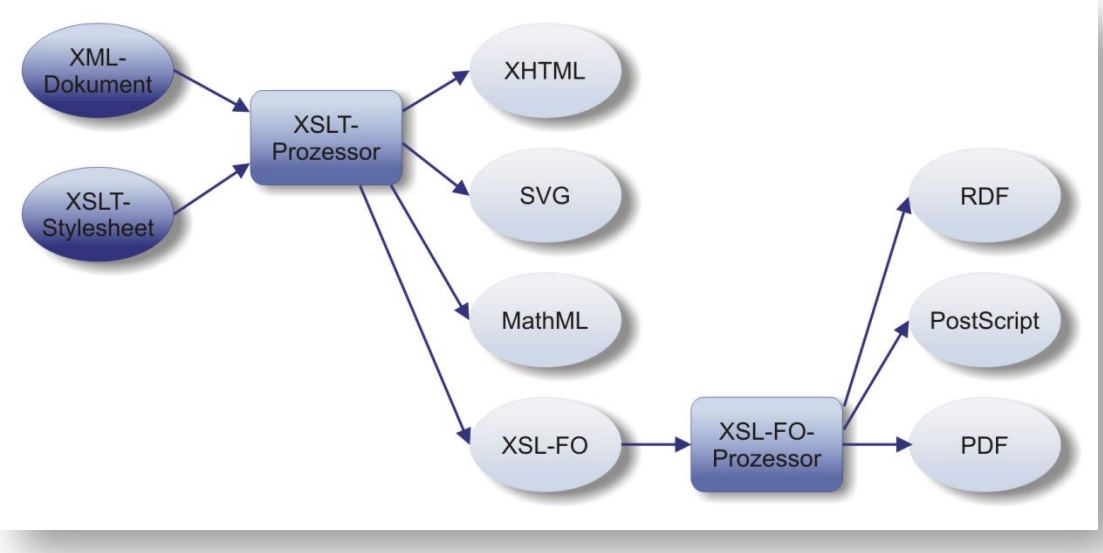

ABBILDUNG III-3: DER XSLT-PROZESSOR ERZEUGT AUS STRUKTURIERTEN DATEN UND STILANGABEN FERTIGE AUSGABEFORMATE (BEISPIELSWEISE XHTML) ODER ZWISCHENFORMATE FÜR DIE WEITERE VERARBEITUNG (BEISPIELSWEISE XSL-FO).

Mit XSLT zu HTML

Um dem Leser die Anwendung von XSLT ohne große Hürden nahe zu bringen, wird im Folgenden eine einfache Beispieltransformation gezeigt.

Als Ausgangsdaten dienen einfache Fehlerberichte (bericht.xml), die von einer fiktiven Applikation im XML-Format erzeugt werden. Als Ausgabeformat ist einfaches HTML gewünscht.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE bericht SYSTEM "bericht.dtd">
<bericht>
     <melder>Hans Mustermann</melder>
     <datum>2008-06-02</datum>
     <fehlerkategorie>GUI-Fehler</fehlerkategorie>
     <beschreibung>
          Die Zeitanzeige verschwindet teilweise hinter dem
          rechten Rand des Hauptfensters, wenn das Fenster
          verschoben wird.
     </beschreibung>
</bericht>
```

Quelltext III-1: Fehlerbericht als XML-Dokument

Ein mögliches Stylesheet wäre das Folgende:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version ="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"
     doctype-public="-//W3C//DTD HTML 4.0 Transitional"
     doctype-system="http://www.w3.org/TR/REC-html40"
     indent="yes"/>

<xsl:template match="bericht">
          <html>
          <head>
               <title>Fehlerbericht -
                    <xsl:apply-templates se-
lect="fehlerkategorie"/>
               </title>
          </head>
          <body>
               <h1><xsl:value-of se-
lect="fehlerkategorie"/></h1>
               Datum:  <xsl:value-of select="datum"/><br/>
               Melder: <xsl:value-of select="melder"/><br/>
               Beschreibung:<br/>
               <xsl:value-of select="beschreibung"/><br/>
          </body>
          </html>
     </xsl:template>
</xsl:stylesheet>
```

Quelltext III-2: Template zur Wandlung von XML-Berichten in HTML-Berichte

Genutzt werden kann dieses Stylesheet nun durch einen XSLT-Prozessor wie Saxon oder Xalan. Ist Xalan installiert, kann die Transformation in der Eingabekonsole (unter Linux) etwa durch einen Aufruf von

```
xalan -html -in bericht.xml -xsl bericht.xsl -out bericht.html
```

in die Ausgabedatei `bericht.html` geschrieben werden[50]. Die DTD in der Datei `bericht.dtd` muss dabei zwingend vorhanden sein. Oder man verzichtet auf eine Grammatik für `bericht.xml`, dann erwartet Xalan auch keine DTD.

[50] Xalan schreibt ohne weitere Angaben in die Standardausgabe. So kann man die Ausgabe beispielsweise direkt für eine http-Antwort verwenden.

Erläuterung des Beispiels

Alle Anweisungen an den XSLT-Prozessor beginnen, wie oben ersichtlich, mit „xsl:“. Als Ausgabeform („output method“) wurde HTML angegeben. Neben HTML wäre hier auch XML möglich. HTML muss deshalb als besonderes Ausgabeformat angegeben werden, weil HTML nicht in allen Punkten den formalen Kriterien einer XML-Sprache entspricht. So müssen beispielsweise die Tags ohne Endtag (wie
) in der XML-Quelle mit abschließendem Slash geschrieben werden. In der Ausgabe dürfen manche dieser Tags jedoch keinen Slash mehr besitzen.

Ein Template wie in Quelltext III-2 bewirkt eine Transformation eines Quellknoten in einen Zielknoten. Das im Attribut „match“ angegebene Muster gibt an, auf welchen Knoten des Quelldokuments sich das Template bezieht. Hierbei werden XPath-Ausdrücke verwendet.

Ein paar Beispiele für solche Ausdrücke[51]:

```
/                    Wurzelknoten
/tle                 Top-Level-Element  „tle“
absatz | listing     Element „absatz“ oder „listing“
id('myid42')         Element mit dem Attribut „id“ und dem Wert
                     „myid42“
```

Innerhalb des Templates steht die Ausgabe für das Zieldokument. „apply-templates“ sorgt dafür, dass die spezifizierten Kindknoten des aktuellen Knoten abgearbeitet werden. Mit der Anweisung „value-of“ lassen sich die Inhalte spezieller Quellknoten abrufen.

Ein Inhaltsverzeichnis mit XSLT

Es gibt viele XSLT-Anweisungen, um den Ausgabebaum zu generieren und ebenfalls viele XSLT-Anweisungen, um den Quellbaum abzuarbeiten. Um ein etwas umfassenderes Beispiel aufzuzeigen, sei hier das folgende Listing[52] angegeben:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:html="http://www.w3.org/1999/xhtml"
    xmlns="http://www.w3.org/1999/xhtml"
    exclude-result-prefixes="html"
>

    <xsl:output
```

[51] Beispiele aus [Beh00]

[52] Beispiel aus [Wik04]

```
        method="xml"
        doctype-
system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"
        doctype-public="-//W3C//DTD XHTML 1.1//EN"
    />

<xsl:template match="html:body">
        <xsl:copy>
            <xsl:apply-templates select="@*"/>
            <h1><xsl:apply-templates select="//html:title//text()"
/></h1>
            <h2 id="toc">Inhaltsverzeichnis</h2>
            <ul>
                <li><a href="#toc">Inhaltsverzeichnis</a></li>
                <xsl:for-each select=".//html:h2">
                    <li>
                        <xsl:if test="not(@id)">
                            <xsl:message>Achtung: Kann ohne Id kei-
nen Link erzeugen</xsl:message>
                        </xsl:if>
                        <a href="#{@id}"><xsl:apply-templates/></a>
                    </li>
                </xsl:for-each>
            </ul>
            <xsl:apply-templates/>
        </xsl:copy>
    </xsl:template>

    <xsl:template match="node()|@*">
        <xsl:copy>
            <xsl:apply-templates select="@*"/>
            <xsl:apply-templates/>
        </xsl:copy>
    </xsl:template>
```

Quelltext III-3: XSLT-Stylesheet zur Erzeugung eines HTML-Inhaltsverzeichnisses

Die angegeben Templates erzeugen ein einfaches Inhaltsverzeichnis für eine HTML-Seite.

Erläuterung des obigen Beispiels

Zu erkennen ist, dass zuerst der Titel des Dokuments (aus dem `title`-Tag des Headers) als `h1`-Überschrift ausgegeben wird. Dann wird der Text „Inhaltsverzeichnis“ als `h2`-Überschrift ausgegeben. Es folgt eine `ul`-Liste, welche als ersten Eintrag das Inhaltsverzeichnis selbst hat. Anschließend folgt eine `for-each`-Schleife, die für jedes `h2`-Element des Bodys einen Link erstellt (anhand der ID). Besitzt ein `h2`-Element des Bodys keine ID, wird eine entsprechende Warnung ausgegeben.

Literaturverzeichnis dieser Dissertation

Ein weiterer, konkreter Anwendungsfall für die Programmierung von XSLT-Stylesheets ergab sich während dem Verfassen dieser Arbeit. Die neue Word-Version in Office 2007 enthält zum ersten Mal eine dedizierte Komponente für die Verwaltung von Literaturverzeichnissen. Format des Verzeichnisses und der Zitate kann der Anwender aus einer Liste von Vorlagen wählen. Hier sind die wichtigsten Standards wie APA, Chicago oder ISO 690 hinterlegt. Keine der Vorlagen entsprach jedoch dem gewünschten Format, nach dem bei Referenzen bzw. Zitaten der erste Autor zusammen mit dem Veröffentlichungsjahr in eckigen Klammern verwendet wird.

Microsoft hat das Speicherformat in Office 2007 komplett auf XML umgestellt. Für die Formatierung des Literaturverzeichnisses wird ein Satz von XSLT-Stylesheets verwendet. Um das gewünschte Ausgabeformat zu erreichen, waren einige Veränderungen an den Stylesheets notwendig. Als Ausgangsbasis dienten die Stylesheets für den Standard ISO 690 Numerical[53].

Folgende Dateien und Verzeichnisse sind dabei wichtig:

Office 2007 verwaltet alle Bibliotheks-Ressourcen in folgender Datei (sie wird erst erzeugt, wenn man mindestens eine Ressource angelegt hat):

```
%APPDATA%\Microsoft\Bibliography\Sources.xml
```

Unter Microsoft Vista lautet dieses Verzeichnis beispielsweise:

```
C:\Users\[Username]\AppData\Roaming\Microsoft\Bibliography
```

Beispiel-Eintrag aus der `Sources.xml` für ein Buch:

```
<?xml version="1.0" ?>
- <b:Sources SelectedStyle=""
xmlns:b="http://schemas.openxmlformats.org/officeDocument/2006/bibliography"
xmlns="http://schemas.openxmlformats.org/officeDocument/2006/bibliography">
- <b:Source>
      <b:Tag>Mül02</b:Tag>
      <b:SourceType>Book</b:SourceType>
      <b:Guid>{E13E4920-C510-4D04-9077-284E19AD14F7}</b:Guid>
      <b:LCID>1031</b:LCID>
      <b:Author>
            <b:Author>
                  <b:NameList>
```

[53] Eine Anleitung für den Neuaufbau eigener Templates mit rudimentären Funktionen hat Microsoft unter der Adresse http://blogs.msdn.com/microsoft_office_word/archive/2007/12/14/bibliography-citations-1011.aspx veröffentlicht.

```
                    <b:Person>
                        <b:Last>Müller-Kalthoff</b:Last>
                        <b:First>Björn</b:First>
                    </b:Person>
                </b:NameList>
            </b:Author>
        </b:Author>
        <b:Title>Cross-Media Management</b:Title>
        <b:Year>2002</b:Year>
        <b:City>Berlin</b:City>
        <b:Publisher>Springer</b:Publisher>
        <b:StandardNumber>ISBN 3-540-43692-8</b:StandardNumber>
    </b:Source>
```

Quelltext III-4: Bucheintrag in der Literaturverwaltung dieser Arbeit

Die XSL-Dateien für die verschiedenen Ausgabestile liegen im Verzeichnis:

```
C:\Program Files\Microsoft Office\Office12\Bibliography\Style
```

In diesem Verzeichnis findet sich die zu modifizierende Stilanweisung `ISO690Nmerical.xslt`. Wenn man sowohl im Zitat als auch im Literaturverzeichnis das Quellen-Tag in eckigen Klammern haben will, also

…Text Lore Ipsum [Har02] Lore Ipsum…

und das Literatur-Verzeichnis:

...[Har02] Harrot, James. Titel...

müssen zunächst die Zeilen 2047 und 2058 geändert werden:

Vorher:

```
  <xsl:template name="templ_prop_OpenBracket" >
    <xsl:param name="LCID" />
    <xsl:variable name="_LCID">
      <xsl:call-template name="localLCID">
        <xsl:with-param name="LCID" select="$LCID"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:value-of se-
lect="/*/b:Locals/b:Local[@LCID=$_LCID]/b:General/b:OpenBracket"/>
  </xsl:template>

  <xsl:template name="templ_prop_CloseBracket" >
    <xsl:param name="LCID" />
    <xsl:variable name="_LCID">
      <xsl:call-template name="localLCID">
        <xsl:with-param name="LCID" select="$LCID"/>
```

```
      </xsl:call-template>
    </xsl:variable>
    <xsl:value-of se-
lect="/*/b:Locals/b:Local[@LCID=$_LCID]/b:General/b:CloseBracket"/>
  </xsl:template>
```

Quelltext III-5: Ausschnitt des originalen Stylesheet von Microsoft Office 2007

Nachher:

```
<xsl:template name="templ_prop_OpenBracket" >
    <xsl:param name="LCID" />
    <xsl:variable name="_LCID">
      <xsl:call-template name="localLCID">
        <xsl:with-param name="LCID" select="$LCID"/>
      </xsl:call-template>
    </xsl:variable>
    <!--<xsl:value-of se-
lect="/*/b:Locals/b:Local[@LCID=$_LCID]/b:General/b:OpenBracket"/>-
->
    <xsl:text>[</xsl:text>
  </xsl:template>

  <xsl:template name="templ_prop_CloseBracket" >
    <xsl:param name="LCID" />
    <xsl:variable name="_LCID">
      <xsl:call-template name="localLCID">
        <xsl:with-param name="LCID" select="$LCID"/>
      </xsl:call-template>
    </xsl:variable>
    <!--<xsl:value-of se-
lect="/*/b:Locals/b:Local[@LCID=$_LCID]/b:General/b:CloseBracket"/>
-->
    <xsl:text>]</xsl:text>
  </xsl:template>
```

Quelltext III-6: Die modifizierte Fassung fügt Zitate in eckigen Klammern in das Dokument ein.

Des Weiteren in Zeile 4446 (ohne die o.a. Änderungen ist es ca. die Zeile 4442)

Vorher:

```
<xsl:value-of select="b:RefOrder"/>
```

Nachher:

```
<xsl:text>[</xsl:text>
     <xsl:value-of select="b:RefOrder"/>
<xsl:text>]</xsl:text>
```

Damit hat man die Nummerierung im Literaturverzeichnis mit eckigen Klammern versehen, ebenso wie die Zitate im Text.

Jetzt soll die Nummerierung durch das Buch-Tag ersetzt werden, welches sich aus den ersten drei Buchstaben des Nachnames des ersten Autors und der zweistelligen Jahreszahl des Erscheinungsdatums generiert.

Ab Zeile 3650 ersetzt man:

```
<xsl:variable name="author">
     <xsl:value-of select="msxsl:node-
set($ListPopulatedWithMain)/b:Citation/b:Source/b:RefOrder"/>
</xsl:variable>
```

durch:

```
<xsl:variable name="author">
     <xsl:value-of select="msxsl:node-
set($ListPopulatedWithMain)/b:Citation/b:Source/b:Tag"/>
</xsl:variable>
```

Der Knoten „`RefOrder`“ ist in o.a. Zeile 4447 ebenfalls durch „`Tag`“ zu ersetzen.

Um den (falschen) Punkt hinter den eckigen Klammern im Literaturverzeichnis zu entfernen, muss man noch folgende Zeile auskommentieren, ca. bei Zeile 4450:

```
<!-- <xsl:call-template name="templ_prop_Dot"/>-->
```

Somit erhält man das gewünschte Format:

[Gen78] **Genzmer, Fritz und Kuchel, Jürgen H.** *Umgang mit …*

Standardmäßig sortiert Word 2007 die Quellen in der Reihenfolge der Eingabe. Im originalen numerischen Format hat der Leser so schnellen Zugriff auf die Quellen, wenn er auf verweisende Zitate stößt.

Nach den o.a. Änderungen sind die Quellen jedoch nur aufwändig zu finden. Es bietet sich an, die Sortierung so zu ändern, dass nach dem Nachnamen des Hauptautors alphabetisch aufsteigend sortiert wird. Artikel oder Internetreferenzen, die nur einer Firma oder einer Gruppe (beispielsweise „Apache Foundation“) zugeordnet werden können und keinen primären Autor besitzen, sollen am Beginn der Verzeichnisses aufgelistet werden.

Dazu ist bei Zeile 4980 und erneut ca. 20 Zeilen tiefer die Reihenfolge der Einträge wie folgt zu ändern und eine neue `Sort`-Anweisung für den Firmennamen einzufügen:

Vorher:

```
<xsl:apply-templates select="msxsl:node-set($sourceRoot)/*">
     <xsl:sort select="b:RefOrder" data-type="number"/>
     <xsl:sort se-
lect="b:Author/b:Main/b:NameList/b:Person[1]/b:Last" />
     <xsl:sort se-
lect="b:Author/b:Main/b:NameList/b:Person[1]/b:First" />
     <xsl:sort se-
lect="b:Author/b:Main/b:NameList/b:Person[1]/b:Middle"/>
     <xsl:sort select="b:Title"/>
</xsl:apply-templates>
```

Quelltext III-7: Original MS Office 2007 Stylesheet für die Sortierung des Literaturverzeichnisses nach ISO 690.

Nachher:

```
<xsl:apply-templates select="msxsl:node-set($sourceRoot)/*">
     <xsl:sort se-
lect="b:Author/b:Main/b:NameList/b:Person[1]/b:Last" />
     <xsl:sort se-
lect="b:Author/b:Main/b:NameList/b:Person[1]/b:First" />
     <xsl:sort se-
lect="b:Author/b:Main/b:NameList/b:Person[1]/b:Middle"/>
     <xsl:sort select="b:Author/b:Main/b:Corporate" />
```

```
    <xsl:sort select="b:Title"/>
    <xsl:sort select="b:RefOrder" data-type="number"/>
</xsl:apply-templates>
>
```

Quelltext III-8: Die modifizierte Fassung sortiert primär nach dem Nachnamen des ersten Autors, sekundär nach dem Namen der Firma bzw. des Verlages.

Unter Windows XP können die Operationen wie beschrieben durchgeführt werden, danach erscheinen das Literaturverzeichnis und die Zitate im gewünschten Format und in einer hilfreichen Sortierung.

Hinweis für Benutzer von Windows Vista

Die XSL-Datei liegt zwar unter Vista ebenfalls im o.a. Verzeichnis

```
C:\Program Files\Microsoft Office\Office12\Bibliography\Style
```

Es handelt sich jedoch nur um die sogenannten Kompatibilitätsdateien, die für den Betrieb von Programmen, die noch nicht an Vista angepasst wurden, angezeigt werden. Bearbeitet man die `ISO690Nmerical.xslt`, so wird man zunächst keine Änderungen bei der erneuten Zuweisung der Stilvorlage in Word bemerken. Auch das Dateidatum der XSL-Datei ändert sich nicht. Klickt man nun auf die Schaltfläche „Kompatibilitätsdateien“ in der Befehlsleiste des Vista-Explorers, so wechselt der Explorer in das Verzeichnis

```
C:\Users\[Username]\AppData\Local\VirtualStore\Program
Files\Microsoft Office\Office12\Bibliography\Style
```

und zeigt nur noch die geänderte XSL-Datei an. Diese muss jetzt manuell in das Style-Verzeichnis unter

```
C:\Program Files\Microsoft Office\Office12\Bibliography\Style
```

kopiert werden. Nach der Bestätigung der UAC-Sicherheitsabfrage von Vista wird die bearbeitete XSL-Datei von Word berücksichtigt und das Literaturverzeichnis umformatiert. Anschließend kann die Datei direkt in diesem Verzeichnis weiter bearbeitet werden.

Diskussion

Grundsätzlich ist XSLT der etablierte und anerkannte Standard für die Transformation XML-basierter Dokumente. TML-Ausdrücke in eine der geforderten Zielformate zu wandeln, wäre ohne weiteres damit möglich.

Trotzdem sprachen letztendlich mehrere Gründe dafür, für die Transformation nach XML eine eigene Transformationsengine zu entwickeln.

XSLT-Stylesheets müssen programmiert werden. Im angestrebten späteren industriellen Einsatz crossmedialer Produktionen waren häufige Änderungen an den typographischen Übersetzungen zu erwarten. Da für XSLT-Stylesheets keine geeigneten Editoren für eine einfache Anpassung existieren, wäre hier ein hoher Aufwand auf der Seite der Nutzer entstanden.

Die Geschwindigkeit der getesteten Engines schien zwar auf den ersten Blick ausreichend zu sein, sollte sich aber beim späteren praktischen Einsatz Probleme mit den Laufzeiten zeigen, so wären die Eingriffsmöglichkeiten gegenüber einem eigenen Übersetzer beschränkt gewesen.

Kompliziertere Transformationen wären zudem mit XSLT nicht realisierbar gewesen, da die Mächtigkeit nicht mit einer ‚normalen' Programmiersprache vergleichbar ist. Wie sich später herausstellte, ließen sich nicht wohlgeformte, ‚unsaubere' Formate wie Quark XPress nur mit aufwändiger Programmierung in ein XML-Format übersetzen. Das wäre mit XSLT-Stylesheets nicht möglich gewesen.

Kapitel 9

ELECTRONIC PUBLISHING MIT XSL-FO

XSL-FO, auch kurz mit FO bezeichnet, ist die Formatierungssprache, die vom W3C für XML definiert wurde [W3C06] und für die XSLT ursprünglich entworfen wurde. FO dient der Formatierung von XML-Dokumenten zur professionellen Gestaltung von Printprodukten. FO-Dokumente sind ein indirektes Format und werden nicht direkt im Druckbereich verwendet. Für ein spezifisches Ausgabeformat werden Dokumente aus der FO-Zwischendarstellung mit einem FO-Prozessor in ein für den Druck geeignetes Format transformiert.

Bis heute existiert kein Programm, mit dem ein naiver Anwender FO-Dokumente direkt editieren und anzeigen könnte. Zudem ist das Vokabular von XSL-FO sehr umfangreich, es enthält unter anderem Tags für folgende Eigenschaften:

- Regionen, Ränder und Bereiche einer Seite
- Breite und Höhe von Seiten
- Abfolge von Seiten
- Seitennummerierung
- Rahmen, Abständen, Spalten und Blöcke
- Absätze, Listen und Tabellen
- Textformatierung
- Linien, Bilder und andere Objekte

Aus diesem Grund werden FO-Dokumente automatisiert mit Hilfe von XSLT aus XML-basierten Quelldokumenten abgeleitet. Aus dieser Repräsentation lassen sich verschiedene Ausgabeformate wie PDF, RTF, HTML oder auch eine Sprachausgabe erzeugen. XSL-FO ermöglicht also einen Weg zum *Multi-Platform-Publishing*.

Das Formatierungsmodell von XSL-FO arbeitet mit rechteckigen, ineinander schachtelbaren Bereichen. Dabei wird zwischen Block-Bereichen und Inline-Bereichen unterschieden. Block-Bereiche werden durch Zeilenumbrüche voneinander getrennt. Bei Inline-Bereichen ist dies nicht der Fall. Diese Unterscheidung wirkt sich auf die endgültige Formatierung im FO-Prozessor aus.

Struktur eines XSL-FO-Dokuments

Bei XSL-FO handelt es sich um ein XML-Format. Das Wurzel-Element eines FO-Dokuments heißt `root` und hat die Kind-Elemente `layout-master-set`, `declarations` und `page-sequence`. Ein reales FO-Dokument muss über genau ein `layout-master-set`-Element und mindestens ein `page-sequence`-Element verfügen. `declarations` ist optional.[Har041]

Ein `layout-master-set` kann mehrere `simple-page-master` und `page-sequence-master` enthalten. In diesen Elementen werden grundlegende Formatierungseinstellungen für Seiten festgelegt, darunter auch Seitengröße, Abstände des Inhaltsbereichs vom Rand und die Richtung, in der Seitenelemente in Seiten eingefügt werden.

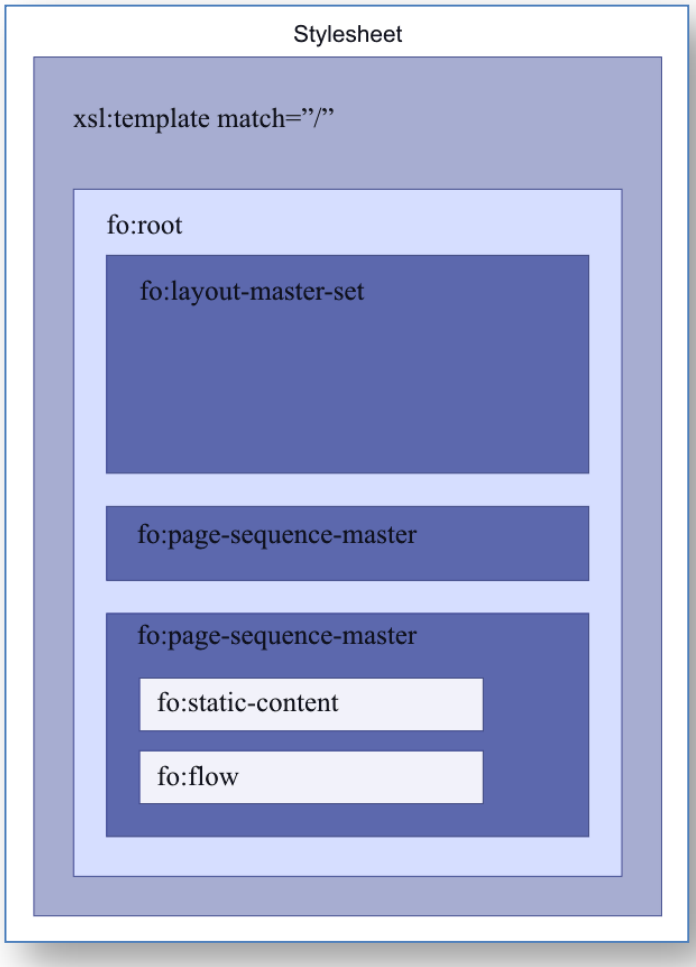

ABBILDUNG III-4: GRUNDSTRUKTUR EINES XSL-FO DOKUMENTS.

Der eigentliche Inhalt eines Dokuments ist innerhalb von `page-sequence` abgelegt. Die Formatierung einer Seite hängt dabei von der gewählten Musterseite ab, vergleichbar

der Musterseitenvorlagen in Präsentations- oder Desktop Publishing-Programmen. Eine Musterseite wird durch das `master-reference`-Attribut in einem `page-sequence`-Element angegeben.

Man kann Bereiche mit XSL-FO exakt platzieren, aber wenn man keine exakten Platzierungsinformationen angibt, entscheidet der FO-Prozessor, an welcher Stelle ein Bereich platziert wird[54]. Im Gegensatz zu Musterseiten müssen sichtbare Seiten nicht explizit angelegt werden. Auch hier entscheidet der FO-Prozessor selbständig über das Anlegen neuer Seiten. Eine vollständige Beschreibung von XSL-FO findet sich unter [W3C06].

Von XML nach XSL-FO

XSL-FO ist eine allgemeine und komplizierte Layoutsprache. FO-Dokumente direkt zu schreiben, dürfte in der Praxis nie ein relevanter Ansatz werden. Wie oben beschrieben, gibt es ähnlich wie in LATEX zwar viele Kann- und wenige Muss-Formatierungen, aber bereits ein in der Formatierung relativ anspruchsloses Dokument führt zu einer umfangreichen Beschreibung.

Einfacher ist es, XML-Dokumente mit Hilfe von XSLT-Stylesheets in FO-Dokumente zu transformieren. Mit dem folgenden XSLT-Code kann das Beispieldokument in Quelltext III-1 in ein FO-Dokument überführt werden:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:fo="http://www.w3.org/1999/XSL/Format">
     <xsl:template match="/">
          <fo:root>
               <fo:layout-master-set>
                    <fo:simple-page-master master-name="only"
margin-left="2cm" margin-right="2cm" margin-top="3cm" mar-
gin-bottom="3cm">
                         <fo:region-body />
                    </fo:simple-page-master>
               </fo:layout-master-set>
               <fo:page-sequence master-reference="only">
                    <fo:flow flow-name="xsl-region-body">
                         <xsl:apply-templates />
                    </fo:flow>
               </fo:page-sequence>
          </fo:root>
     </xsl:template>
     <xsl:template match="bericht">
          <fo:block margin-bottom="0.5cm">
               <xsl:apply-templates />
          </fo:block>
     </xsl:template>
     <xsl:template match="melder">
```

[54] Diese Arbeitsweise ist mit LATEX vergleichbar. Durch einen Seitenrenderer wird dem Anwender die Formatierung und Gestaltung einer Seite abgenommen, wenn er nicht explizit eingreift.

```
          <fo:block>
Melder: <xsl:apply-templates />
          </fo:block>
     </xsl:template>
     <xsl:template match="datum">
          <fo:block>
Datum: <xsl:apply-templates />
          </fo:block>
     </xsl:template>
     <xsl:template match="fehlerkategorie">
Kategorie: <fo:inline font-weight="bold"><xsl:apply-
templates /></fo:inline>
     </xsl:template>
     <xsl:template match="beschreibung">
          <fo:block>
Beschreibung: <xsl:apply-templates />
          </fo:block>
     </xsl:template>
</xsl:stylesheet>
```

Quelltext III-9: Das XSLT-Stylesheet zur Transformation des Fehlerberichts in ein XSL-FO-Dokument.

Im Stylesheet können neben den reinen Format- und Strukturtransformationen die Quelldokumente auch semantisch angereichert werden. In diesem Beispiel wurden die Wörter Melder, Datum, Kategorie und Beschreibung, jeweils gefolgt von einem Doppelpunkt und einem Leerzeichen ergänzt. Das resultierende FO-Dokument sieht wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
     <fo:layout-master-set>
          <fo:simple-page-master master-name="only"
          margin-left="2cm" margin-right="2cm"
          margin-top="3cm" margin-bottom="3cm">
               <fo:region-body/>
          </fo:simple-page-master>
     </fo:layout-master-set>
     <fo:page-sequence master-reference="only">
          <fo:flow flow-name="xsl-region-body">
               <fo:block margin-bottom="0.5cm">
                    <fo:block>
Melder: Hans Mustermann
                    </fo:block>
                    <fo:block>
Datum: 2008-06-02
                    </fo:block>

Kategorie: <fo:inline font-weight="bold">GUI-
Fehler</fo:inline>
                    <fo:block>
Beschreibung:
          Die Zeitanzeige verschwindet teilweise hinter dem
          rechten Rand des Hauptfenster, wenn das Fenster
          verschoben wird.
```

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

Quelltext III-10: Das XSL-FO Dokument nach der Transformation.

Von XSL-FO nach PDF

Wie oben beschrieben, muss zur eigentlichen Druckausgabe eines Dokumentes aus der Zwischendarstellung in XSL-FO erst ein druckfähiges Format erzeugt werden. Dies geschieht mit Hilfe eines XSL-FO-Prozessors. Eine häufig genutzte Variante ist der Apache *Formatting Objects Prozessor* (FOP) [Apa07].

FOP ist Java-basiert und implementiert derzeit eine große Teilmenge der XSL-FO-Spezifikation in der Version 1.1. Zu den unterstützen Ausgabeformaten zählen die im Printbereich klassischen Formate PDF und Postscript, aber auch neuere Formate wie SVG. Für das zuvor generierte FO-Dokument (Quelltext III-10) liefert FOP mit folgendem Aufruf das PDF-Dokument „`bericht.pdf`“:

```
fop bericht.fo bericht.pdf
```

Alternative FO-Prozessoren sind beispielsweise die kommerziellen Lösungen Antennahouse XSL Formater[55] und RenderX XEP Engine[56]. Eine weitere interessante Alternative sind die TeX-Makros „PassiveTeX“[57], die zum Rendern die bewährte TeX-

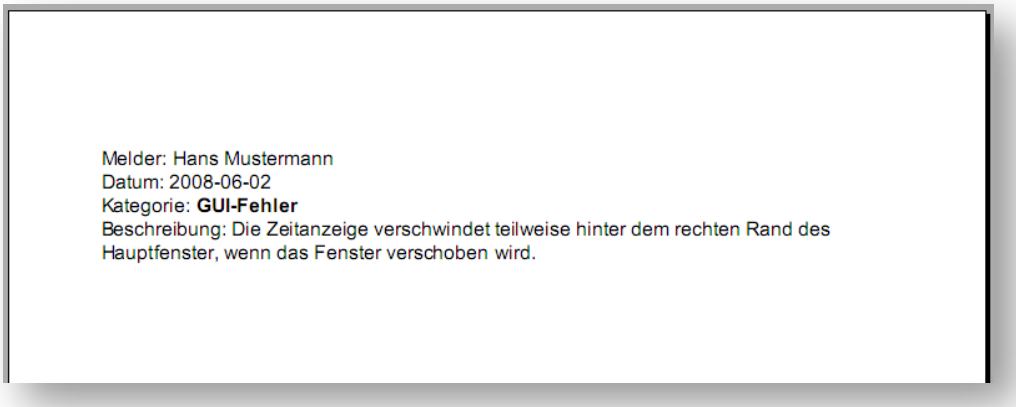

ABBILDUNG III-5: DAS ERGEBNIS DES FEHLERBERICHTS ALS PDF-DOKUMENT.

[55] http://www.antennahouse.com

[56] http://www.renderx.com/tools/xep.html

[57] http://www.tei-c.org.uk/Software/passivetex/

Engine verwenden.

Erläuterung des Beispiels

Für das Beispiel des Fehlerberichtes wurde ein Seiten-Master `simple-page-master` mit dem Namen „`only`“ definiert. Der Seiten-Master enthält rudimentäre Angaben für die Seitenformatierung mit Angabe der Randgrößen.

Anschließend folgt die Seiten-Sequenz `page-sequence`, welche in diesem kurzen Beispiel aus nur einer Seite besteht. Die Seiten-Sequenz enthält eine Referenz auf den Seiten-Master, welcher der Seite zu Grunde liegen soll („`only`“). Es folgt der erste „Flow“, also der erste Ausgabeteil. Im „`Flow`“-Tag wird der Seitenbereich angegeben, welcher den Flow enthalten soll. Hier ist dies „`xsl-region-body`“. Zum Schluss werden die Inhalte in einzelnen Blöcken ausgegeben.

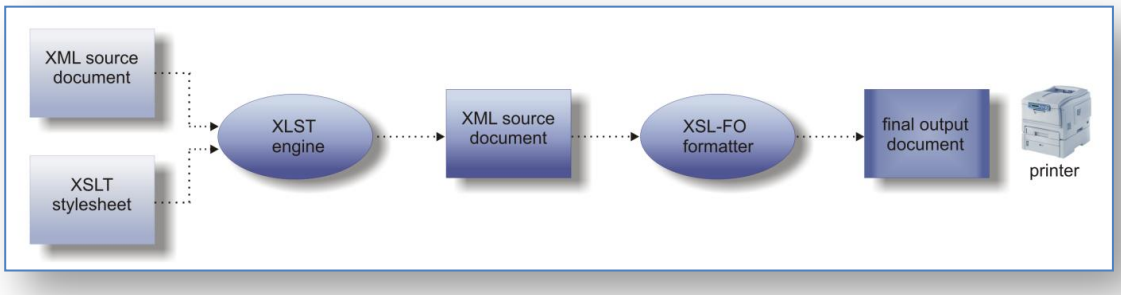

ABBILDUNG III-6: DIE PROZESSKETTE VOM XML-DOKUMENT ÜBER DIE ZWISCHENDARSTELLUNG IN XSL-FO BIS ZUM DRUCKFÄHIGEN DOKUMENT.

Diskussion

XSL-FO ist der XML-Standard zur typographischen Auszeichnung bzw. Transformation von Inhalten für die Druckausgabe. Für die zugrunde liegende Anwendung war das FO-Sprachkonstrukt jedoch zu mächtig[58]. Zudem hätten Fehler im Aufbau nicht ohne größeren Aufwand behoben werden können. Des Weiteren kommt hinzu, dass zur Eingabe im WYSIWYG-Editor eine weitere Meta-Sprache hätte entwickelt werden müssen, da das Sprachkonstrukt von XSL-FO zur Eingabe durch Endanwender zu kompliziert ist.

[58] Eine höhere Mächtigkeit ist im Normalfall von Vorteil, es müssen ja nicht alle Anweisungen genutzt werden. Der Dokument-Container bei XSL-FO muss jedoch immer benutzt werden, es wird immer die Einheit mindestens einer Seite beschrieben.

Ein weiteres Problem waren Berichte in diversen Internet-Foren von unterschiedlichen Ausgaben, die unterschiedliche XSL-FO Prozessoren aus identischen FO-Dokumenten erzeugen. Für den industriellen, kommerziellen Einsatz in Verlagen ist dieses Risiko zu hoch. Es war abzusehen, dass bis auf weiteres die als Quasi-Standards geltenden Render-Engines von Quark und Adobe für die Ausgabe der Dokumente verwendet werden würden.

Auch die Ermittlung von Laufzeiten brachte ernüchternde Ergebnisse. Für ein Minimal-Dokument wie aus Quelltext III-10 dauert die Übersetzung nach PDF bereits ca. 5 Sekunden auf einem Rechner mit Single-Core CPU und 1,6 GHz. Mehrseitige Dokumente benötigten immer mehr als 10 Sekunden. Diese Laufzeiten sind für den dynamischen Einsatz in einer crossmedialen Produktionsumgebung deutlich zu lang.

Kapitel 10

TRANSFORMATION MIT DSSSL

Die bisher vorgestellten Transformationsverfahren bewegen sich im Kontext von XML. Basis von XML ist jedoch SGML und auch dort bestand schon das Bedürfnis nach Transformationen zwischen SGML-Anwendungen.

DSSSL ist ein Akronym für *Document Style Semantics and Specification Language*[Cla98] und wird „Dissel" ausgesprochen. DSSSL ist eine Sprachfamilie, mit der sowohl Transformationen als auch Formatierungen vorgenommen werden können. Im Gegensatz zu den bisher vorgestellten Seitenbeschreibungssprachen ist DSSSL nicht in XML notiert. Anweisungen in der Sprache DSSSL sind Programme aus einer funktionalen Programmiersprache, die auf SCHEME basiert. Da DSSSL ursprünglich für SGML konzipiert wurde, kann es auch auf XML-Quelldokumente angewendet werden.

Die Anwendung der DSSSL-Stylesheets erfolgt durch einen DSSSL-Prozessor. Die Implementation JADE[59] von James Clark (James' DSSSL Engine) wird seit fast 10 Jahren nicht mehr weiterentwickelt und lässt sich nur mit großem Aufwand auf aktuellen Linux-Distributionen übersetzen. Besser geeignet ist OpenJade[60], das von der DSSSL-Community weiterentwickelt wurde und für die meisten Linux-Distributionen als fertiges Binary zur Verfügung steht.

Beim Ausführen einer Transformation mit DSSSL wird keine Zwischendarstellung wie bei XSL-FO erzeugt, es wird direkt das gewünschte Zielformat[61] erzeugt.

DSSSL-Stylesheets

Die Idee beim Aufbau von DSSSL-Stylesheets ist mit XSLT-Stylesheets vergleichbar, die Syntax entspricht aber den applikativen Ausdrücken in SCHEME. Für jedes SGML- bzw. XML-Element des Quelldokuments, für das man in einem XSLT-Template den Code

```
<xsl:template match="tag_name_x">
     instruction_for_tag_name_x
</xsl:template>
```

[59] Siehe: http://www.jclark.com/jade/

[60] Siehe: http://openjade.sourceforge.net/

[61] Beispielsweise HTML, RTF, mit weiteren Tools auch PDF

einsetzen würde, trägt man in das DSSSL-Template einen Eintrag der Form

```
(tag_name_x instruction_for_tag_name_x)
```

ein. DSSSL-Stylesheets sind selbst SGML-Dokumente.

DSSSL und XSL-FO verwenden eine unterschiedliche Struktur der Anweisungen hinter `instruction_for_tag_name_x`. XSL-FO-Dokumente erzeugt man mit Hilfe von XSLT, in dem Teile des Quelldokuments und frei definierbare Zeichenketten beliebig konkateniert werden. Im XSL-FO-Beispiel der Fehlerberichte wurden die einzelnen Teilangaben (Melder, Datum, Kategorie, usw.) gefolgt von Doppelpunkt und Leerzeichen als frei definierbare Strings benutzt. Der XSL-FO-Code wurde identisch zur späteren Ausgabe im FO-Dokument manuell im Stylesheet eingetragen.

In DSSSL-Stylesheets können ebenfalls frei definierbare Strings benutzt werden. Den Auszeichnungscode formuliert man allerdings nicht direkt, sondern in Form von *flow objects*, auch *sosofos* genannt [Pre97]. Nach dem Motto der Computerindustrie - keine Technologie ohne Akronym – hat auch *sosofo* eine beschreibende Langfassung: „*Specification Of a Sequence Of Flow Objects*" und ist tatsächlich sehr aussagekräftig.

Die flow objects entsprechen in etwa den formatting objects von XSL-FO. Um das Element `tag_name_x` als Absatz zu formatieren, wäre der Eintrag

```
(element tag_name_x (make paragraph))
```

notwendig. Markus Reinsch von der Universität Bielefeld hat ein paar schöne Java-Animationen[62] erstellt, welche die Prozesse in einer DSSSL-Engine darstellen und leichter verständlich machen.

Mit DSSSL nach RTF

Das folgende Beispiel der bereits oben verwendeten Fehlerberichte kann mit OpenJade nachvollzogen werden. Es wurde ein DSSSL-Stylesheet erstellt, dessen Ausgabe dem XSLT-Stylesheet für XSL-FO möglichst nahe kommt:

```
<!doctype style-sheet PUBLIC "-//OpenJade//DTD DSSSL Style
Sheet//EN">

(element berichte
     (make simple-page-sequence left-margin: 2cm header-
margin: 3cm)
)
```

[62] Siehe: http://coli.lili.uni-bielefeld.de/Texttechnologie/dsssl_ma/dsssl/index.html

```
(element bericht
     (make paragraph (process-children))
)

(element melder
     (make sequence
          (make paragraph-break)
          (literal "Melder: ")
          (process-children)
     )
)

(element datum
     (make sequence
          (make paragraph-break)
          (literal "Datum: ")
          (process-children)
     )
)

(element kategorie
     (make sequence
          (make paragraph-break)
          (literal "Kategorie: ")
          (make sequence font-weight: 'bold
          (process-children)
          )
     )
)

(element beschreibung
     (make sequence
          (make paragraph-break)
          (literal "Beschreibung: ")
          (process-children)
     )
)
```

Quelltext III-11: Das DSSSL-Stylesheet für die Fehlerberichte

Mit dem Konsolenbefehl

```
openjade –t rtf –d bericht.dsl bericht.xml
```

wird die XML-Quelldatei in ein RTF-Dokument transformiert. Eine eventuell vorhandene DTD kann hinter dem DSSSL-Stylesheet (bericht.dsl) angegeben werden.

Erläuterung des Beispiels

Im direkten Vergleich mit dem XSLT-Stylesheet lässt sich eine vergleichbare grundsätzliche Struktur erkennen. Nach der Definition einer Seitenvorlage wird für die einzelnen

Elemente des XML-Quelldokumentes jeweils eine Transformationsvorschrift angegeben. Für Anwender, die bisher keine Erfahrung mit funktionalen Programmiersprachen machen konnten, ist die Syntax eventuell weniger intuitiv. Man kann sich mit dem applikativen Paradigma der funktionalen Programmiersprachen helfen: Eine Funktion wird auf dem Argumente angewendet. Im Gegensatz zur ‚normalen' Infixschreibweise von Operatoren und Argumenten werden die Funktionsaufrufe in Präfixnotation notiert. Im Folgenden ein einfaches Beispiel mit einem arithmetischen Ausdruck und seinen zweistelligen Operatoren:

Beispiel:

Infix		Präfix
`(6 + (18 - 6)`	⇔	`(+ 6` `(- 18 6))`

Eine weitere Besonderheit sind Konkatenationen: In XSLT-Stylesheet schreibt man die Bausteine, die zusammengesetzt werden sollen, einfach nacheinander. In DSSSL werden Konkatenationen explizit mit `(make sequence)` definiert. Die Argumente (=die zu konkatenierenden Teile) folgen als geklammerte Ausdrücke, die wiederum konkateniert und geschachtelt sein können.

Diskussion

Mit DSSSL kann in einem Durchgang transformiert und formatiert werden. Im direkten Vergleich fasst es die Technologien XSLT und XSL-FO zusammen.

DSSSL wurde ursprünglich als universelle Sprache für Transformationen und für layouttechnische Beschreibungen von SGML-Daten konzipiert. Die Spezifikation von DSSSL ist mächtig und umfangreich [Bos96]. DSSSL-Stylesheets werden in einer funktionalen Sprache programmiert, der Ersteller eines solchen Stylesheets muss also entsprechende Kompetenzen vorweisen können.

Vermutlich haben diese beiden Tatsachen und die Popularität von XML und XSL dazu geführt, dass die Weiterentwicklung von DSSSL seit dem Ende der 1990er Jahre stehen geblieben ist. Heutige Anwendungsfälle beschränken sich im Wesentlichen auf Transformationen von Docbook-Dokumenten[63].

Tatsächlich sind die Links auf viele DSSSL-Quellen (beispielsweise [Cla98]) ungültig und viele Seiten nicht mehr auffindbar. Für eine zukunftsweisende Anwendung im industriellen Umfeld ist DSSSL sicher nicht (mehr) eine geeignete Technologie.

[63] Siehe: http://docbook.org/

Kapitel 11

TRANSFORMATION MIT COST/TCL

Eine weniger verbreitete Technologie für die Verarbeitung formatfreier SGML- und XML-Daten ist das *Copenhagen SGML Tool* (Cost) [Eng03].

Cost basiert auf der Skriptsprache Tool Command Language (Tcl) und wurde ursprünglich für die Verarbeitung von SGML-Dokumenten entwickelt. Cost kann somit auch die daraus abgeleiteten XML-Dokumente verarbeiten.

Cost kann entweder mit dem package-Befehl dynamisch in eine Tcl-Applikation geladen oder statisch mit dem Tcl-Interpreter verlinkt werden. Für einfache Shellskripte oder interaktive Sitzungen kann auch das Konsolen-Interface *costsh* verwendet werden. Cost enthält ein event-gesteuertes Programminterface und eine Abfragesprache, um die Knoten des Quelldokumentes abzufragen. Für die Anwendung von Cost ist ein SGML-Parser notwendig. Da sich der ursprünglich verwendete Parser „nsgmls“ von James Clark [Cla98] mit vertretbarem Aufwand nicht mehr übersetzen lässt, bietet sich das Paket OpenSP an, welches zusammen mit OpenJade installiert wird.

Beispiel:

Als Ausgangsdokument dient wieder der XML-Fehlerbericht, siehe Quelltext III-1. Mit Hilfe einer kleinen Cost-Applikation sollen diese XML-Daten in ein reines Textformat für eine E-Mail transformiert werden.

```
require Simple.tcl
specification translate {
     {element MELDER} {
          prefix "\n----------------------------------------
-\nFehlermeldung von: \t"
          suffix "\n"
     }
     {element DATUM} {
          prefix "\nDatum: \t\t"
          suffix "\n"
     }
     {element FEHLERKATEGORIE} {
          prefix "\n----------------------------------------
-\nKategorie: "
          suffix "\n----------------------------------------
-\n"
     }
     {element BESCHREIBUNG} {
          prefix "\nBericht:\n"
          suffix "\n----------------------------------------
-\n"
     }
}
```

Quelltext III-12: Cost-Applikation zum Erzeugen des E-Mail-Textes

Dieses Beispiel ist leicht verständlich. Jeder `element`-Block bezieht sich auf ein Element des XML-Dokuments. Innerhalb des Blocks wird der Inhalt des Elementes ausgegeben, mit `prefix` wird ein Vorspann und mit `suffix` ein Nachspann für jeden Abschnitt definiert. Daraus resultiert folgende Ausgabe:

```
----------------------------------------
Fehlermeldung von:      Tester MvH


Datum:                  2007-02-11


----------------------------------------
Kategorie: GUI-Fehler
----------------------------------------


Bericht:
Es erscheint eine Fehlermeldung, wenn der Dialog zur Eingabe
der Kalibrierwerte mit "Abbrechen" wieder geschlossen wird.

----------------------------------------
```

Fügt man in prefix und suffix HTML-Tags ein, so kann die Ausgabe auch leicht für einen Browser oder für eine HTML-Mail formatiert werden.

Diskussion

Cost bietet weit mehr Möglichkeiten, als im o.a. Beispiel sichtbar werden. Es enthält primitive Operationen, mit denen vollständige SGML-Applikationen erstellt werden können. Der Umfang der Abfrage- und Event-Kommandos ist überschaubar[64] und lässt sich leicht erlernen. Mit Cost könnten aus XML-Daten auch die externen Formate erzeugt werden, die in dieser Arbeit berücksichtigt werden. Aufgrund der fehlenden Formatabstraktionen würde im Kontext dieser Arbeit für jedes Format ein eigenes ‚Programm' notwendig. Zudem wäre der Rückweg aus Formaten, die nicht auf XML oder SGML basieren nicht möglich. Auch auf die Verwendung von Parsergeneratoren und anderen XML-Tools hätte man verzichten müssen. Zum Beginn der Forschungsarbeiten 2002 war das Projekt noch aktiv, danach hat es bis Ende 2008 auf der Cost-Homepage keine Einträge mehr gegeben.

[64] Siehe: http://www.flightlab.com/cost/manual.html

Teil IV

TML

(The Typographic Markup Language)

Kapitel 12

TML-PROTOTYP

Im folgenden Kapitel wird die Konzeption und die prototypische Umsetzung der Transformationsengine und die dabei zugrunde liegende Formatspezifikation der typographischen Markup-Sprache TML erläutert. Entwicklungsziel des TML-Prototyps war die Möglichkeit, einzelne Absätze aus den Quelldokumenten der beteiligten Applikationsformate per Shell-Skript nach TML und von TML in ein beliebiges Format zurück zu übersetzen. Von den funktionalen Eigenschaften sollten möglichst viele der in Kapitel 6 erläuterten Anforderungen erfüllt werden. Durch die Ausrichtung auf einen praxistauglichen Einsatz im industriellen Umfeld waren die Anforderungen an die Wartbarkeit und Fehlertoleranz höher, als bei rein akademischen Prototypen. Dort soll im Allgemeinen nur die prinzipielle Korrektheit des Ansatzes gezeigt werden, hier war ein funktionsfähiges und praxistaugliches Gesamtsystem gefordert.

Der TML-Prototyp wurde in den Jahren 2004 bis 2006 von bis zu sechs Entwicklern entworfen und implementiert, dabei entstanden ca. 25.000 Zeilen Code, 11 Perl-Module mit über 100 Methoden. Aufgrund dieses Umfangs werden im Folgenden nur grundlegende technische Aspekte und algorithmisch besonders interessante Lösungsansätze und Bereiche beschrieben. Besonders interessant ist die Schema-Definition von TML, die Bausteine für die syntaktische Analyse der Textabschnitte und die Persistenz der Transformationsregeln, also der Grammatiken.

Wie in Abbildung IV-1 gezeigt, besteht der TML-Prototyp im Wesentlichen aus vier diskreten Bestandteilen:

- TML Transformation Engine:
    - Die Kernkomponente mit den sprachspezifischen Übersetzern für die interne Sprache TML und für die externen Sprachen.
- Database Encapsulation:
    - Abstrahiert die Anwendungslogik von der Persistenzlogik (Data Centric Tier)
- TML-Schema Definition:
    - Definiert den Aufbau des TML-Dokumenttyps und erlaubt die Gültigkeitsprüfung der internen TML-Speicherformate
- Web Application:
    - Einfache Endanwender-gerechte HTML-Oberfläche zur Pflege der Grammatiken, Sonderzeichen-Übersetzungen und weiterer Konfigurationen.

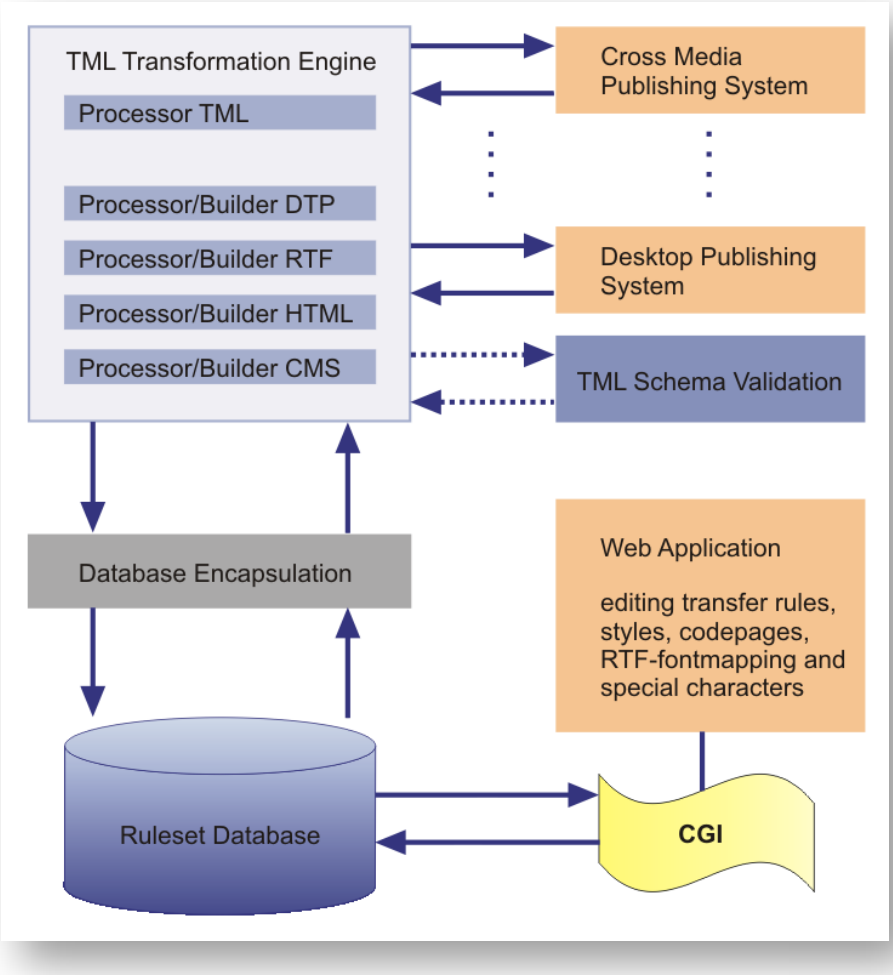

ABBILDUNG IV-1: HAUPTKOMPONENTEN DES TML-PROTOTYPEN

Um eine reibungslose Transformation der typographischen Informationen aus den unterschiedlichen externen Formaten zu gewährleisten, wurde für die Persistenz ein eigenes zentrales Datenformat entwickelt. Dieses Format sollte mächtig genug sein, die typographischen Informationen aus allen anderen Formaten in sich aufnehmen zu können oder in geeigneter Weise zu kapseln. Auch sollte es möglichst gut für Menschen lesbar und editierbar sein.

Während der Überlegungen zur Frage des zentralen Speicherformats wurden einige Formate und Definitionen diskutiert. Content Management Systeme verwenden häufig selbst definierte Tags, die mit einer im normalen Gebrauch selten benutzten Zeichenkombination geklammert werden. Damit wird das umständliche Maskieren der Tags vermieden. Im InterRed-Redaktionssystem existiert dazu ein eigenes Format namens

DefTags[65]. Da allerdings mit den DefTags die Eingabe von überlappenden Auszeichnungsformaten im WYSIWYG-Editor nicht möglich war, wurde gegen die weitere Entwicklung auf der Basis von DefTags entschieden. Zudem sollten die Muster der Token einem standardisierten Format folgen.

Als Austausch- und Speicherformat für die Typographischen Metadaten wurde demzufolge ein XML-basiertes Format angestrebt. Nahezu alle Lösungen, Verfahren oder Standards im Bereich der Medienbrüche und der Transformationen in einer Mediengattung basieren auf XML. Zudem ist mit einem XML-basierten Format eine einfache Validierung der erzeugten Dokumente möglich. In der praktischen Anwendung zeigt sich, dass der oft versprochene Vorteil durch die Verwendung von „XML-Standards" geringer als erwartet ausfällt. Allgemein gesagt, ist die Berücksichtigung von XML nicht viel mehr als eine Übereinkunft über die verwendete Nomenklatur und die Beschreibung des Tokenformats. Daraus resultieren als Hauptvorteile die Möglichkeit, große Teile der Prozesskette mit allgemein bekannten Standardwerkzeugen zu behandeln und die vergleichsweise einfache Anpassung zwischen unterschiedlichen Formaten.

Das zu entwickelnde Format sollte leicht zu pflegen sein und eine möglichst große Schnittmenge mit HTML besitzen, um den bereits angesprochenen Forderungen nach einfacher Lesbarkeit und Editierbarkeit entgegen zu kommen. Die Namen der Elemente sollten ausschließlich aus der englischen Sprache stammen. Da das Format speziell für typographische Informationen entwickelt wurde, bekam es den Namen TML[66].

Zur Definition der Auszeichnungen von TML wurde für die Verwendung von XML Schema und gegen die von DTD entschieden, da es die derzeitige Empfehlung des W3C ist. Schema umfasst zudem mehr Datentypen, erlaubt die Nutzung eigener Datentypen und ermöglicht die Inhaltsbeschränkung von Elementen. Auch ist ein XML-Schema selbst wieder ein XML-Dokument und kann daher mit XML-Tools verarbeitet werden. Des Weiteren sind durch die eindeutige Darstellung und seine saubere interne Struktur Fehler leichter aufzufinden.

TML-Schema

TML ist wie schon erwähnt eine XML-basierte Auszeichnungssprache und stellt das Datenhaltungsformat für die gesamte Transformationsengine dar. Prinzipiell sollte TML daher alle Typographieauszeichnungen aller externen Formate unterstützen. Wie schon in Kapitel 6 erwähnt, sollte der TML-Prototyp nur das Gegenstück zum Inhalt eines Absatzelementes[67] bilden können, die Transformation von Headerdefinitionen[68] war zunächst nicht geplant.

---

[65] Beispiel für das DefTag zur Fettung von Text: *$(fett:dieser Text wird fett geschnitten)$*

[66] *Typographic Markup Language*

[67] beispielsweise DTP-Textbox, HTML- , etc.

[68] wie beispielsweise Stildefinitionen in RTF

Mit Hilfe einer entsprechenden XML Schema-Datei kann die Menge der unterstützen Auszeichnungen einfach definiert und die erzeugten XML-Dokumente auf Validität überprüft werden. Im Kontext der geplanten Verwendung in Produktivsystemen kann auf diese Weise eine generelle Grundmenge von Auszeichnungen pro Kundensystem vorgegeben werden. Da die Administration der Transformationsregeln in größeren Verlagen auf Abteilungs- bzw. Redaktionsebene erfolgen würde, können auf diesem Weg Formatierungen verhindert werden, welche den Unternehmensrichtlinien widersprechen.

In webbasierten Redaktionssystemen wird zumeist ein HTML-basierter Editor[69] eingesetzt, der üblicherweise ein eingeschränktes WYSIWYG erlaubt. Es war zu erwarten, dass zwischen TML und diesem Editor besonders häufig Transformationen durchzuführen sein würden. Aufgrund der effizienteren Anbindung an einen HTML-Editor werden im TML-Schema überwiegend (X)HTML-ähnliche Bezeichner verwendet. Daneben sind diese Bezeichner ausschließlich in englischer Sprache und überwiegend einprägsam und weitestgehend selbsterklärend.

TML-Schema ist in drei logische Gruppen für Zeichen-, Absatz- und Spezialformatierungen[70] unterteilt, welche alle wiederum einer Obergruppe[71] angehören (siehe Abbildung IV-2). Das Wurzelelement des Schemas heißt `typoml`. Es kann alle Elemente der Gruppe `typography` als Nachfolger enthalten, also alle Elemente außer sich selbst. Die drei logischen Gruppen können Elemente in beliebig tiefer Verschachtelung enthalten. Die einzige Ausnahme stellt das `<br>`-Tag dar, welches keine untergeordneten Elemente enthalten darf.

Beispielsweise ist das `<font>`-Tag für die Auszeichnung einer Teilmenge von Zeichen eines Absatzes zuständig und gehört somit der Gruppe `characterstyle` an. Es kann drei verschiedene Attribute enthalten:

- `face` mit einem Wert vom Typ `string` zur Definition einer Schriftart
- `size` mit einem Wert vom Typ `integer` zur Definition der Schriftgröße
- `cpg` mit einem Wert vom Typ `string` zur Definition der zugehörigen Codepage.

Folgender Auszug aus dem XML Schema für die TML zeigt beispielhaft die zugehörige Definition des <font>-Tag:

```
<xs:element name="font">
     <xs:complexType mixed="true">
          <xs:complexContent>
               <xs:extension base="typographytype">
                    <xs:attribute name="face"
```

[69] Beispielsweise „TinyMCE“, siehe http://tinymce.moxiecode.com/

[70] Gruppen characterstyle, paragraphstyle und special

[71] *typography*

```
type="xs:string"/>
                    <xs:attribute name="size"
type="xs:integer"/>
                    <xs:attribute name="cpg"
type="xs:string"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
</xs:element>
```

Quelltext IV-1: Definition des <font>-Tag im TML-Schema

Eine Übersicht der im TML-Prototyp umgesetzten Tags ist im Anhang zu finden.

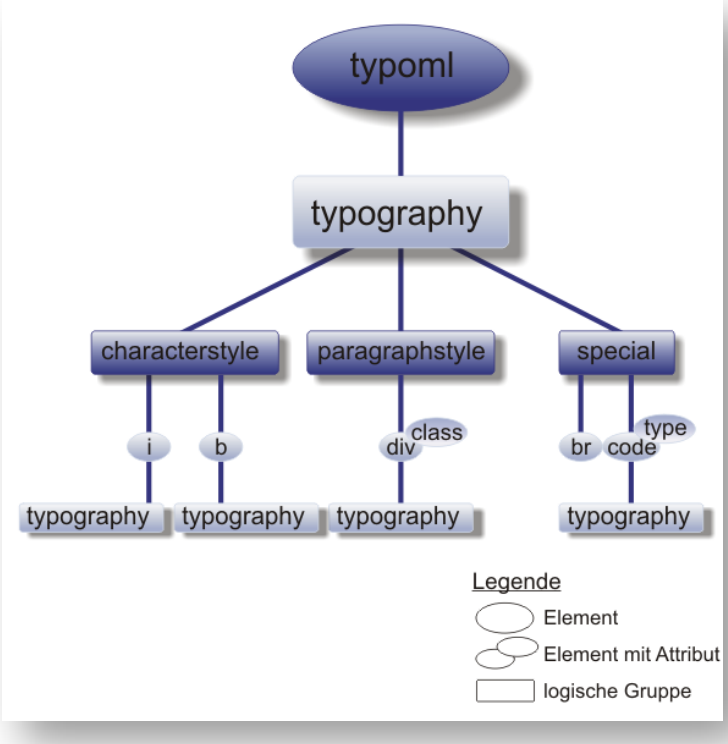

ABBILDUNG IV-2: LOGISCHE STRUKTUR DER TML-SCHEMA DEFINITION IM PROTOTYPEN

Im TML-Schema werden zusätzlich zur Deklaration von Elementen und ihren Attributen notwendige Attribute als `required` definiert, sowie Attributwerte auf bestimmte Typen oder, sofern sinnvoll, sogar auf eine Auswahl von Werten beschränkt. Vor einer Prüfung auf Validität werden den TML-Fragmenten automatisch Header- und Footerinformationen hinzugefügt und anschließend wieder entfernt.

Die Validierung erfolgt in zwei Schritten. In einem ersten Schritt wird die Wohlgeformtheit des Dokumentes geprüft. Ein Dokument gilt als wohlgeformt, wenn zu jedem Start-Tag ein passendes End-Tag existiert und das Dokument keine ungültigen Verschachte-

lungen von Tags enthält. Im zweiten Schritt wird das Dokument gegen das XML Schema validiert. An dieser Stelle wird unter anderem überprüft, ob die verwendeten Tags und Attribute gültig sind, die Positionierung der Tags erlaubt ist und die Werte der Attribute den Vorgaben entsprechen. Ein Dokument ist gültig für ein Schema, wenn es von Struktur und Inhalt mit den Schema-Spezifikationen übereinstimmt.

In Kapitel 6 wurde die Problematik der überlappenden Tags an einem Beispiel dargestellt. Um solche Überlappungen in XML zu speichern, müssen diese vorher in eine überlappungsfreie Form umgewandelt werden. Dabei werden die „störenden" Tags geschlossen, um sie gleich nach Beendigung der Überlappung wieder zu öffnen. Aus einer Überlappung werden also mehrfache Schachtelungen gemacht:

Das Beispiel

```
„Test von überlappenden Tags:<b>Hier beginnt fett und <i>hier kur-
siv. Hier endet fett</b> und hier kursiv.</i>Das war´s."
```

würde damit zu

```
„Test von überlappenden Tags:<b>Hier beginnt fett und
</b><i><b>hier kursiv. Hier endet fett</b> und hier kursiv.</i>Das
war´s."
```

Wie bei der Klammerung arithmetischer Operatoren kann die Wandlung in eine überlappungsfreie Form entweder linksassoziativ oder rechtsassoziativ erfolgen. Beispielsweise kann $a * b * c$ entweder als $((a * b) * c)$ oder als $(a * (b * c))$ interpretiert werden. Wir sagen das " $*$ " linksassoziativ ist, wenn die Teilausdrücke von links nach rechts in der Form der folgenden Klammerung erscheinen: $((a * b) * c)$. Wir nennen einen Operator rechtsassoziativ, wenn er die Teilausdrücke in der umgekehrten Richtung gruppiert: $(a * (b * c))$. [Aho73]

Die linksassoziative Umformung ergibt den folgenden Ausdruck:

```
<Tag1>Lorem ipsum </Tag1><Tag2><Tag1>dolor sit amet
</Tag1>consectetur </Tag2>
```

oder rechtsassoziativ:

```
<Tag1>Lorem ipsum <Tag2>dolor sit amet
</Tag2></Tag1><Tag2>consectetur </Tag2>
```

ABBILDUNG IV-3: ÜBERLAPPENDE TAGS IN WOHLGEFORMTER XML-STRUKTUR

Alle arithmetischen Operatoren sind linksassoziativ. Um diese gewohnte Klammerungsweise beizubehalten, wurde die Auflösung der Tag-Überlappungen ebenfalls linksassoziativ implementiert.

Mit der Auflösung der Überlappung werden HTML-Fragmente in der Chomsky-Hierarchie aus der Klasse der kontextsensitiven Sprachen in die Klasse der kontextfreien Sprachen ‚verbessert'[72].

Modellierung der Datenhaltung

Die Übersetzung der typografischen Metadaten zwischen der TML und den externen Formaten erfolgt mittels dynamischer, gesteuerter Parser[73]. Um die Parser zur Laufzeit dynamisch nicht neu erzeugen zu müssen, sind die syntaktischen Regeln durch die Grammatikproduktionen fest definiert. Andererseits war es gefordert, im praktischen Einsatz das Verhalten der Parser einfach beeinflussen zu können, da je nach Kundensystem die Anforderung an die Übersetzung der Typographie unterschiedlich sein würde. Dazu mussten die semantischen Regeln der Parser einfach zu ändern sein, um das Verhalten der Parser zur Laufzeit beeinflussen zu können.

---

[72] Das ist streng genommen nicht richtig. Klammersprachen gehören zwar zur Klasse der kontextfreien Sprachen und können mit einem Kellerautomaten erkannt werden, Tag-Klammersprachen jedoch, bei der das schließende mit dem öffnenden Tag übereinstimmen muss, gehören zur Klasse der kontextsensitiven Sprachen und können nur mit einem Linear beschränkten Automat erkannt werden. Beschränkt man allerdings die Menge der Tags und fügt die Token für diese Tags der Menge der Terminalzeichen zu und erweitert man die Grammatik um Produktionen für jedes Tag, dann können auch Tag-Klammersprachen mit Kellerautomaten erkannt werden. Näheres dazu findet der Leser in Kapitel 14.

[73] Da nicht nur die Einhaltung der korrekten Syntax geprüft wird, sondern Übersetzungen mit dem Ergebnis einer Art Zwischencodedarstellung erzeugt werden, müsste korrekterweise von Compilern gesprochen werden. Üblicherweise wird dieser Begriff im Zusammenhang mit der Übersetzung von Programmiersprachen mit dem Ziel einer Byte-Code- oder Maschinen-Code-Darstellung verwendet. Aus diesem Grund wird in dieser Arbeit der Begriff des Parsers beziehungsweise des Übersetzers verwendet.

Für die Persistenz dieser projektspezifischen Transformationsregeln[74] wurden verschiedene Ansätze untersucht. Die Regeln im Quelltext der jeweiligen Parser zu halten, hätte zwar den Vorteil eines schnellen Zugriffs gehabt, die schlechte Wartbarkeit und die fehlende konzeptionelle Trennung von Implementierung und Datenhaltung sprachen aber gegen diese Variante.

Eine weitere Möglichkeit war die Speicherung der Regeln in separaten Dateien. Problematisch hierbei ist, dass das System schreibende Zugriffsrechte auf dem jeweiligen Server benötigt und zunächst keine Struktur vorgegeben ist, obwohl sich eine tabellarische Speicherung der Regeln augenscheinlich anbietet. Die dritte Alternative war die Speicherung in einer relationalen Datenbank, hier sind die tabellarische Struktur und die benötigten Zugriffsrechte systemimmanent. Da auch das beteiligte Redaktionssystem intensiven Gebrauch von Datenbanken[75] macht und der Prototyp möglichst einfach integriert werden sollte, fiel die Entscheidung zugunsten der Regelhaltung in einer relationalen Datenbank auf Basis der frei verfügbaren MySQL-Datenbank aus. Gestützt wurde diese Entscheidung von der Tatsache, dass die zu realisierende Transformationsengine komplett in Perl programmiert werden sollte. Perl stellt über die Module *DBI* und *DBD* eine leistungsfähige und in der Praxis häufig genutzte und erprobte Schnittstelle zur MySQL-Datenbank zur Verfügung. Nachdem die Entscheidung für eine MySQL-Datenbank zur Regelhaltung gefallen war, stand das grundlegende Modell der Transformationsengine fest:

Die sprachspezifischen Parser greifen mit Hilfe des entwickelten Datenbankschnittstellen-Moduls auf die in der MySQL-Datenbank hinterlegten Regeln zu. Eine Anfrage enthält dabei die zu tranformierende typografische Auszeichnung, die zu nutzende Sprache und die Transformationsrichtung. Diese werden an das Datenbankschnittstellen-Modul per Subroutinen-Aufruf übergeben. Um das Testing zu vereinfachen, wurden für das Einpflegen der Transformationsregeln einfache HTML-basierte Applikationsoberflächen implementiert.

[74] Um den Fokus auf die Betrachtung aus logischer Sicht und weniger aus technikgetriebener Sicht zu legen, wird im Rest der Arbeit von Transformationsregeln statt von Grammatikproduktionen gesprochen.

[75] Als Standard-DBMS für den Betrieb kommt MySQL zum Einsatz.

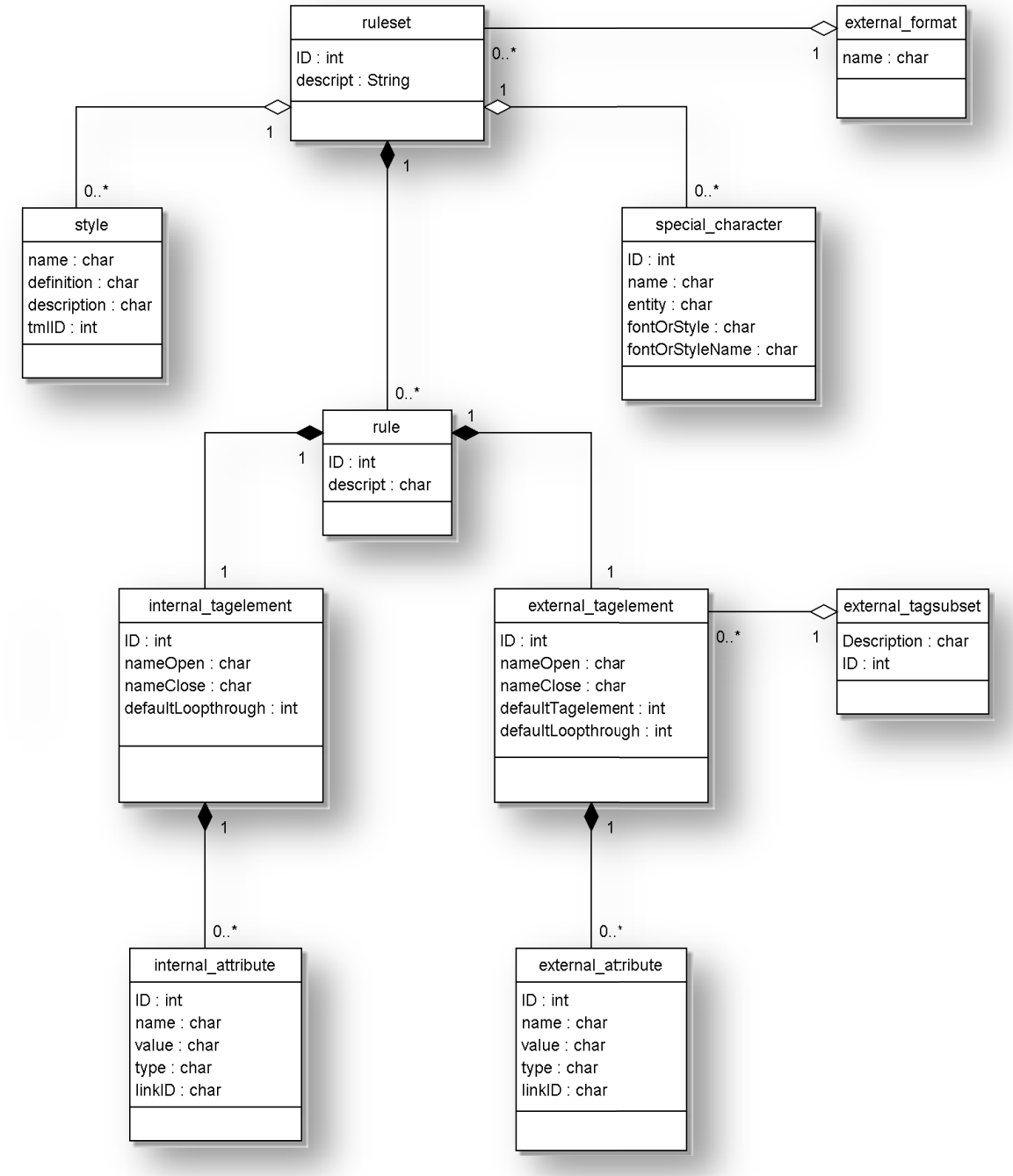

ABBILDUNG IV-4: KONZEPTMODELL DER PRIMÄREN ENTITÄTSKLASSEN DER REGELDATENBANK

In der Anfangsphase der Implementation griffen die einzelnen Parser und Builder direkt per SQL-Anfragen auf die Datenbank zu. Mit wachsender Komplexität des relationalen Modells wurden gleichzeitig die Abfragen komplizierter und verschlechterten ihr Laufzeitverhalten merklich, insbesondere, wenn es darum ging, Regeln mit mehreren Attributen richtig zu behandeln.

Für die Abfragelogik wurde später ein separater Data Centric Tier eingezogen, in welchem ein Großteil der Abfragelogik in Perl geschrieben wurde, da es nur so möglich war, die einzelnen Attribute zu verarbeiten. Eine genauere Erläuterung der Datenbankkapselung folgt auf Seite 117.

Grundlegender Aufbau einer Transformationsregel

Für jedes beteiligte externe Format werden die zugehörigen Transformationsregeln in Regelschemata zusammengefasst. Bei einem direkten Aufruf einer Transformation per Konsolenbefehl muss neben der Transformationsrichtung auch die Schemanummer mit übergeben werden. Ein Schema ist immer genau einer Sprache zugeordnet, pro Sprache können ein oder mehrere Schemata existieren.

Eine Transformationsregel entspricht einer semantischen Regel, welche die Übersetzung zwischen den beteiligten Formaten beschreibt. Sie besteht aus zwei Seiten: Einer externen Seite in einer der unterstützen externen Sprachen und einer internen Seite, der Repräsentation der typographischen Metadaten in TML. Die Regeln sind so aufgebaut, dass sie in beide Richtungen angewendet werden können. Entweder erfolgt die Übersetzung von TML in eine der externen Sprachen oder umgekehrt. Diese bidirektionale Anwendung der Regeln musste so implementiert werden, dass bei einer Übersetzung von einer externen Sprache nach TML und wieder zurück in die Ursprungssprache keine Veränderung an der Semantik des Dokumentes stattfindet.

Jede Regelseite besteht aus zwei Teilen. Der erste und obligatorische Teil spezifiziert das Tag, welches der Transformationsregel zugrunde liegt. Dies kann entweder ein einfaches Tag wie `<B>` sein, welches die typographische Information selbst enthält oder ein Tag wie beispielsweise `<div>`, welches zusätzlich mehrere typographische Informationen enthalten kann.

Diese zusätzlichen typographischen Metadaten bilden den zweiten Teil einer Regelseite in Form einer Liste von Wertepaaren aus Attributname und –wert. Um die Bidirektionalität der Transformationen zu gewährleisten, muss für jedes Attribut, welches auf der externen oder internen Seite definiert wird, ein passendes Gegenstück auf der jeweiligen Gegenseite definiert sein.

Modellierung der primären Entitäten

Die Entitätsklasse `ruleset` stellt die oberste Hierarchiestruktur im konzeptionellen Modell der Datenhaltung dar. In ihr sind die Regelsätze gespeichert, welche mehrere Regeln, die einer Sprache angehören, zusammenfasst. Jede Zeile dieser Tabelle entspricht dabei genau einem solchen Regelsatz. Es besteht eine N:1-Beziehung zur Entitätsklasse `external_format`, da jeder Regelsatz genau eine externe Sprache behandelt.

`external_format` enthält die verschiedenen, von der Transformationsengine unterstützten Sprachen. Im Zuge der Normalisierung der Entitätsklasse `ruleset` wurden Informationen in diese Tabelle ausgelagert.

Unterhalb der Entitätsklasse `ruleset` ist die Klasse `rule` angeordnet, welche sämtliche Regeln aller Regelsätze enthält. Jede Regel gehört hierbei genau einem Regelsatz an, die referentielle Integrität wird durch entsprechende Schlüsselattribute sichergestellt. Unterhalb von `rule` spaltet sich die Hierarchie in den Regelanteil der externen und internen Seite (TML) auf.

Die Entitätsklasse `external_tagelement` enthält die externe Regelseite. Jeder externen Regelseite ist hier mit einer Kardinalität von eins genau eine Regel zugeordnet. Die Tabelle beinhaltet sowohl das öffnende, wie auch das schließende Tag der jeweiligen Regel. Letzteres wurde im ursprünglichen TML-Konzept nicht dediziert mit abgespeichert. Bei der Analyse der Sprachen für Quark XPress und RTF zeigte sich jedoch, dass die zugehörigen Grammatiken nicht balanciert[76] waren, so dass die Parser auch die Definitionen der schließenden Tags benötigten. Weiterhin wird in dieser Tabelle für mehrere typographisch äquivalente Regelseiten eine Standardregel definiert.

Aus dem ersten Testprojekt entwickelte sich die Erkenntnis, dass die Transformation in externe Stile noch über die Anzahl der Spalten im Zielformat variieren kann. Damit wäre die Anzahl der zu definierenden Transformationsregeln quadratisch über die Menge der paarweise verschiedenen Spaltenanzahlen angestiegen[77].

Da geometrische Metadaten eigentlich medienabhängige Informationen sind, wurden sie im ursprünglichen Konzept der TML-Engine nicht berücksichtigt. Um auch hier das Wachstum der möglichen Regeln zu beschränken, war eine Erweiterung der Regelschemata notwendig.

Ein allgemeiner Ansatz, solche geometrieabhängigen Stilanpassungen zu unterstützen, wäre eine rekursive Erweiterung der Datenhaltung gewesen, die jeder externen Regeldefinition beliebig viele und beliebig tief strukturierte Regelvarianten erlaubt hätte. Dieser Ansatz wurde aus folgenden Gründen verworfen:

- Rekursive Strukturen in relationalen DBMS führen schnell zu problematischen SQL-Anfragen, zudem verschlechtert sich das Antwortverhalten der Datenbank.

- Die Entitätsklasse `external_tagelement` hätte um die Möglichkeit der rekursiven Datensätze erweitert werden müssen, die bereits zu diesem Zeitpunkt implementierte Verwaltung dieser Klasse hätte angepasst werden müssen.

- Es war fraglich, ob diese universelle Mächtigkeit für die Praxis überhaupt einen Mehrwert gebracht hätte, da keine realistischen Anforderungen bekannt oder zu erahnen waren.

Gelöst wurde diese Anforderung mit Hilfe einer weiteren Entitätsklasse mit dem Namen `external_tagsubset`, in welche die Anzahl der Spalten im Zielformat in einem wei-

[76] Siehe Kapitel 14

[77] Eine genauere Erläuterung findet sich in Kapitel 15

teren Attribut normalisiert wurde. Das Kardinalitätsverhältnis ist mehrfach bedingt, `(0,n)`, eine externe Regelseite kann also null, ein oder mehrere Subschemata besitzen.

Daraus folgt, dass es sich um eine reine Exportfunktionalität handelt. Für die Rückführung nach TML ist die Anzahl der Spalten im Quelldokument nicht relevant. Die konditionelle Kardinalität erlaubt die Definition globaler Regeln, beispielsweise für einen fetten Schriftschnitt, die allgemein gelten, sowie von Absatzstilen, die je nach zugeordneten Subschemata anders attributiert sein können. Bei der Transformation werden also sowohl die Regeln des Hauptschemas, als auch die Regeln des jeweiligen Subschemas benutzt.

Die Entitätsklasse `external_attribute` ordnet den attributierten, externen Regelseiten per Relation auf die Klasse `external_tagelement` die zugehörigen Attribute zu. Eine Regel kann auch ohne Attribute auskommen. Das Kardinalitätsverhältnis lautet hier `(0,n)`. Falls Attribute vorhanden sind, muss deren Anzahl auf beiden Seiten der Regel identisch sein.

Das Attribut selbst wird in drei Spalten[78] gespeichert. Die erste Spalte nimmt den Namen des Attributes auf, die zweite dessen Wert und die dritte bestimmt den Typ des Attributes.

Während der Entwicklung des TML-Prototypen tauchte das Problem auf, dass Regeln notwendig wurden, die sowohl mit Attributen als auch ohne Attributangabe auskommen mussten. Zu diesem Zweck wurde `external_attribute` eine weitere Spalte hinzugefügt, die einerseits festlegt, ob das jeweilige Attribut in der Regelanfrage vorkommen kann oder muss und die Attribute der beiden Regelseiten zueinander in Beziehung setzt, damit die Parser wissen können, welche Attributpaare zusammengehören.

Die Tabelle `internal_tagelement` repräsentiert die interne Regelseite und stellt das Pendant zur Tabelle `external_tagelement` dar, wie diese steht sie mit der Tabelle `rule` in einer N:1-Relation. Im Gegensatz zur externen Regelseite wird hier jedoch keine Standardregel definiert. `internal_attribute` gleicht der oben beschriebenen Tabelle `external_attribute` und enthält die Attribute der internen Regelseite.

[78] Um hier Sprachwirrwarr zu vermeiden, wird im Weiteren nicht der Ausdruck der ‚Entitätsattribute' aus der konzeptionellen Sicht des ER- bzw. UML-Modells verwendet, sondern von den Spalten aus dem daraus abgeleiteten logischen Modell der relationalen Tabellen gesprochen, da sie der direkten Umsetzung der Datenhaltungsschicht entsprechen und so die Verwechselung mit den Attributen der typografischen Tags vermieden wird.

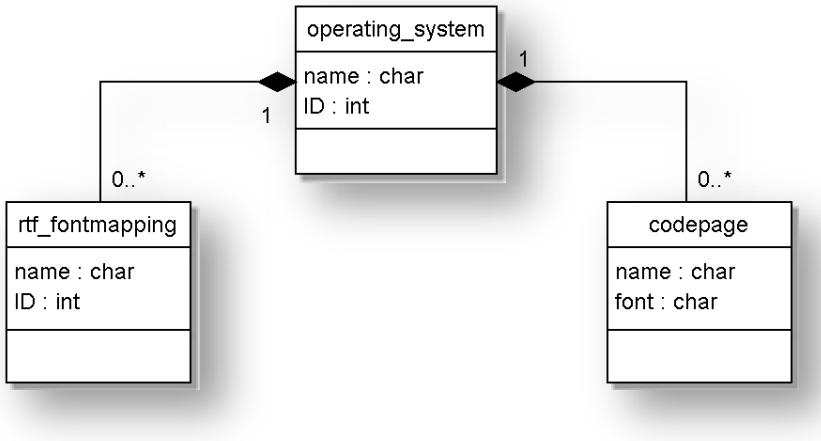

ABBILDUNG IV-5: KONZEPTMODELL DER SEKUNDÄREN ENTITÄTSKLASSEN DER TML-ENGINE

Modellierung der sekundären Entitäten

Die Transformationsengine muss nicht nur zwischen verschiedenen Sprachstandards übersetzen, sondern auch berücksichtigen, dass die zugehörigen Anwendungen auf unterschiedlichen Betriebssystemen ausgeführt werden. Das machte eine gesonderte Behandlung von Umlauten und Sonderzeichen notwendig.

Im Modell wurden dafür weitere Entitätsklassen definiert. Die Klasse `codepage` enthält die Codepage-Bezeichnungen, welche von der Codepage-Transformation benötigt werden, um die Eingabedaten entsprechend umzuwandeln. Die zu verwendende Codepage ist abhängig von der verwendeten Schriftart, die ebenfalls in dieser Klasse hinterlegt ist. Sie ist zudem betriebssystemabhängig, weshalb auch eine N:1-Beziehung zur Entitätsklasse `operating_system`[79] definiert ist.

In funktionalem Zusammenhang mit den Tabellen `codepage` und `operating_system` steht die Klasse `special_character`, ohne mit ihnen eine Relation zu bilden. Sie dient der Abbildung von Sonderzeichen zu deren Unicode-Numbered-Entity und vice versa, da Sonderzeichen in TML als HTML-Entitäten gespeichert werden. Die zu einer Entität gehörenden Sonderzeichen können von der verwendeten Schriftart oder dem verwendeten Stil abhängen, diese Metadaten werden pro Sonderzeichen zusätzlich gespeichert. Zusätzlich kann per Parametereintrag `default` eine von Font oder Stil unabhängige Sonderzeichenbehandlung realisiert werden.

[79] Aufgrund der wenigen Attribute wurde zugunsten einfacher und schnellerer SQL-Statements auf eine vollständige Normalisierung verzichtet, da zwischen operating_system und codepage eigentlich eine N:M-Beziehung mit dem Beziehungsattribut ‚font' besteht.

Ein ähnliches Problem trat bei der Font-Bearbeitung in RTF-Fragmenten auf. Innerhalb der zu übersetzenden Fragmente verwendet RTF nur künstliche Schlüssel für die Zuordnung zu Font und Codepage. Die Zuordnungen zu den natürlichen Werten sind im Header des RTF-Dokumentes definiert, welcher der Transformationsengine jedoch nicht vorliegt. Zu diesem Zweck wurde die Tabelle `rtf_fontmapping` dem Modell hinzugefügt. Sie ordnet eine vom Parser gelesene Nummer einer bestimmten Schriftart zu. Hierbei muss zusätzlich noch das jeweilige Betriebssystem berücksichtigt werden, aus diesem Grund besteht eine N:1-Beziehung zur Tabelle `operating_system`.

Absatz- und Zeichenstile

Um für den TML-Prototypen eine frühzeitige Evaluierung durch die Anforderungen der Praxis zu erreichen, erklärte sich ein Kunde der InterRed GmbH bereit, den TML-Prototypen in einem crossmedialen Produktionsszenario einzusetzen. Hier gewonnene Erkenntnisse flossen in die weitere Konzeption und Entwicklung des TML-Systems ein.

Ein markanter Punkt war der nahezu vollständige Verzicht auf individuelle typografische Formatierungen. Im Verlag existierte bereits seit längerem ein Styleguide, der die ausschließliche Verwendung von Absatz- und Zeichenstilen im Print-Produktionsprozess vorschrieb.

In der Tabelle `style` werden die Absatz- und Zeichenstildefinitionen gespeichert. In TML werden diese über einen eindeutigen numerischen Index repräsentiert und referenziert. Stildefinitionen assoziieren einen bestimmten Regelsatz, zwischen `ruleset` und `style` existiert dazu eine 1:N-Beziehung.

Diese Klasse wurde dem Modell hinzugefügt, da die Stildefinitionen bei einer Transformationsanfrage nicht mit an den jeweiligen Processor übergeben werden. Entweder liefert das externe Format die Stildefinitionen gar nicht erst mit[80] oder die Definitionen werden im Dokument-Header mitgeführt[81], der aber von der TML-Engine nicht behandelt wird.

Ausgehend von der Erkenntnis vom Umgang mit Stilen bei diesem ersten Kunden, wurde bei weiteren Verlagen recherchiert, in welchem Umfang die möglichen typographischen Eingriffe durch die Verwendung von Stilen beeinflusst wurden. Auch wenn es nicht immer gelebte Praxis war, zumindest das Ziel war die teilweise oder ausschließliche Benutzung von Absatz- und Zeichenstilen. In den meisten Fällen waren diese Stile mit sprechenden, natürlichen Namen versehen. Aufgrund der zentralen Bedeutung von Stilnamen und ihren Definitionen wurde das Konzept so erweitert, dass nur Stile transformiert werden, die in der Klasse `style` hinterlegt sind.

[80] Beispielsweise Adobe Indesign Dokumente im Tagged Text-Format

[81] Beispielsweise bei Dokumenten im RTF-Format

TML Transformation Engine

Die Übersetzung der typographischen Sprachen erfolgt innerhalb der TML Transformation Engine mit spezifischen Compilern, die je nach Richtung der Übersetzung als Prozessor oder Builder bezeichnet werden. Prozessoren übersetzen die proprietären Sprachformate der Anwendungshersteller wie XTG[82] oder existierende Standardformate wie RTF[83] in das zentrale TML-Format. Builder werden in der umgekehrten Übersetzungsrichtung aktiv, erzeugen also aus TML das benötigte externe Format. Für jedes externe Format existieren je ein Prozessor- und ein Builder-Modul, ihre Funktionsweise wird im Folgenden für die im Prototyp umgesetzten Formate näher beschrieben.

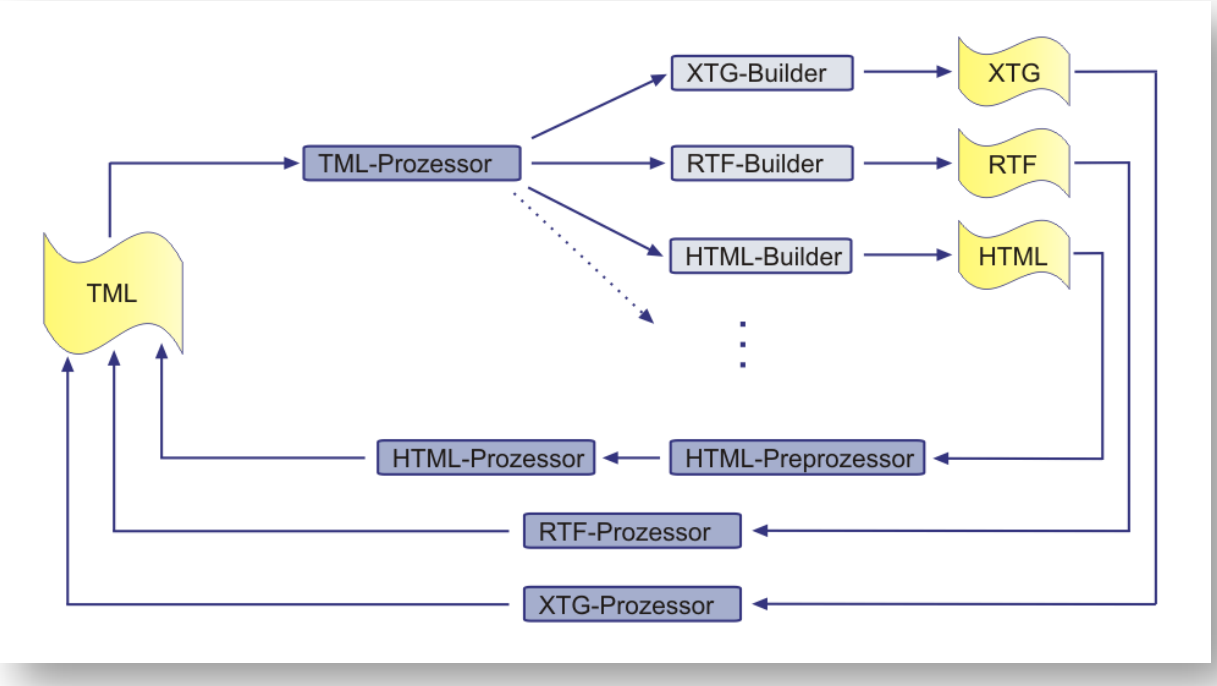

ABBILDUNG IV-6: PROZESSABLAUF IN DER TML-ENGINE

Der Aufruf der einzelnen Module erfolgt über das zentrale Perl-Modul `Transformationsengine.pm`. Im ersten Konzept der Engine sollte hier bereits eine für alle Module gemeinsam nutzbare lexikalische Analyse implementiert werden. Die zu unterstützenden Formate erwiesen sich jedoch als zu unterschiedlich. Die Eingabe für ein Prozessor-Modul ist also der vollständige String eines Absatzes, so wie er an die Transformationsengine übergeben wird[84].

[82] Quark XPress Tag

[83] Rich Text Format

[84] Entweder dediziert per Konsolenparameter oder später automatisch vom Cross Media Publishing System.

Jedes der Builder-Module wird indirekt über den TML-Prozessor von `Transformationsengine.pm` angesteuert. Die Eingabe an ein Builder-Modul ist jeweils ein einzelnes Tag bzw. ein Fragment Plain-Text.

Zur Verarbeitung der externen Formate konnte in den Fällen HTML und RTF auf fertige CPAN-Module[85] für Scanner und Parser zurückgegriffen werden. Für XTG existierte kein CPAN-Modul, stattdessen wurde ein eigenes Modul auf Basis von `Parse::RecDescent`[86] entwickelt. Bei der Verarbeitung von TML kam wiederum ein CPAN-Modul zum Parsen des XML-Textes zum Einsatz.

XTG (QuarkXPress)

Das Format XTG erwies sich nach näherer Analyse als sehr problematisch für die Programmierung eines entsprechenden Parsers. In XTG existiert nicht zu jedem öffnenden Tag ein passendes schließendes Tag. Stattdessen gibt es in XTG einige Tags, die mehrere öffnende Tags schließen können. Darüber hinaus bewirkt das Ende eines Absatzes die Aufhebung mancher Tags. XTG gehört damit zur Klasse der kontextsensitiven Sprachen, für die Transformation von XTG nach TML musste ein gesonderter Weg eingeschlagen werden.

Es existieren Tags mit und ohne Attribute. Tags ohne Attribute[87] wirken alternierend als öffnendes und schließendes Tag.

Beispiel: `<B>Dieser Text ist fett.<B> Dieser Text ist nicht mehr fett.`

Diese im Weiteren als „Toggle-Tags“ bezeichneten Auszeichnungen müssen also gezählt werden. Eine ungerade Anzahl von Vorkommen aktiviert die Auszeichnung, gerade Anzahlen deaktivieren sie[88]. Dazu werden die Tags und der jeweils aktive Zustand in einem Hash gespeichert, in welchem die Tags die Hash-Schlüssel bilden und die Information, ob dieses Tag gerade aktiv sind, die Hash-Werte. Unter den Toggle-Tags existieren einige, die im wechselseitigen Ausschluss zueinander stehen. So schaltet <+>[89] nicht nur ein vorangegangenes <+> ab, sondern auch ein vorangegangenes <->[90] und umgekehrt.

Beispiel: `<+>hochgestellt <->tiefgestellt <->nicht tiefgestellt, aber auch nicht hochgestellt.`

---

[85] CPAN steht für „Comprehensive Perl Archive Network“ und ist das zentrale Online-Archiv, in dem weltweit die Module von Perl-Entwicklern gesammelt und dokumentiert werden.

[86] Parse::RecDescent ist ein Parsergenerator auf Perl-Basis, vergleichbar dem bekannten YACC bzw. BISON.

[87] *<B>* für fett, *<I>* für kursiv, usw.

[88] Näheres siehe „Semantik der Toggle-Tags“ auf Seite 137

[89] hochgestellt

[90] tiefgestellt

Tags mit Attributen haben keinen Toggle-Charakter, man kann also nicht einfach die Information *gesetzt* oder *nicht gesetzt* speichern. Stattdessen müssen für diese Tags exakte Attributwerte gespeichert werden. Dazu dient ein weiterer Hash. Wie bei den Toggle-Tags dient der Tagname als Schlüssel, als Wert wird der Attributwert verwendet.

Im Gegensatz zu anderen bekannten XML- bzw. SGML-basierten Sprachen ist man in XTG nicht darauf beschränkt, nur ein einzelnes Tag in ein spitzes Klammerpaar zu setzen. Stattdessen ist beispielsweise die Kombination `<BIf"Arial">` möglich, mit der man Text auf einen Schlag fett, kursiv und mit der Schriftart *Arial* formatiert.

Prozessor

Während für alle anderen im Prototyp behandelten Datenformate bereits fertige Parser vorhanden waren, musste für das XTG-Format von Quark XPress ein eigener Parser implementiert werden. Aufgrund der Vielzahl kontextabhängiger Regeln wurde zur einfacheren Umsetzung der Perl-basierte Parsergenerator `Parse::RecDescent` verwendet. Die Anwendung ist vergleichbar dem bekannten Parsergenerator `YACC` für C/C++, erzeugt aber - wie der Name schon vermuten lässt - einen LL(1)-Parser. Voraussetzung dafür ist, dass die zugrunde liegende Grammatik LL(1) ist. Um ein Parserobjekt zu erzeugen, muss eine Grammatik der Sprache angegeben werden, die mit diesem Parser verarbeitet werden soll. Dies geschieht in Form einer Datei oder einer Zeichenkette. Wie in YACC können zu jeder Grammatikregel Aktionen angegeben werden, die bei einer erfolgreichen Ableitung dieser Regel ausgeführt werden sollen.

Der XTG-Prozessor unterscheidet zwischen Tags und Text. Trifft er auf ein Tag, wird zunächst nur überprüft, ob es sich um ein in einer Regel definiertes Tag handelt, bzw. um ein Tag, das verarbeitet werden soll. Tags, die nicht definiert sind oder nicht verarbeitet werden sollen, werden in ein Code-Tag gekapselt. Dieses Code-Tag bleibt bei allen bidirektionalen Transformationen der Transformationsengine erhalten und wird bei der Rücktransformation von TML zu XTG wieder entpackt. Damit wird die Erhaltung der Semantik der Ursprungsdokumente über alle Transformationen innerhalb der Engine gewährleistet.

Definierte Tags werden zunächst nur insoweit verarbeitet, als dass eine Registrierung, wie weiter oben beschrieben, in eines der beiden dafür zuständigen Hashes geschrieben wird. Die Tags im Zielformat TML werden erst generiert, wenn der XTG-Prozessor auf Text stößt. Der gefundene Text wird an die Codepage-Konvertierung (siehe Seite 123) übergeben und das Resultat wird in alle nötigen Tags verpackt. Dazu werden die Belegungen der beiden Hashes ausgelesen und in eine Liste geschrieben. Die Reihenfolge der öffnenden Tags vor dem Text entspricht einem Auslesen dieser Liste von vorne nach hinten. Analog entspricht die Reihenfolge der entsprechenden schließenden Tags nach dem Text einem Rückwärts-Auslesen dieser Liste von hinten nach vorne. Durch dieses Vorgehen ist die Wohlgeformtheit des Inhaltes eines generierten TML-Absatzes sichergestellt. Da der Inhalt einer Zeile einer XTG-Datei in einen TML-Absatz gewandelt wird, indem ein öffnendes `div`-Tag mit allen nötigen Attributen vorne und ein schließendes `div`-Tag hinten am Inhalt eines Absatzes angefügt wird, ist die Wohlgeformtheit des gesamten Dokumentes sichergestellt.

Da jede ungerade Anzahl an Vorkommen eines Toggle-Tags dafür sorgt, dass es sich auswirkt und jede gerade Anzahl dafür sorgt, dass es sich nicht auswirkt, wird nicht jedes Vorkommen eines solchen Tags einzeln übersetzt. Stattdessen werden die Tag-Informationen gesammelt und erst, wenn der XTG-Prozessor auf Text stößt, werden alle zurzeit aktiven Tags – sowohl Toggle-Tags als auch freie Tags – geschrieben.

Builder

Während beim XTG-Prozessor aufgrund der kontextsensitiven Grammatik des XTG-Formates einiger Aufwand betrieben werden musste, um XTG nach TML zu übersetzen, gestaltete sich die Übersetzung von TML nach XTG vergleichsweise einfach.

Der XTG-Builder wird vom TML-Parser für jedes Tag aufgerufen, Plain-Text wird von der Codepage-Konvertierung verarbeitet und zuvor gekapselte Tags werden ausgepackt.

RTF

Im Vergleich zu den anderen Prozessoren und Buildern gab es auch bei der Konzeption und Implementation der Übersetzer für RTF einige Sonderfälle zu beachten. Problematisch waren insbesondere die ungenaue RTF-Spezifikation und die daraus resultierenden unterschiedlichen Implementationen in den verschiedenen RTF-Editoren wie Microsoft Word, Open Office Writer, etc.

Die syntaktische Struktur der typographischen Metadaten in RTF-Dokumenten ähnelt der Syntax funktionaler Programmiersprachen, im Gegensatz zum eher imperativen Ansatz von Auszeichnungssprachen. In geklammerten Ausdrücken erfolgt die Applikation typografischer Funktionen auf den jeweiligen Inhalt (Text). Symbolisiert werden diese sogenannten Gruppen durch öffnende und schließende geschweifte Klammern. Innerhalb einer Gruppe können beliebig viele RTF-Befehle stehen, die beispielsweise die Typographie des umklammerten Textes beeinflussen. Diese Formatierungen sind nur innerhalb dieser Gruppe gültig.

Builder

Im zentralen TML-Format kann die Gruppierung der typografischen Befehle nicht per se abgebildet werden. Für jeden RTF-Befehl wird vom Builder daher eine neue Gruppe erzeugt. Dies vergrößert zwar die Datenmenge im RTF-Dokument, verändert aber nicht die Semantik im Vergleich zur gruppierten RTF-Variante. Es wäre mit der Aufnahme zusätzlicher Metadaten zwar möglich gewesen, die Gruppierungen wieder zu rekonstruieren, das hätte die Umsetzung der RTF-Übersetzer aber unnötig verkompliziert und keinen semantischen Mehrwert gehabt.

Grundsätzlich arbeitet der RTF-Builder, wie alle anderen Builder auch, eng mit dem TML-Prozessor zusammen. Im TML-Prozessor erfolgt die lexikalische und syntaktische Analyse der TML-Daten. Der Event-orientierte `XML::Parser` ruft beim Auftreten

definierter Token[91] entsprechende Unterprogramme auf und sendet die Lexeme[92] an den RTF-Builder. Diese werden dann entsprechend ihrer Struktur bearbeitet. Während dieser Bearbeitung wird in einem Stack die Information abgelegt, welcher Absatz- bzw. Zeichenstil bzw. welche Schriftart gerade aktuell ist, damit die Codepage-Konvertierung korrekt arbeiten kann.

Auch hier waren für RTF einige Besonderheiten zu beachten. Wie auf Seite 106 bereits erwähnt, werden im RTF die Schriftartinformationen, Farbinformationen usw. nicht direkt angesprochen, sondern über eine Mapping-Tabelle im Header der Datei in benötigte Tupel gruppiert und diese mit natürlichen Zahlen referenziert. Beispielsweise wird im Header der Schrift *Times New Roman* der Wert *4* zugeordnet, im Inhaltsbereich wird dann nur noch die Schrift *4*[93] aufgerufen. Hierfür musste ein geeignetes Mapping in der Datenbank integriert werden, da die Header-Informationen während der Transformierung nicht zur Verfügung stehen.

Ein weiteres Merkmal im RTF-Format ist die Tatsache, dass Befehle keine Attribute besitzen können. So muss ein `<div>`-Tag aus TML mit n Attributen in RTF in $n + 1$ einzelne Tags aufgesplittet werden. Erschwerend kam hinzu, dass alle Größenangaben in RTF in der Maßeinheit `twips` gespeichert werden, wobei $1\ twips \Leftrightarrow \frac{1}{72} Zoll$. Der Definitionsbereich von twips ist $\mathbb{N}$, der Bereich der übergebenen Werte in $\mathbb{Q}$. Es musste also ein Weg gefunden werden, der die Konvertierung nach RTF erlaubte, ohne aber die Semantik der originalen TML-Daten zu verlieren. Dies wurde durch Einsatz von in Feldern geschachtelten Informationen realisiert. Alle Größenangaben wurden sowohl gerundet in die entsprechenden RTF-Befehle konvertiert als auch ihr Original-Wert in einen Code-Tag geschrieben, den der RTF-Prozessor im Anschluss wieder auslesen kann, um die Original-Werte wiederherzustellen. Dafür werden alle Attribut-Werte zuerst in einem Array gespeichert, um dann am Ende eines Absatzes in mehrere Code-Tags aufgeteilt wieder ausgegeben zu werden.

Ebenso in Code-Tags gekapselt werden mussten alle Informationen aus TML, die in RTF nicht darstellbar sind, so etwa einige Absatz-Attribute. Auch Font-Informationen, die in TML vorhanden sind, für die aber kein Mapping in RTF vorhanden ist, müssen gekapselt werden, damit eine Wiederherstellung bei einer Rückkonvertierung wieder möglich ist.

Weiterhin mussten Inkompatibilitäten zwischen verschiedenen RTF-Versionen ausgeglichen werden. So existieren für die Laufweitendefinition eines Textes zwei RTF-Befehle: `\expnd` und `\expndtw`. Ersterer erhält als Wertangabe eine Laufweiteninformation in Viertelpunkt, letzterer erhält eine Angabe in twips. Die Angabe in Viertelpunkt ist in neueren Versionen des RTF nicht mehr definiert, aus Gründen der Kompatibilität sollten aber beide Befehle genutzt werden können, damit auch ältere Editoren und Viewer die Information korrekt erhalten.

---

[91] Start-Tag, End-Tag, Text, usw.

[92] Tag-Name und evtl. Attribute

[93] Mit: *\f4*

Prozessor

Im RTF-Prozessor mussten ebenfalls die Besonderheiten des RTF-Formats berücksichtigt werden. Für die Laufweitendefinition musste sichergestellt werden, dass nicht die durch TML erzeugte Redundanz beider Befehle interpretiert und nach TML konvertiert werden.

Der funktionale Aufbau des Formats in Gruppen von geklammerten Ausdrücken erforderte ein anderes Verfahren zum Erkennen des korrekten Endes einer Formatierung. Da kein schließendes Tag wie in XML-basierten Sprachen vorhanden ist, wurde ein Tag-Stack realisiert, in dem die Gruppen-Ebene sowie alle aktuell geöffneten RTF-Befehle gespeichert werden. Bei jedem Auftreten einer schließenden Gruppeninformation kann so über den Stack ermittelt werden, welche RTF-Befehle an dieser Stelle geschlossen werden müssen.

Die Lieferung der RTF-Token wie Gruppen, Kontrollwörter etc. übernimmt das fertige Perl-Modul `RTF-Tokenizer`, welches aus einem übergebenen RTF-String einen Tokenstrom erzeugt. Im RTF-Format sind die Kontrollwörter feingranularer als in TML. So werden dort fast alle Absatz-Informationen in Attributen des `<div>`-Tags gehalten, im RTF sind all diese Informationen einzelne Befehle, die nicht unbedingt zu Beginn des Absatzes stehen müssen. Dazu wurde ein *collecting mode* eingeführt. Dieser nimmt alle konvertierten Informationen eines Absatzes der Reihe nach auf. Erst nach Beendigung des Absatzes werden diese Informationen dann in die eigentliche Output-Variable geschrieben. So kann das Auftreten der einzelnen RTF-Befehle korrekt behandelt und die Ausgabe in korrektem TML gewährleistet werden.

Für alle Größenangaben musste zudem gewährleistet sein, dass die korrekten Attributwerte in das TML-Dokument eingetragen würden. Diese Werte wurden beim Erzeugen des RTF gerundet und der korrekte Wert in ein Code-Feld geschrieben. Diese Attribut-Informationen werden dann Absatz für Absatz ausgelesen und in einem Array vorgehalten. Bei jedem Auftreten eines RTF-Befehls mit gerundeten Angaben wurde dann die entsprechende Information in den ausgelesenen Array-Informationen gesucht und gegebenenfalls in das TML eingesetzt.

Neben Feldern, die Attribut-Informationen speichern, müssen zum Erhalt der Absatz-Semantik über alle Transformationsrichtungen hinweg zudem die Felder mit gekapselten Befehlen anderer Formate[94] ausgewertet werden. Diese müssen in die korrekte Kapselung im TML-Format übernommen werden[95] beziehungsweise korrekt in die TML-Daten eingetragen und entkapselt werden[96]. Die gekapselten Befehle können auch Backslashes enthalten. Backslashes sind im RTF-Format aber einleitende Zeichen für einen neuen Befehl und mussten deswegen in den Code-Feldern separat maskiert wer-

---

[94] TML, Quark XPress, etc.

[95] für Format-spezifische Metadaten, die nicht nach TML aufgelöst werden, für die also keine Übersetzungsanweisung in TML existieren

[96] für Metadaten, die als medienneutrale Formatierung in TML formuliert sind, aber in RTF nicht umgesetzt werden können oder sollen

den. Neben diesen Feldern werden auch noch Felder gespeichert, die weitere Schrift-Informationen enthalten, die bei der Konvertierung in das RTF-Format nicht übernommen werden können, da kein entsprechendes Font-Mapping in der Datenbank vorhanden ist. Um auch hier die Semantik von RTF-Quelldaten nicht zu verändern, muss verhindert werden, dass Felder, die keine gekapselten Informationen enthalten und durch den Benutzer im Textverarbeitungsprogramm erstellt wurden, korrekt erkannt und entfernt werden.

HTML

Im Vergleich zu den bisher vorgestellten Formaten, war die Konzeption und Implementierung des HTML-Builders und –Prozessors, bis auf die im Folgenden beschriebenen Besonderheiten, mit dem TML-Prozessor vergleichbar.

Prozessor

Beim Entwurf des HTML-Prozessors musste die grundsätzliche Frage entschieden werden, ob der Parser mit Hilfe eines Parsergenerators, wie er bereits mit Hilfe von `Parse::RecDescent` beim XTG-Prozessor erstellt wurde, selbst umgesetzt wird oder ob ein eventuell bereits vorhandener Parser verwendet werden konnte. Nach ausgiebiger Recherche wurde das Modul `HTML::Parser` aus dem CPAN näher auf Anwendbarkeit überprüft. `HTML::Parser` arbeitet folgendermaßen:

Als erstes erzeugt man ein neues Objekt vom Typ `HTML::Parser`, welchem man daraufhin Handler für definierte Events übergibt, sowie spezielle Optionen festlegt, die den eigentlichen Parservorgang beeinflussen. Events entsprechen einzelnen Produktionen aus der Grammatik von HTML, beispielsweise „Öffnendes Tag“. Anschließend übergibt man den zu parsenden HTML Text in einer Stringvariablen an die Funktion `parse()`, die nun den eigentlichen Parserlauf startet. Beim Durchlaufen des Strings ruft der `HTML::Parser` beim Auffinden von speziellen Tags die für die jeweilige Produktion definierten Funktionen auf. In diesem konkreten Fall sind dies *Starttags*, *Endtags*, *Kommentare* und *Plaintext*. `HTML::Parser` kann natürlich eine Reihe weiterer Events auslösen, allerdings werden im HTML-Prozessor nur diese vier verwendet, weshalb an dieser Stelle auf die Onlinedokumentation des `HTML::Parser` [Aas08] Moduls verwiesen wird. Die Funktionen werden mit vorher definierten Parametern aufgerufen. Im Falle von Kommentaren oder Plain-Text handelt es sich trivialerweise um den tatsächlich aufgefundenen Text. Bei den Start-Tags um den Tag, sowie einen Hash, welcher die Attribute und deren Werte enthält, bei den End-Tags nur das jeweilige Tag.

Zur Übersetzung der Tags bei den Events *Starttag* und *Endtag* genügt dann eine Datenbankanfrage mit den übergebenen Parametern. So kann der zu transformierende Text schrittweise in TML umgewandelt werden. Die Übersetzung arbeitet nach dem Prinzip des rekursiven Abstiegs eines Top-Down-Parsers.

Eine bei HTML zu beachtende Besonderheit sind Single-Tags. Sie bestehen nur aus einem Tag und besitzen kein separates schließendes Tag[97]. Solche Tags generieren im Parser jeweils einen Event vom Typ *Starttag* und einen vom Typ *Endtag*. Es musste also einer dieser beiden Events ignoriert werden. Dazu wird mittels eines Arrays überprüft, welche Tags als Single-Tags vorkommen können. In einem solchen Fall wird der Startevent und damit die dazugehörige Datenbankanfrage unterdrückt.

Mit diesem Konzept auf Basis des fertigen HTML-Parsers war die erste Implementierung des HTML-to-TML-Übersetzers bereits effektiv. Im Praxisbetrieb des ersten Prototyps zeigten sich jedoch problematische Sonderfälle, die eine gesonderte Behandlung notwendig machten. Tags und Text wurden sofort, nachdem der HTML-Parser das zugehörige Token geliefert hat, transformiert.

Beispiel: `<strong>Fetter Text</strong>`

Hier wird zuerst das Start-Tag `<strong>` in `<b>` gewandelt, dann der Text mit Hilfe der Codepage-Konvertierung umgesetzt und dann das schließende `</strong>` in `</b>`. Falls jetzt das Start-Tag unbekannt ist, weil es beispielsweise nicht definierte Attribute beinhaltet und deshalb nach Ausgabe einer entsprechenden Statusmeldung verworfen wird, der dazugehörige Endtag aber bekannt[98] ist, wird Letzteres nicht verworfen. Das Ergebnis wäre ein nicht mehr wohlgeformtes TML, welches nicht weiter verwertbar wäre, da der TML-Prozessor nur wohlgeformtes XML als Eingabe akzeptiert. Aus diesem Grund wurde ein Stack eingeführt, der sowohl die Namen und Attribute der Tags speichert, als auch deren Status, d.h. ob dieses Tag entfernt wurde oder nicht. Dieser Stack wird nun beim Aufrufen der End-Tag–Funktion überprüft, was weiterhin eine Prüfung des HTML-Inputs auf Einhaltung der Produktionen der Grammatik ermöglicht[99]. Um zu vermeiden, dass die Single-Tags diese Prüfung auf Wohlgeformtheit stören, wurde ein Array mit allen gültigen Single-Tags angelegt, die nicht auf diesen Stack[100] geschoben werden.

Im HTML-Prozessor wurde noch ein weiterer Stack benötigt. Die Implementierung der Codepage-Konvertierung erforderte den globalen Zugriff auf die im zu transformierenden Dokument vorhandenen Stile und Schriftarten. Dazu wird beim Auftreten eines gültigen Start-Tag[101] der Name dieses Stiles bzw. Fonts und der Typ des Eintrags auf einen Stack geschrieben. Liefert der Parser das Token *Plain-Text*, wird nun dieser Stack überprüft und die dort abgelegten Werte an die Codepage-Konvertierung übergeben. Analog zum Start-Tag wird beim Auffinden eines gültigen Endtags dieser Eintrag aus dem Stack entfernt.

---

[97] beispielsweise <*br*> oder <*br* />

[98] Da Endtags immer attributlos sind, kann dieser Fall häufiger auftreten.

[99] HTML::Prozessor verarbeitet nur wohlgeformten HTML Code und bricht den Parsevorgang mit einer Statusmeldung und einem Fehlercode bei Nichtwohlgeformtheit ab; da der HTML-basierte Richtext-Editor des InterRed CMS Wohlgeformtheit sicherstellt, wurde auf die Implementierung eines eigenen Mechanismus zur Prüfung auf unvollständige Tags verzichtet

[100] Die Aufgabe des Stacks ist vergleichbar mit der Symboltabelle in einem herkömmlichen Compiler.

[101] beispielsweise <*div*>, <*span*> oder <*font*>; mit entsprechenden Attributen

Builder

Für die entgegengesetzte Übersetzungsrichtung wurde ebenfalls aus dem CPAN Archiv das bereits beim RTF-Builder eingesetzte Modul `XML::Parser` [Ser08] verwendet. Ebenso wie in HTML::Parser musste für die semantische Analyse auf eigene Lösungen zurückgegriffen werden. Für die Aufgabe, die Tags als geklammerte Ausdrücke prüfen zu können, wurde ebenfalls die Ablage der öffnenden Tags auf einem Stack realisiert.

Im ersten Entwurf waren TML–Prozessor und HTML-Builder noch ein einzelnes Modul. Während der Implementation der XTG- und RTF-Builder wurde zumindest das Lesen des XML Formates in ein eigenes Modul ausgegliedert, da einige Redundanzen in den Buildern auftraten. Dabei handelt es sich insbesondere um die Initialisierung des `XML::Parser`-Objektes, sowie die Definition der Funktionen für spezielle Events.

Eine weitere Anforderung an die Implementierung des HTML-Builders gegenüber dem HTML-Prozessor und den anderen Builder-Modulen war die Unterstützung von zwei verschiedenen Regelsätzen. Neben der bidirektionalen Übersetzung nach und von HTML als eine der externen Sprachen, sollte für die reine Ausgabe von HTML-Seiten im Internet ein unidirektionales Regelwerk verwendet werden. Hierbei werden alle Tags, für die es beispielsweise keine gültige Regel, keinen gültigen Stil oder keine gültige Schriftart gibt, verworfen. Ziel war ein rein für die Ausgabe auf Hochlast-Internetseiten optimiertes, schlankes Format, das nicht wieder nach TML übersetzt werden konnte und sollte. Implementiert wurde dies durch einen separaten Regelsatz in der Datenbank und dessen separater Behandlung im HTML-Builder, in dem keine semantikerhaltenden Metadaten in die Ausgabe geschrieben werden.

Webapplikation als Pflegeoberfläche

Die Implementation der Pflegeoberfläche für die Übersetzungsregeln verwendet die CPAN-Module `CGI::Application` und `HTML::Template`.

`CGI::Application` dient der Erstellung von modularen, wieder verwendbaren Web-Applikationen. Mit `CGI::Application` konzentriert man die allgemeinen Daten und Funktionen eines Themengebiets und richtet für jede Aktion, die vom aufrufenden Client angestoßen werden kann, einen *run mode* genannten Ausführungsmodus ein. Ein *run mode* wird auf eine Subroutine in Perl abgebildet. Manche dieser Ausführungsmodi greifen auf die Datenbank zu und übergeben die ausgelesenen Daten nach Aufbereitung zur Anzeige im Browser an ein Template, manche führen nur serverseitige Aktionen aus und übergeben die Kontrolle danach an einen anderen Modus.

`HTML::Template` ist eine beliebte und einfache Template-Engine[102]. Aufgrund der relativ strikten Trennung von Design und Inhaltsverarbeitung lässt sich damit gewisser-

[102] Mit „Template“ bezeichnet man Dateien, die statt fertigem HTML-Text zur Ausgabe im Browser Platzhalter enthalten. Fordert ein Surfer eine solche Seite im Internet an, dann interpretiert die zugehörige Template-Engine diese Platzhalter und führt die entsprechende Aktion aus, beispielsweise werden modulare Definitionen eingebunden oder ein Script zur Abfrage von Datenbankinhalten ausgeführt.

maßen ein einfaches Content Management System realisieren. Insbesondere beim Aufbau von listenartigen Konstrukten wie Tabellen spart `HTML::Template` viel Implementationsarbeit. In Verbindung mit der Perl-Funktion `map()` wird die Erzeugung einer HTML-Tabelle aus einer Datenbanktabelle stark vereinfacht.

Um die Anzahl der Zugriffe auf den Webserver gering zu halten und eine einfache Transaktionsverarbeitung zu realisieren, wurden die zur Darstellung benötigten Daten parallel zum HTML-Code in eine JavaScript-Datenstruktur geschrieben. Das Verändern, Hinzufügen und Löschen von Datensätzen bewirkt zunächst nur eine Änderung innerhalb dieser Struktur. Beim Absenden des Formulars werden diese Daten gesammelt übergeben und serverseitig von Perl verarbeitet.

Konzept und Implementation des Data Centric Tier

Die Grundlage der Kommunikation mit der Datenbank erwies sich aufgrund der guten Unterstützung von Perl als relativ trivial. Perl kommuniziert mit Hilfe der beiden Schnittstellenmodule `DBD` und `DBI` mit dem MySQL-Server. Sie enthalten eine Reihe von Methoden, welche alle wichtigen Grundfunktionen von MySQL, beispielsweise SQL-Anweisungen, Verbindungsaufbau usw. unter Perl direkt zur Verfügung stellen. Das *DBI*-Modul stellt hierbei die allgemeine abstrakte Schnittstelle zur Verwendung von Datenbanken zur Verfügung, das *DBD*[103]-Modul ist der Schnittstellentreiber zwischen Programmiersprache und der MySQL-Programmier-API.

Die Anfragen der Übersetzungsanweisungen an die Datenbank, werden von den sprachspezifischen Prozessoren und Buildern durch den Aufruf von verschiedenen Subroutinen gestellt.

Beispiel:

Exemplarisch sei im Folgenden eine Anfrage mit Hilfe der Subroutine `getOpenTag()`[104] erläutert.

Der erste Übergabeparameter einer Regelanfrage enthält eine Referenz auf ein anonymes Array, dieses wiederum besteht aus drei Zahlenwerten, welche die Richtung der Transformation, das zu verwendende Regelschema, sowie einen Subregelsatz angeben.

- Als erstes wird bei jeder Transformation eine Richtung angegeben. Entweder wird eine Regel gesucht, um aus TML heraus zu transformieren oder umgekehrt, also eine Regel einer externen Sprache soll nach TML transformiert werden. Diese Unterscheidung wird mit Hilfe eines einfachen Flags getroffen, eine `0` bedeutet eine Transformation nach TML, eine `1` eine Transformation in eine unterstützte externe Sprache.

---

[103] in diesem Fall *DBD::MySQL*

[104] Der genaue Aufruf lautet beispielsweise im HTML-Builder:

@transformedTag = InterRed::TML::RuleDBI->getOpenTag(\@options, $startingTag, \@attributeList);

- Der zweite Übergabeparameter bestimmt den zu verwendenden Regelsatz. Dieser ist eindeutig einer externen Sprache zugeordnet, allerdings können für eine Sprache mehrere Regelsätze existieren. Sinn ist beispielsweise die weiter oben beschriebene Unterscheidung von Transformationen bei HTML mit Erhalt der Semantik für den Zweck der Bearbeitung des Dokumentes in einem speziellen Editor oder der Transformation nach HTML mit Veränderung der Semantik um die Inhalte in einem Browser darzustellen. Beide Übersetzungen verwenden HTML[105] als externe Sprache, ihre Zielsetzungen sind jedoch verschieden, was dazu führt, dass die beiden Regelsätze jeweils problemspezifische Transformationsregeln enthalten.

- Neben dem Regelsatz wird zusätzlich ein Subregelsatz für die eventuelle Anpassung der Typographie nach Anzahl der Fließtextspalten übergeben. Dieser wird bei der Transformation nach TML ignoriert, da er eine reine Exportfunktionalität zur Verfügung stellt. Soll kein Subregelsatz benutzt werden, wird an dieser Stelle eine `1` übergeben, ein Wert größer `1` veranlasst das Modul, auch nach Regeln zu suchen, welche dem jeweiligen Subregelsatz angehören. Falls keine Regel im Subregelsatz gefunden werden kann, wird anschließend der übergeordnete Regelsatz nach einer passenden Regel durchsucht.

- Nach diesen drei Parametern wird das vom Parser gefundene Tag übergeben. Nach der Angabe dieses Tags können schließlich beliebig viele Attributblöcke übergeben werden, wobei jeder dieser Blöcke aus drei Feldern bestehen muss:

  - Das erste Feld enthält den Namen des Attributes, das zweite Feld den vorgefundenen Wert und das dritte Feld schließlich den Typ, der allerdings nur bei der Transformation aus TML nach HTML eine Rolle spielt, in allen anderen Fällen wird hier das Schlüsselwort *undefined* übergeben. Bei einer Transformation nach HTML kann hier eines der beiden Schlüsselwörter *HTML* oder *CSS* angegeben werden. Dieses gibt an, ob der zuständige Builder das Attribut als CSS- oder HTML-Attribut aufbauen soll.

```
External Format                  TML Format
-------------------------------------------------------
<font                  →         <font
face="…"                         font-family="…" (HTML)
size="…"                         font-sizer= "…" (HTML)
name="…">                        font-name=  "…" (CSS)>
```

Eine Transformationsregel mit drei Attributen, das letzte wird in der externen Sprache mittels CSS formatiert.

- Mit Hilfe der übergebenen Parameter wird dann ein SQL-Kommando zusammengesetzt, welches die vom Parser übergebene linke Seite der Regel in der Da-

[105] bzw. HTML & CSS

tenbank sucht. Ist diese Suche erfolgreich, merkt sich die Datenbankschnittstelle die vom SQL-Kommando zurückgelieferte Regelnummer und startet mit dieser Nummer die Anfrage für die gesuchte rechte Seite.

Das dynamische Generieren dieser SQL-Kommandos erwies sich aus zwei Gründen als schwierig. Einerseits sind die SQL-Kommandos je nach Transformationsrichtung unterschiedlich, da auf der externen Seite mehrere Spalten enthalten sind, die auf der internen Seite fehlen[106]. Dies führte dazu, dass die SQL-Kommandos aus einzelnen Strings konkateniert werden mussten, die in Abhängig von der Richtung der Transformation verschieden waren. Desweiteren kam es bei dieser Konkatenierung zu Problemen mit dem sowohl in SQL als auch in Perl als Escape-Zeichen[107] verwendeten *Backslash*[108]. Dieses Problem wurde mit einer gesonderten *Backslash*-Maskierung behoben.

Die so generierte SQL-Anfrage liefert dann die gewünschte Zielseite der Transformationsregel zurück, welche noch bezüglich der benutzten Attribute und der durchzuschleifenden Attributwerte (Erläuterung siehe unten) bearbeitet wird.

Anschließend wird das Ergebnis an den Parser zurückgeliefert, wobei diese Rückgabe entweder leer[109] sein kann, aus einem einzelnen Tag besteht oder zusätzlich zum Tag noch mehrere Attribute zurückgeliefert werden. Jedes dieser Attribute wird wiederum, wie bei dem Aufruf durch den jeweiligen Parser, durch drei Skalare[110] repräsentiert, den Attributnamen, den Wert und den Typ.

Erste Übersetzungstestläufe mit realen Testdaten und die Erkenntnisse aus dem Probebetrieb in einem Verlag ergaben auch für die Konzeption des TML-Prototyps eine Reihe bis dato nicht bedachter Probleme, deren Lösungsansätze im Folgenden beschrieben werden.

Syntaktische Varianten

In einigen Sprachen gibt es verschiedene Auszeichnungen, denen jedoch dieselbe semantische Bedeutung zu Grunde liegt. So gibt es beispielsweise in HTML verschiedene Tags, die alle zur Folge haben, dass der umklammerte Text fett dargestellt wird. Welches dieser Tags in einem konkreten Dokument vorliegt, hängt unter anderem von den individuellen Vorlieben des Anwenders oder der zur Dokumenterstellung verwendeten Applikation ab.

Ist gewährleistet, dass sich die typographische Information bei mehreren solcher Auszeichnungselemente nicht unterscheidet, kann auf der externen Regelseite jedes dieser

---

106 Beispielsweise der Foreign Key auf die Subregelsätze in *tagsubset*.

107 Selbst in sonst nicht ausgewerteten Single-Quotes wird ein Backslash vom Perl-Interpreter als Escape-Zeichen benutzt.

108 Ein Backslash muss in Perl als \\ geschrieben werden.

109 Rückgabewert: *undefined*

110 In Perl werden einfache Variablen, die Zahlen oder Strings speichern können, als Skalare bezeichnet.

Elemente gleichwertig behandelt werden. Die Zielsetzung für die Transformationsengine war, alle syntaktischen Varianten der externen Sprachen mit identischer Semantik auf genau ein Element in TML abzubilden.

Bei der Transformation aus TML heraus liegt die Information, welche Auszeichnung bei der eingehenden Transformation benutzt wurde, jedoch nicht mehr vor. Aus diesem Grund wird hier eine der syntaktischen Alternativen auf der externen Seite als Standard-Regel markiert. Ausgehende Transformationen benutzten ausschließlich diese Transformation, die anderen möglichen Schreibweisen werden nur bei einer Übersetzung nach TML berücksichtigt.

Der Semantikerhalt bei bidirektionaler Regelanwendung wird hier nicht verletzt, da Standardregeln nur definiert werden dürfen, falls semantisch keine relevanten Unterschiede zwischen den verschiedenen externen Tags existieren.

External Format		TML Format
<b>	→	<b> (default)
<bold>	→	<b>
	→	<b>

Drei Transformationsregeln mit identischer Semantik. Die oberste Regel ist als Standardregel definiert.

Identität von Attribut-Tupeln

Für die Übersetzung von attributierten Tags war bereits im ersten Konzept die Möglichkeit, mehr als eine Regel pro Tag anzugeben, vorgesehen. Im Testbetrieb stellte sich jedoch heraus, dass ein einfaches Prüfen auf die Identität der Attribut-Tupel nicht genügt. Eine Teilmenge der Attribute in einem Tupel sollte potenziell nicht übersetzt, sondern unverändert weitergereicht – „durchgeschliffen" – werden. Für die Kennzeichnung solcher Attribute wurde das Schlüsselword *loopthrough* definiert.

Existieren dann für ein Tag mehrere Regeln mit unterschiedlichen Teilmengen von durchzuschleifenden Attributen, dann kann die Datenbankkapselung eventuell nicht mehr entscheiden, welche Regel anzuwenden ist.

Beispiel:

Gegeben seien zwei Regeln für das gleiche Tag mit jeweils zwei Attributen. Bei der ersten Regel ist das erste, bei der zweiten Regel das zweite Attribut auf *loopthrough* gestellt. Die beiden anderen Attributwerte enthalten jeweils einen festen Attributwert. Mit der Annahme, dass eine Regelanfrage an die Datenbank gestellt wird, die bei beiden Attribu-

ten als Wert den jeweiligen Festwert enthält, kann die Datenbankschnittstelle nicht mehr entscheiden, welche der beiden Regeln sie anwenden soll. Im Zweifelsfall repräsentieren beide Regeln zwei unterschiedliche typographische Formatanweisungen.

Somit wäre der Erhalt der Semantik in beide Übersetzungsrichtungen nicht mehr zu gewährleisten. Ein erster Lösungsansatz war eine Markierung von Regeln vergleichbar der oben beschriebenen syntaktischen Varianten. Bei einem Tupel mit mehr als zwei Attributwerten würde eine Standardregel aber nicht mehr ausreichen. Daraufhin wurde eine Prioritätshierarchie innerhalb solcher Sätze von Regeln entwickelt.

Damit gibt es zwei verschiedene Möglichkeiten, Attributwerte in der Datenbank zu hinterlegen. Die erste Möglichkeit ist, dem Attribut einen beliebigen Wert per String zuzuweisen. Bei einer Anfrage wird eine solche Regel nur bei einer Identität der Attributwerte als Ergebnis zurückgeliefert.

Die zweite Möglichkeit ist die Benutzung des Schlüsselwortes *loopthrough* gefolgt von einer Ziffernfolge mit beliebig vielen Stellen. Wird bei einer Regelanfrage dieses Schlüsselwort vorgefunden, übernimmt das Datenbankschnittstellen-Modul den vom Parser bei der Regelanfrage übergebenen Wert auf die Zielseite der Regel.

Falls das Schlüsselwort *loopthrough* bei mehreren Attributen derselben Regel benutzt wird, dient die angefügte Ziffernfolge dazu, die jeweils zusammengehörigen Attribut-Paare zu identifizieren. So wird der Attributwert `loopthrough1` auf der übergebenen Regelseite in das Attributwertfeld der Zielseite eingetragen, welches ebenfalls die Ziffer `1` enthält.

Die Regelanwendung wurde damit zwar eindeutig, im Testbetrieb stellte sich dieser Ansatz aber als rein akademisches Problem heraus, da mit der fast ausschließlichen Verwendung von Stilen mit festen Formatierungsangaben das Durchschleifen der Attribute obsolet wurde. In den wenigen Fällen, wo mit instanzbasierter Formatierung Typographie ohne Stil-Container definiert wurde, war das Durchschleifen von Attributen nicht gefordert.

External Format		TML Format

`<div`
`align="loopthrough1"`
`color="blue">`
	→	`<div`
		`align="loopthrough1"`
		`color="green">`

Eine Transformationsregel mit zwei Attributen.

Attribut 1 „align“ wird durchgereicht, Attribut 2 „color“: fester Attributwert.

Notwendige und optionale Attribute

Vergleichbar dem oben beschriebenen Problem der Identität von Attribut-Tupeln sollte es auch möglich sein, eine Transformationsregel für ein Tag zu verwenden, auch wenn die Anzahl der zugehörigen Attribute unterschiedlich ist. Beispielsweise können Absatz-

definitionen eine Vielzahl von Attributen enthalten, welche die verschiedenen Eigenschaften dieses Absatzes definieren. Ein Absatz kann aber auch ohne Attribute definiert werden und trotzdem zu einer korrekten Ausgabe führen.

Ohne eine weitere Behandlung müsste der Nutzer der TML-Engine für sämtliche Kombinationen der Attribute jeweils separate Regeln in die Datenbank eintragen, welche die verschiedenen Attributkombinationen von der Quell- in die Zielsprache übersetzen würden. Dieser hohe Konfigurationsaufwand sollte vermieden werden. Zu diesem Zweck wurde das Datenmodell so erweitert, dass es möglich wurde, Attribute entweder als *optional* oder als *required* zu markieren.

Wenn optionale Attribute in einer Regelanfrage nicht vorkommen, wird auch bei der Antwort des Datenbankschnittstellen-Moduls dieses Attribut entfernt. Zu diesem Zweck sind alle optionalen Attribute einer Seite - ähnlich den loopthrough-Attributen – über eine eindeutige Nummer mit dem jeweiligen Attribut der Gegenseite verbunden. Somit werden nur die Attribute in die Regelanfrage-Antwort übernommen, deren Nummer auch in der Anfrage benutzt wird.

Als *required* definierte Attribute sind essentieller Teil der Auszeichnungsdefinition, sie dürfen also bei einer Regelanfrage nicht fehlen. Eine Regelanfrage ist nur erfolgreich, falls sie alle benötigten Attribute enthält. Die benötigten Attribute benötigen keine fortlaufende Nummerierung, da sie nie aus der Regel ausgeblendet werden. Hier wurden bei der Implementierung also zwei Probleme mit einer Tabellenspalte gelöst, einerseits die Verknüpfung von Attributen auf der linken und rechten Seite der Regel, sowie die Möglichkeit Attribute optional anzulegen.

Beispiel 1:

Eine Transformationsregel mit einem -Tag mit mehreren Attributen, welche alle als *optional* deklariert sind. Diese Regel übersetzt ein -Tag ohne Attribute, ein -Tag, in dem sämtliche deklarierten Attribute vorkommen und alle -Tags mit einer beliebigen Kombination der als optional deklarierten Attribute. Nicht genutzte Attribute werden automatisch aus der Übersetzungsantwort gestrichen.

External Format		TML Format
<font face="…" (optional) size="…" (optional) name="…" (optional)>	→	<font font-family="…" (optional) font-sizer= "…" (optional) font-name= "…" (optional)>

Beispiel 2:

Vergleichbar mit Beispiel 1, mit dem Unterschied, dass das dritte Attribut als required markiert ist. Es werden nur Anfragen transformiert, die mindestens das dritte Attribut beinhalten.

External Format		TML Format
<font face="…" (optional) size="…" (optional) name="…" (required)>	→	<font font-family="…" (optional) font-sizer= "…" (optional) font-name= "…" (required)>

HTML-Präprozessor

Die unterschiedliche Interpretation und mehrdeutige Verwendung von HTML in den verschiedenen User-Agents und in den vom InterRed-CMS verwendeten HTML-Editor-Browserplugins erschwerte die standardisierte Verarbeitung von HTML. Semantisch äquivalente typographische Auszeichnungen wurden durch die unterschiedlichen HTML-Editoren im Content Management System teilweise durch syntaktisch unterschiedliche Tags formuliert.

Beispielsweise unterscheiden sich das Editor-Plugin für Browser, die auf der Mozilla-Engine basieren[111], von dem des Microsoft Internet Explorers in der Verwendung von Anführungszeichen für Attributwerte oder der Groß- und Kleinschreibung von Tags. Es musste daher eine Lösung gefunden werden, verschiedene externe HTML-Varianten in eine einheitliche und für den HTML-Prozessor verwendbare Form zu bringen. Das galt unabhängig von der bereits beim HTML-Builder beschriebenen Anforderung, für die unidirektionale Übersetzung von TML nach HTML eine eigene Übersetzungsvariante zu implementieren. In diesem Fall mussten beide Varianten die bidirektionale Transformation ohne Veränderung der Semantik unterstützen.

Zu diesem Zweck wurde ein HTML-Präprozessor entwickelt, welcher verschiedene HTML-Varianten in ein einheitliches Eingabeformat transformiert. So werden durch den Präprozessor unter anderem <p>-Tags in -Tags umgewandelt, Attributwerte in Anführungszeichen gesetzt, alle Tags in lowercase-Schreibweise zurückgeliefert und es findet eine Leerzeilenbehandlung statt. Durch die Verwendung des Präprozessors wurde es möglich, mit einem HTML-Regelsatz für den HTML-Prozessor zu arbeiten und trotzdem kompatibel zu den verschiedenen HTML-Editoren zu sein. Die Alternative, für die verschiedenen HTML-Editoren unterschiedliche Regelsätze anzulegen, hätte einen deutlich höheren Administrationsaufwand bedeutet, da für jeden Editor ein separater Regelsatz hätte angelegt werden müssen. Damit wäre das Problem zum späteren Nutzer verlagert worden, der dagegen sicher zu Recht protestiert hätte.

Codepage-Konvertierung

Wie bereits im Abschnitt „Modellierung der sekundären Entitäten“ auf Seite 106 kurz erläutert wurde, ist für die Übersetzung der verschiedenen externen Formate in das XML-basierte TML auch eine Konvertierung des Plain-Textes in Unicode beziehungs-

[111] Zum Beispiel der marktrelevante Firefox-Browser

weise von Unicode wieder zurück in den ursprünglichen Zeichensatz[112] der externen Formate notwendig.

Als XML-Format werden Informationen in TML im Unicode-Zeichensatz gespeichert. Damit ist sichergestellt, dass in der zentralen Datenhaltung alle Zeichen der verschiedenen existenten Zeichensätze gespeichert werden können. Im externen Format ist der gewählte Zeichensatz zum einen entweder global für das gesamte Dokument festgelegt oder mit den verwendeten Schriften definiert, da diese Schriften auf einem bestimmten Zeichensatz aufbauen.

Für eine korrekte Übersetzung der Textdaten musste zuerst einmal bekannt sein, um welchen Zeichensatz es sich handelt. Da diese Informationen nicht aus allen externen Dokumentformaten auslesbar sind, wurde in der Datenbank eine Mapping-Tabelle angelegt, in der für alle Schriftart-Betriebssystem-Tupel die entsprechende Codepage hinterlegt werden kann. Damit ist es während der gesamten Übersetzung notwendig, immer die aktuell eingesetzte Schrift zu kennen, um eine korrekte Konvertierung zu gewährleisten.

Erschwerend für die Programmierung der Codepage-Konvertierung war die schwierige Test-Situation[113]. Um ein korrektes Funktionieren zu gewährleisten, mussten Daten in verschiedenen Zeichensätzen mehrfach hin und her konvertiert werden. Die Kontrolle der Daten gestaltete sich aber schwierig, da die Darstellung im Latin1- bzw. im Unicode-Zeichensatz erfolgte, so dass es gar nicht einwandfrei möglich war, zu überprüfen, ob die Zeichen nun korrekt konvertiert wurden.

Diese Schwierigkeiten, sowie die Unterschätzung der Bedeutung und der Komplexität des Themas, führten dazu, dass die Umsetzung des Moduls als abgeschlossen galt, sich jedoch während des Testbetrieb im Verlag einige Probleme damit herausstellten, die eine erneute, intensivere Betrachtung notwendig machten. Später wurden für das Testing die Dokumente in Hex-Editoren untersucht, sowie verschiedene Perl-Module für die Übersetzungen getestet. Bis zum Abschluss der Arbeit traten im Praxisbetrieb noch vereinzelt Fehler in diesem Modul auf.

---

[112] Siehe [The05]

[113] Siehe Kapitel 13

Kapitel 13

TESTING UND OPTIMIERUNG

Bereits nach den ersten erfolgreichen Übersetzungsläufen im TML-Prototyp wurde ein Testing-Team etabliert, das die Aufgabe hatte, die jeweils aktuelle Version der Transformationsengine in der späteren realen Umgebung des Content Management Systems InterRed zu testen.

Testkonzept

Neben der allgemeinen Stabilität des Systems waren drei Ziele wesentlich für das Testing:

1. Die Transformationen müssen zur Laufzeit während der Bearbeitung in Layout und Text durch normale Anwender ausgeführt werden. Schnelle Übersetzungszeiten waren also ein K.O.-Kriterium für den praktischen Einsatz. Der Aufruf eines kurzen Zeitungsartikels sollte um nicht mehr als eine Sekunde, der Aufruf eines mehrseitigen Fachartikels um nicht mehr als fünf Sekunden verzögert werden.

2. Auch bei mehrfachen Konvertierungsdurchläufen muss die Semantik des Quelldokumentes erhalten bleiben. Da der formale Beweis des Semantikerhalts durch die Transformationen des TML-Systems mit realistischem Aufwand nicht zu erbringen ist, musste eine Sichtprüfung der Testdokumente nach mehreren Übersetzungsläufen die Wahrscheinlichkeit des Semantikerhalts stützen.

3. Sonderzeichen und Umlaute durften durch die Konvertierung zwischen verschiedenen externen Datei-Formaten und Betriebssystemen nicht verändert werden oder verloren gehen.

Vor dem ersten Praxiseinsatz wurden die Testläufe der Transformationsengine unter Laborbedingungen, unabhängig von der Integration in ein steuerndes Redaktionssystem, durchgeführt. Es wurden dafür dedizierte Referenzdokumente und Test-Skripte erstellt, welche alle bis dahin realisierten Auszeichnungen enthielten.

Für jede zum aktuellen Test-Zeitpunkt implementierte externe Sprache wurde je ein Referenzdokument erzeugt. Diese enthielten neben sämtlichen von der Transformationsengine unterstützten Auszeichnungsarten in der jeweiligen Sprache zusätzlich Umlaute und Sonderzeichen. Mit einem Ansichtsvergleich zwischen der Darstellung in den externen Anwendungen konnte von jeder Sprache aus die korrekte Transformation geprüft werden, in dem man den transformierten Text mit dem jeweiligen Referenzdokument verglich.

Testing der einzelnen Prozessoren und Builder

Die Referenzdokumente fanden ihre erste Verwendung bei den ständigen Funktionstests während der Entwicklung der Prozessoren und Builder. Bei der Arbeit an einem Prozessor außer dem TML-Prozessor wurde eine Transformation vom externen Format in TML durchgeführt. Bei einem Builder wurde entsprechend eine Transformation von TML ins externe Format durchgeführt. Die Ergebnisse wurden dann jeweils mit dem korrespondierenden Referenzdokument verglichen.

Um auch Anhaltspunkte über Probleme bei der Transformation zu haben, wenn ein Modul sich in einem nicht korrekt arbeitenden Zustand befindet, wurden an zahlreichen Stellen Debug-Ausgaben auf den Standardfehlerstrom eingebaut.

Testing der Codepage-Konvertierung

Das Testing der Codepage-Konvertierung wurde dreistufig durchgeführt. Die Funktionen wurden zuerst losgelöst von der gesamten Transformationsengine getestet. Dazu wurde ein Test-Script geschrieben, welches nach Eingabe eines Strings eine Konvertierung vornahm und diese ausgab. Als Eingabe dienten hier unter anderem Testtexte aus der CPAN-Dokumentation des Perl-Moduls `Unicode::Map8`. Diese lagen in einem norwegischen Zeichensatz vor und konnten somit relevant getestet werden. Besonderes Augenmerk lag bei den Tests darauf, dass die übliche Arbeitsumgebung im *Latin1*-Zeichensatz gehalten ist. Somit musste sichergestellt werden, dass die Ausgabe nicht nur die richtigen Zeichen enthielt, sondern diese auch korrekt[114] angezeigt wurde und nicht im Zeichensatz *Latin1*. Dafür wurden auch Tests mit Hex-Editoren durchgeführt, in denen die Ausgabe dann mit dem Zeichensatz abgeglichen wurde.

Diese Tests wurden zuerst nur für eine Konvertierungsrichtung durchgeführt[115]. Im zweiten Schritt wurden dann Konvertierungen in beide Richtungen und zwischen den Formaten ausgeführt. Als letzter Test-Schritt wurde dann die Einbettung in die gesamte Transformationsengine getestet. Hier musste überprüft werden, ob die konvertierten Zeichen auch in den Darstellungsprogrammen und Editoren der entsprechenden externen Formate korrekt dargestellt wurden.

Testing des Datenbank-Moduls

Für das Testen der Datenbank wurde ein Script geschrieben, mit dem manuell Regelanfragen gestellt werden können. So konnten die verschiedenen Subroutinen methodisch durchgetestet werden, sowohl mit gültigen Regeln, als auch mit falsch geformten oder unvollständigen Anfragen.

---

[114] Beispielsweise im norwegischen Zeichensatz

[115] Nach Unicode oder aus Unicode heraus

Um die verschiedenen Regelsonderfälle, wie Standardregeln, das Durchschleifen von Attributen und optionale oder notwendige Attribute zu testen, wurde eine separate Testdatenbank angelegt, die für jeden dieser Spezialfälle einen oder mehrere Einträge und verschiedene Kombinationen dieser Testfälle mit unterschiedlicher Anzahl von Attributen enthielt.

Schließlich wurde ausgiebiger Gebrauch von *Debug*-Ausgaben gemacht, um die einzelnen Schritte und logischen Verästelungen der Subroutinen zu überprüfen. Hier galt ein Hauptaugenmerk der Analyse der von der Schnittstelle an die Datenbank geschickten SQL-Kommandos, welche so anschließend auch außerhalb des Moduls analysiert und getestet werden konnten.

Roundtrips

Um Transformationen über mehrere Formate hinweg zu testen, wurden Rundläufe (*round trips*) in den in der Grafik angedeuteten Richtungen, jeweils ausgehend von XTG, durchlaufen. Die Ausgabe einer Transformation dient dabei als Eingabe für die nächste. Alle Zwischenergebnisse werden in Dateien abgelegt, die anschließend mit den entsprechenden Referenzdokumenten verglichen werden können.

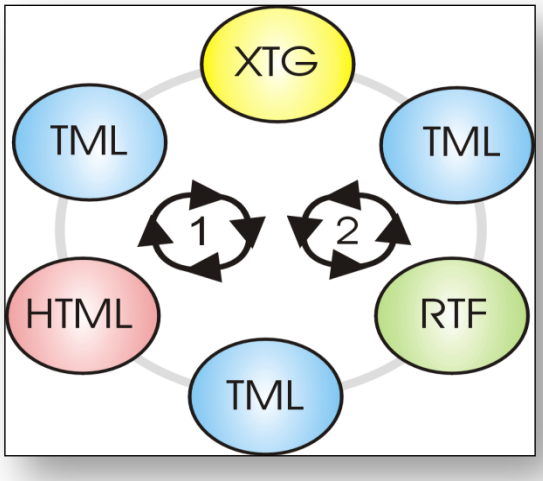

ABBILDUNG IV-7: TESTING-ROUNDTRIPS

Die aus diesen Rundläufen hervorgegangenen Dokumente wurden in einem zur Betrachtung geeigneten Programm optisch verglichen, zeichenweise in einem Hexadezimal-Editor und unter Verwendung von *Diff*-Tools[116] auf semantische Äquivalenz geprüft.

Mit den beiden Laufrichtungen 1 und 2 ließ sich bei Dokumenten mit identischem Format leicht feststellen, in welchem Modul ein Inhalt verfälscht wurde.

[116] Die Differenz-Werkzeuge dienten hier zur Unterstützung des Menschen beim manuellen Vergleich der Semantik. Sie alleine sind natürlich nicht in der Lage eine semantische Übereinstimmung zu erkennen.

Optimierung

Erste Benchmarks kompletter Übersetzungsdurchläufe zeigten Laufzeiten, die im praktischen Einsatz nicht akzeptabel gewesen wären. Nach einer ersten Grobanalyse war schnell offensichtlich, dass der deutlich größte Teil der Laufzeit auf eine hohe Latenz bei den Abfragen an die Datenbank zurück zu führen war. In Folge dessen wurde eine Reihe von Optimierungsmaßnahmen geplant, implementiert und getestet.

Verwendung von mod_perl

mod_perl bezeichnet ein Modul für den Apache-Webserver, welches einen vollständigen Perl-Interpreter innerhalb der Webserver-Umgebung zur Verfügung stellt. Dieses Modul dient einerseits dazu, die ursprünglich in C geschriebenen Apache-Schnittstellen mittels Perl zugänglich zu machen und erlaubt es andererseits, Perl-Scripte und Module innerhalb von Apache serverseitig auszuführen.

Die Verwendung von *mod_perl* bringt gegenüber der üblichen Interpretation von Perl-Modulen zur Laufzeit einen erheblichen Geschwindigkeitsvorteil, da die so genutzten Module nicht interpretiert, sondern einmalig kompiliert und persistent im Speicher gehalten werden. Da Perl aber eine Interpretersprache ist und der Perl-Interpreter bei *mod_perl* nicht mehr jedesmal neu initialisiert wird, muss bei der Implementation berücksichtigt werden, dass die modulinternen Datenbestände ebenfalls im Speicher erhalten bleiben und somit alte Werte enthalten können.

Für die TML-Engine ist dieses Verhalten jedoch nicht unerwünscht, sondern wird vom Datenbankschnittstellenmodul genutzt, um einen Cache zur Verfügung zu stellen.

Problematisch in diesem Zusammenhang waren die durch die Benutzung von *mod_perl* verschlechterten Testbedingungen. Erstens muss der Apache-Server nach jeder Änderung des Datenbankzugriffs-Moduls neu gestartet werden, zweitens werden Fehler im Modul immer mit der wenig aussagekräftigen Meldung *Internal Server Error* im Browser angezeigt. Die eigentlichen Fehlerbeschreibungen finden sich in den Log-Dateien des Apache-Servers.

Surrogatschlüssel

Um die natürlichen Daten innerhalb der Datenbank von den Strukturdaten der Datenbank zu trennen, sind in jeder Tabelle Surrogatschlüssel (künstliche Schlüssel) eingefügt worden. Sie sind die Grundlage der Relationenbildung zwischen Tabellen mit Hilfe von Primär- und Fremdschlüsseln. Überall wo nötig, wird folglich zwischen der ID, welche Teil der Datenbestände ist, und dem Primärschlüssel unterschieden, selbst wenn diese beiden Spalten inhaltlich identisch sind.

Künstliche Schlüssel haben zudem den Vorteil, dass bei Änderungen an den natürlichen Daten die Schlüsselwerte in den verknüpften Tabellen nicht angepasst werden müssen.

Damit werden Datenmanipulationsoperationen in der Laufzeit beschleunigt und im syntaktischen Aufbau vereinfacht. In den Anfragen an die Datenbank dienen die Fremd- und Primärschlüssel zur Realisierung der Restriktion in *JOINS*.

Verwendung von Indizes

Mit der geschickten Definition von Indizes kann der lesende Zugriff auf die Datenbank erheblich beschleunigt werden. Alle Tabellen wurden um einen Index auf den in den Restriktionen verwendeten Spalten erweitert. Der für diese Indizes benötigte zusätzliche Speicheraufwand war aufgrund der sehr geringen Datenmenge nicht relevant.

Ein Zuviel an Indizes kann die Performanz einer Datenbank bei Datenmanipulationsoperationen negativ beeinflussen. Aus diesem Grund ist im Normalfall eine geschickte Reihenfolge der Restriktionsdefinitionen mit einem notwendigen Minimum an Indizes anzustreben, so dass der Query-Optimizer der Datenbank die Indizes optimal verwenden kann. Im Rahmen der TML-Engine finden schreibende Zugriffe nur bei Einrichtungs- und Wartungsarbeiten und damit sehr selten statt, die negativen Auswirkungen der Indizes können also vernachlässigt werden.

Verwendung von Unique-Indizes

Unique-Indizes, also die Bedingung, dass die Werte unter einem als Unique-Index definierten Satz von Tabellenspalten eindeutig sein müssen, wurden überall dort hinzugefügt, wo diese Einschränkung durch die Logik des entworfenen Modells Sinn bringt. Damit wird die Gewährung der Eindeutigkeit der Daten schon auf einem möglichst niedrigen Level innerhalb der Datenbanklogik abgefangen, was einerseits die Umsetzung der Pflegekomponente vereinfachte, andererseits auch bei einem Schreiben in die Datenbank mit anderen Tools oder direkt auf der Konsole vor fehlerhafter Befüllung schützt.

Optimierung der SQL-Anfragen

Im Laufe der Entwicklung der TML-Engine wurde permanent das Modell der Datenbank erweitert und modifiziert. Die Änderungen am Entity-Relationship-Modell der Datenbank waren in den meisten Fällen ohne größeren Aufwand durchzuführen. Im Data Centric Tier wurden die Abfrage-Logik und die Parameterverarbeitung jeweils angepasst und ergänzt. Das führte zu einem immer komplexeren und unübersichtlicheren Code.

Aus diesem Grund wurde die Kernfunktionalität des Datenbankschnittstellenmoduls noch einmal neu analysiert und in der Folge vollständig neu implementiert. Die bestehenden SQL-Kommandos wurden so neu strukturiert, dass möglichst wenige Anfragen abgesetzt werden. Dazu wurden die Verbünde geprüft und zum großen Teil neu definiert. Da das Problem der optimalen Verbundreihenfolge NP-vollständig ist, wurde auf

eine deduktive Herleitung verzichtet. Naheliegende Verbünde wurden implementiert und die Laufzeit getestet.

Diese komplette Überarbeitung des Data Centric Tier brachte den größten Laufzeitgewinn aller durchgeführten Optimierungsmaßnahmen.

Caching von SQL-Abfragen

Durch die Verwendung von *mod_perl* wurde das Caching der Resultsets der SQL-Anfragen innerhalb des Perl-Moduls möglich, da die in einem Modul hinterlegten Datenbestände persistent im Speicher gehalten werden. Zu diesem Zweck werden bereits durchgeführte SQL-Anfragen als Schlüssel für ein Hash verwendet, das Ergebnis der Anfrage wird diesem Schlüssel anschließend als Wert zugeordnet.

Im Data Centric Tier wird nun vor dem Ausführen einer SQL-Anfrage überprüft, ob diese Anfrage bereits als Schlüssel im SQL-Cache-Hash enthalten ist. Ist dies der Fall, wird die Anfrage nicht an die Datenbank übergeben, sondern der Wert des Hash-Eintrages an der jeweiligen Position zurückgegeben.

Die Laufzeitkomplexität der SQL-Anfragen verringerte sich bei Verwendung des SQL-Cache durchschnittlich um den Faktor zehn[117]. Ein neues Problem, welches sich durch das Caching ergab, war der Speicherverbrauch durch den Cache. Er darf nicht beliebig groß werden, da beim Ausschöpfen des Arbeitsspeichers die Geschwindigkeit des Apache-Serverprozesses negativ beeinflusst werden kann, wenn dieser beginnt, den Auslagerungsspeicher auf der Festplatte zu verwenden.

Um diese Speicherauslagerung zu verhindern, wird der Cache regelmäßig teilentleert. Hierzu werden zwei verschiedene Kriterien geprüft, die bestimmen, ob bei einer Teilentleerung der Eintrag im Cache erhalten bleiben soll oder entfernt werden kann.

Das erste Kriterium ist die Aktualität des Eintrags, welche mit Hilfe eines Zeitstempels geprüft wird. Dieses Kriterium folgt der Annahme, dass aktuellere Cache-Einträge eher benötigt werden, als ältere.

Das zweite Kriterium benutzt die Anzahl der Treffer, prüft also, wie oft ein Cache-Eintrag bereits verwendet wurde. Die Annahme ist hier, dass eine bereits häufig benutzte Anfrage auch in Zukunft häufiger benutzt wird. Mit Hilfe einer parametrisierbaren Gewichtung zwischen den Prüfkriterien, einer maximalen Cache-Größe und der Angabe der Größe des Anteils des Caches, der gerettet werden soll, kann das SQL-Caching auf den jeweiligen Projekteinsatz optimiert werden.

Im ersten Konzeptentwurf für die Transformations-Engine war ein SQL-Cache mit Hilfe von Perl-Objekten geplant. Aufgrund von Prioritätsverschiebungen während der Implementierungsphase und der Annahme, dass das in der MySQL-Datenbank integrierte

[117] Arithmetischer Mittelwert aus gemessenen Stichproben

Caching eine eigene Lösung überflüssig machen könnte, wurde die Fertigstellung nicht weiter verfolgt. Als die ersten Benchmarks Probleme bei der Laufzeit der SQL-Anfragen aufzeigten, wurde der Objekt-Ansatz verworfen und durch das effizientere Verfahren auf der Basis von Perl-Hashes umgestellt. Durch den Betrieb unter *mod_perl* bleiben die Datenstrukturen persistent im Speicher und können mit optimaler Performance abgefragt werden.

Benchmarking

Um die Leistungsfähigkeit und das Laufzeitverhalten der Transformationsengine einschätzen zu können, wurden verschiedene Benchmarks durchgeführt.

Die Laufzeit einer Transformation wurde über die Länge der Eingabesätze skaliert und gemessen. Basis war das XTG-Referenzdokument. In diesem wurde einmal über die Länge der reinen Textinhalte und einmal über die Anzahl der Tags skaliert, dazu wurden die Inhalte n-fach hintereinander gehängt. Dabei galt:

$$Faktor: n = 2^m, 0 \leq m \leq 8, m \in \mathbb{N}$$

Zur Zeitmessung wurde das Perl-Modul `Time::HiRes` benutzt, welches Messungen im Millisekundenbereich ermöglicht.

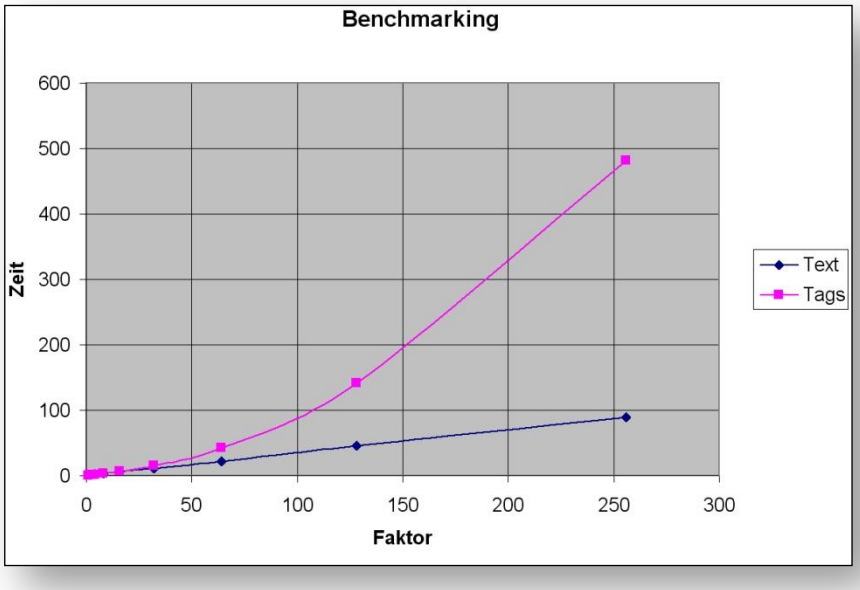

ABBILDUNG IV-8: LAUFZEITEN DER TRANSFORMATIONSENGINE. ZEIT IN MS, FAKTOR = N

Die Ergebnisse in Abbildung IV-8 basieren auf einem Testsystem mit einem Athlon-64-3000-Prozessor und 1 GB DDR-400-Arbeitsspeicher auf Debian 3.1.

Bei den Benchmarktests der Datenbank wurden die einzelnen, weiter oben beschriebenen Optimierungsmaßnahmen separat getestet. Einzeln gemessen wurden die Laufzeitveränderungen durch den Einbau des SQL-Caches, die Verwendung von Indizes sowie die Optimierung der SQL-Anfragen und des Codes im Data Centric Tier.

Die Effizienz des Datenbank-Cachings ließ sich in künstlicher Umgebung mit künstlichen Testdaten nur näherungsweise bestimmen. Eine Anfrage über den SQL-Cache ist gegenüber einer nativen Datenbankanfrage über `DBI/DBD` in etwa um den Faktor zehn schneller. Auf die Laufzeit des Gesamtsystems wirkte sich diese Verbesserung jedoch relativ wenig aus. Um die Laufzeit der Übersetzer und die Optimierungsmaßnahmen in der Datenbank-Kapselung besser evaluieren zu können, blieb das Caching bei den Benchmark-Tests ausgeschaltet.

Kapitel 14

THEORETISCHE ÜBERLEGUNGEN ZUM BAU DER PARSER

Dokumentteile, die nach TML übersetzt und in diesem Format gespeichert werden, sind wie alle XML-basierten Dokumente strukturiert. Das Konzept der Strukturierung von Texten ist keine Errungenschaft von XML. Texte enthalten schon immer interne Strukturen, um sie verständlicher und übersichtlicher zu machen. Dabei lässt sich die Art der Strukturen in zwei Klassen unterteilen: inhaltliche und formale Strukturen.

Inhaltliche Strukturen in einem Dokument können beispielsweise zeitlicher Art sein, wenn sie zusammenhängende Ereignisse in einem Bericht oder einer Erzählung in eine zeitliche Abfolge bringen. Eine Bedienungsanleitung mit aufeinanderfolgenden Schritten hat ebenfalls eine zeitliche Struktur.

Semantische Klassifizierungen können einen Text ebenfalls inhaltlich strukturieren. Ein Lexikon kann nach kulturellen, politischen, naturwissenschaftlichen, etc. Inhalten strukturiert sein, ein Gedichtband nach trivialen, lyrischen und Liebesgedichten.

Um inhaltliche Strukturen zu erkennen, muss der Text vom Leser verstanden werden. Im Gegensatz dazu stehen formale Strukturen. Hier ist ein Verständnis des Textes nicht notwendig. Klassische formale Strukturen sind die Unterteilung eines Textes in Kapitel, Teile, Abschnitte, etc.

Neben dieser Gliederungsstruktur kann eine formale Strukturierung auch durch die Formatierung mit Hilfe typographische Auszeichnungen gegeben sein. Dieses Beispiel zeigt auch, dass inhaltliche und formale Strukturen nicht immer klar abgegrenzt werden können. Formatiert man ein Dokument mit typographischen Auszeichnungen, beispielsweise durch Fettung für wichtige Begriffe, Einrückung zusammengehöriger Aufzählungen, größerer Schrift für Überschriften, etc., so erhält das Dokument dadurch auch eine inhaltliche Struktur.

Datenorientierte Auszeichungssprachen

Markup-Sprachen, die dem Austausch von Daten dienen, können neben inhaltlichen und formalen Strukturen in datenorientierte und in textorientierte Auszeichnungssprachen unterschieden werden. Ein Beispiel für eine datenorientierte Markupsprache ist die Graph Exchange Language (GXL), sie beschreibt getypte, attributierte Graphstrukturen und ihre Schemata und dient dem Austausch zwischen Graphtransformationssystemen und Graphenspeichern.

Ein weiterer spezialisierter Dialekt mit dem Namen Chemical Markup Language (CML) beschreibt Moleküle durch die Anordnung der Atome im Raum sowie ihrer Bindungen. Web-based Distributed Authoring and Versioning (WebDAV) standardisiert den Zugriff

auf Dokumentenserver und erlaubt das verteilte Bearbeiten von bspw. Office-Dokumenten über das Hypertext Transfer Protokoll (HTTP).

Datenorientierte Auszeichnungssprachen sind ohne das vorhandene Markup nicht verständlich oder nicht interpretierbar. CML-Dokumente ohne Markup sind Kolonnen von Zahlen und Texten. Erst durch das Markup werden Tokengrenzen definiert und die Dokumente werden interpretierbar. Bestimmte Informationen finden sich ausschließlich in den Attributwerten. So ist die Zuordnung der Bindungen zu den gebundenen Molekülen auch in einfachen CML-Dokumenten ohne Markup unmöglich. (nach [Mey06])

Textorientierte Auszeichungssprachen

Die textorientierten Markup-Sprachen ergänzen ein Dokument, was im Allgemeinen durch einen Menschen lesbar ist, mit Strukturdaten. Wird das Markup wieder entfernt, so bleibt das Dokument les- und nutzbar. Bekannte Beispielformate für textorientierte Auszeichnungssprachen sind NewsML und NITF (News Industry Text Format). Sie enthalten aktuelle Meldungen und Nachrichten als normalen Text, angereichert mit wichtigen Metadaten. Durch die Trennung von Form und Inhalt können Texte zwischen verschiedenen Systemen übertragen und mediengerecht und kundenspezifisch formatiert ausgegeben werden, beispielsweise als HTML-Seite eines Nachrichtenmagazins oder als Videotextseite eines TV-Kanals.

Die Grenze zu den datenorientierten Sprachen ist nicht strikt, in jeder Auszeichnungssprache kommen beide Formen gemischt vor. Auszeichnungen in datenorientierten Formaten sind häufig mit verständlichen Begriffen bezeichnet, umgekehrt enthalten textorientierte Formate vielfach technische Zusatzinformationen für die automatisierte Verarbeitung, die für Menschen im Normalfall unverständlich sind.

HTML selbst dürfte vielleicht das bekannteste Beispiel für hauptsächlich textorientiertes Markup sein. In der einfachsten Form besteht HTML aus einem Header mit ein paar grundsätzlichen Formatangaben und normalem, lesbarem Text, der durch einfache Strukturinformationen zur Bildung von Absätzen, Einzügen, etc. durchsetzt ist. Einfache typografische Tags genügen für eine grundlegende Formatierung des Textes. Das lesbare, verständliche Format der Auszeichnungen hat zur großen Akzeptanz von HTML beigetragen.

Die typographische Struktur von Dokumenten wird im Rahmen dieser Arbeit medienneutral mit TML beschrieben und als zentrales Austauschformat gespeichert. TML wurde in Anlehnung an HTML als textorientierte Markup-Sprache entwickelt. Dieser Ansatz erleichtert die Entwicklung und die Fehlersuche durch das leichte Verständnis der Semantik von TML-Texten. Des Weiteren kann dieses Format potenziell an dritte Verwertungsstellen weitergeleitet werden und mit standardisierten XML-Werkzeugen weiterverarbeitet werden. Im Folgenden werden die formalen Eigenschaften und besondere Aspekte der Semantik von TML näher erläutert.

Klammersprachen

Typographische Auszeichnungen werden in TML durch Tags definiert. Der Name des Tags und eventuell zugehörige Attribute beschreiben die typographische Wirkung – die Semantik – des Tags. Die Sprachinhalte, auf die die typographischen Formatierungsanweisungen ausgeführt werden sollen, befinden sich zwischen dem Anfangs- und dem Endtag einer Anweisung. Anfangs- und Endtag bilden Paare, die geschachtelt werden können. Dazu ein Beispiel:

```
<div>
    <strong>
        <font_a>Lore Ipsum</font_a>
        <italic>Dolor Sit</italic>
        <sup>Amet</sup>
    </strong>
    <italic>Consectetur</italic>
    ...
</div>
```

Beispielprogramm p_1

Für die Entwicklung der TML-Parser war es nun wichtig zu wissen, zu welcher Sprachklasse eine solche Tag-Sprache gehört. Ist sie nach der Chomsky-Hierarchie eine Typ 2-Sprache und gehört damit zur Klasse der kontextfreien Sprachen? Unter der Annahme weiterer Nebenbedingungen[118] hätte dann manuell oder mit Hilfe eines Parser-Generators[119] ein geeigneter Übersetzer als Kellerautomat implementiert werden können.

Definition Grammatik

Eine Grammatik ist ein 4-Tupel $G = (N, T, P, s)$. Dabei sind N und T (endliche, nichtleere) Alphabete. Sei $V = N \cup T$, dann ist $P \subseteq V^* \times V^*$ eine endliche Relation und $s \in N$.

- N heißt Alphabet der Nichtterminalzeichen.
- T heißt Alphabet der Terminalzeichen.
- P heißt Menge der Produktionen (Regeln).
- s heißt Startsymbol.

[118] Die Einschränkung auf Typ 2-Sprachen reicht für die Praxis nicht aus. Ein nichtdeterministischer Kellerautomat (NPDA) erkennt zwar Typ 2-Sprachen, ist aber zu ineffizient. Um einen effizienten Parser konstruieren zu können, muss die Grammatik weitere Einschränkungen erfüllen, bspw. LR(1), LALR(1) oder LL(1).

[119] Ein von einem Parsergenerator wie YACC erzeugter Parser ist im Wesentlichen ein Kellerautomat mit Ausgabeband.

Sei $G = (N, T, P, s)$ eine Grammatik, dann definiert man zwei Sprachen, die von G erzeugt werden:

- $S(G) = \{w \in V^* | s \underset{P}{\rightarrow}^* w\}$ Menge der Satzformen,
- $L(G) = S(G) \cap T^*$ Menge der terminalen Satzformen.

Definition kontextfreie Grammatik

Eine Grammatik $G = (N, T, P, s)$ heißt Typ 2-Grammatik oder kontextfrei (c.f.)

$\Leftrightarrow P \subseteq N \times V^*$ (nach [Mer02])

Zur Herleitung einer Grammatik G, welche eine Tag-Sprache $L(G)$ erzeugt, so dass Programme der Art $p_1 \in L$ erzeugt werden können, betrachten wir zunächst die primitive Klammersprache $L_2 = \{<^n, >^n,\ n \in \mathbb{N}\}$, welche nur Schachtelungen eines Klammersymbols erlaubt:

Beispiel:

```
<
    <
        <>
    >
>
```

Beispielprogramm p_2

Grammatik G_2 für L_2:

```
T={<, >}
N={Element}
Startsymbol=Element
P:
Element   →    < > | < Element > | ε
```

G_2 ist kontextfrei. Allgemein gilt:

$L = \{a^n b^n | n \geq 1\}$ ist kontextfrei, mit der Grammatik $S \rightarrow aSb|ab$. [Aho88]

Definition Dyck-Sprachen

Die Menge aller wohlgeformten[120] Klammerausdrücke mit n verschiedenen Klammersymbolen: $(_n,)_n$ für $n \in \mathbb{N}$ heißt Dyck-Sprache D_n [Die03], erzeugt durch die Grammatik:

```
T={(i, )i}
N={S}
Startsymbol=S
P:
S    →   SS
S    →   (i S )i
```

Um nun Klammerstrukturen erzeugen zu können, wie sie in XML-Texten oder in den bekannten Programmiersprachen vorkommen können, erweitern wir die primitive Klammersprache um die Konkatenation und erhalten die Dyck-Sprache D_1, die nur ein Klammersymbol enthält:

```
<
    <
        <>
        <>
        <>
    >
    <>
    ...
>
```

Beispielprogramm $p_3 \in D_1$

Grammatik G_3 für D_1:

```
T={<, >}
N={Element, List}
Startsymbol=Element
P:
Element   →   < > | < List > | ε
List      →   Element | Element List
```

Auch G_3 ist (deterministisch) kontextfrei. Als nächstes erweitern wir die Sprache um die Textinhalte zwischen den Klammern:

[120] Definition der Wohlgeformtheit von XML-Dokumenten siehe http://edition-w3c.de/TR/2000/REC-xml-20001006/#sec-well-formed

```
<
    <
        <Lore Ipsum>
        <Dolor Sit>
        <Amet>
    >
    <Consectetur>
    ...
>
```

Beispielprogramm p_4

Eine Grammatik G_4, die Programme der Art p_4 erzeugen kann, lautet:

```
T={<, >, a, b, ..., z, A, B, ..., Z}
N={Char, Ident, Element, List}
Startsymbol=Element
P:
Char        →    a|b|c|...|z|A|B|...|Z
Ident       →    Char|Char Ident
Element     →    < Ident >|< List >|ε
List        →    Element|Element List
```

Auch G_4 ist kontextfrei.

Erweitern wir jetzt $L_4(G_4)$ um Tags und geben somit jedem Klammerpaar einen Namen, so erhalten wir Programme wie im Beispiel p_1. Eine Grammatik G_1, die Programme wie p_1 erzeugt, lässt sich wie folgt formulieren:

```
T={<, >, /, _, a, b, ..., z, A, B, ..., Z}
N={Char, Ident, OTag, ETag, Element, List}
Startsymbol=Element
P:
Char        →    a|b|c|...|z|A|B|...|Z
Ident       →    Char|Char Ident
OTag        →    < Ident >
ETag        →    < / Ident >
Element     →    OTag Ident ETag|ε            [*]
Element     →    OTag List ETag               [*]
List        →    Element|Element List
```

G_1 kann Programme wie p_1 erzeugen, aber auch nicht wohlgeformte XML-Texte. Verhindern lässt sich das mit extern formulierten, semantischen Nebenbedingungen [*], dass Anfangs- und Endtag in diesen Produktionen zusammenpassen müssen. Das ist problematisch, da sich dafür ohne weiteres keine c.f.-Grammatik angeben lässt.

Es kann folgendes, einfacheres Beispiel angegeben werden:

$L_5 = \{wcw | w \in T_0^*\}, T = \{a, b, c\}, T_0 = \{a, b\}$ ist die Menge aller Wörter, die aus zwei identischen Sequenzen von a´s und b´s bestehen, die durch c getrennt sind, beispielsweise $aabcaab$. L_5 gehört nicht zur Klasse der kontextfreien Sprachen. [Aho88].

Für L_5 lässt sich folgende Grammatik G_5 angeben:

```
T={a, b, c}
N={O, E, I, S}
Startsymbol=S
P:
S    →    O I E          [*]
O    →    aO | bO
E    →    aE | bE
I    →    c
```

[*] Hier muss gefordert werden, das O und E identisch sind.

Würde das schließende Tag spiegelbildlich geschrieben, dann wäre die Sprache interessanterweise kontextfrei: $L_5' = \{wcw^R | w \text{ ist in } (a|b)^*\}$, w^R steht für das Spiegelbild von w, ist kontextfrei und kann durch folgende Grammatik erzeugt werden:[Aho88]

$$S \rightarrow aSa|bSb|c$$

Mit dem Ansatz, bis zum Erreichen von I, w^R zunächst als Spiegelbild von w zu erzeugen und Hilfssymbole (X) zwischen die Terminalzeichen in w^R einzufügen und anschließend durch vertauschen der Terminalzeichen aus w^R wieder w zu erzeugen, lässt sich folgende alternative Grammatik G_5' formulieren (nach [Bec04]), die ohne semantische Nebenbedingung auskommt und die L_5 erzeugt:

```
T={a, b, c}
N={I, X, S}
Startsymbol=S
P:
S    → aSaX | bSbX | I      (1, 2, 3)
aXb  → baX                  (4)
aXa  → aaX                  (5)
bXa  → abX                  (6)
bXb  → bbX                  (7)
aX   → a                    (8)
bX   → b                    (9)
I    → c                    (10)
```

Gesucht: Ableitung für $aabcaab$ in G_5'

$$\underline{S} \underset{1}{\rightarrow} a\underline{S}aX \underset{1}{\rightarrow} aa\underline{S}aXaX \underset{2}{\rightarrow} aab\underline{S}bXaXaX \underset{3}{\rightarrow} aab\underline{I}bXaXaX \underset{10}{\rightarrow} aabc\underline{bX}aXaX$$

$$\underset{6}{\rightarrow} aabca\underline{bX}XaX \underset{9}{\rightarrow} aabca\underline{bXa}X \underset{6}{\rightarrow} aabcaa\underline{bX}X \underset{9}{\rightarrow} aabcaa\underline{bX} \underset{9}{\rightarrow} aabcaab$$

Bei jedem Ableitungsschritt ist hier die linke Seite der jeweils verwendeten Regel unterstrichen und die verwendete Regel angegeben.

Der Ableitungsgraph[121] ergibt sich wie folgt:

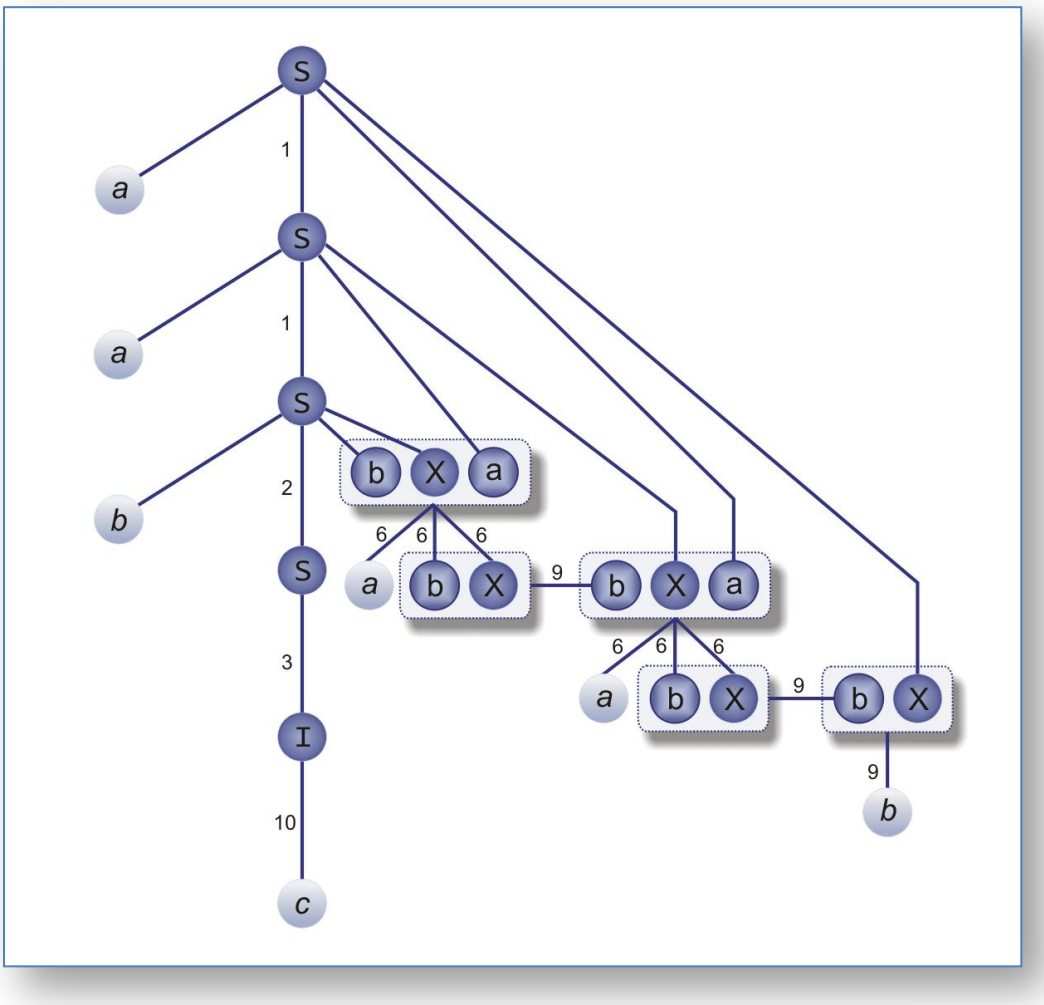

ABBILDUNG IV-9: ABLEITUNGSGRAPH FÜR $aabcaab$ IN $G_5{}'$

Die Folge der Einzelschritte ist also ein Beweis für die Behauptung:

$$aabcaab \in L(G_5')$$

Mit der Hintereinanderausführung der Regeln 1, 10 und 8 gilt beispielsweise auch:

[121] Es handelt sich um keinen (Ableitungs-)Baum, da Sohnknoten potenziell von mehr als einem Vater abstammen können.

$$aca \in L(G_5')$$

Es gilt allgemeiner: Die Syntax der Tag-Klammersprache mit der Menge aller gültigen Sätze $M = \{wcw | w \text{ ist in } (a|b)^*\}$ ist identisch mit der Menge aller Sätze, die sich mit der Grammatik G_5' erzeugen lassen, also:

$$M = L(G_5')$$

Tag-Sprachen mit beschränktem Tag-Alphabet lassen sich also durch eine Grammatik wie G_5' erzeugen.

G_5' ist zwar eine Typ 0-Grammatik, man kann aber zeigen, dass die Sprache L_5 auch mit einem linear beschränkten Automat (LBA) erkannt werden kann. Dazu lässt man den Schreib-/Lesekopf des LBA bis zum Erreichen des c über das Eingabeband laufen. Nach dem Lesen des nächsten Zeichens fährt der Kopf an den Anfang der Eingabe zurück und löscht dort das erste Zeichen, falls es mit dem nach c gelesen Zeichen übereinstimmt. Danach liest er das nächste Zeichen nach dem c und löscht wieder das Pendant am Anfang des Bandes usw. Wenn nach dem Lesen des letzten Zeichens vor dem Bandende und dem Löschen des Pendants nur noch das c auf dem Band steht, dann war das Eingabewort aus L_5.

Dieser LBA ist ein Akzeptor für Typ 1-Sprachen, die Tagsprache L_5 gehört also zur Klasse der kontextsensitiven Sprachen. Wie die Beispiele der Grammatiken G_5 und G_5' verdeutlichen, lässt sich für eine Tag-Sprache, mit einem begrenzten Alphabet für die Konstruktion der Tags, eine Grammatik finden, die ohne Nebenbedingungen auskommt, die außerhalb der Grammatik erfüllt werden müssen. Eine solche Grammatik zu formulieren dürfte aber in den meisten Fällen schwierig sein, zudem wird der zugehörige Automat (LBA) sehr ineffizient und für praktische Anwendungen kaum brauchbar. Ein LBA (mit Ausgabeband) für G_5' muss bei Eingabeworten, die eine bestimmte Länge überschreiten, bereits mit Backtracking (der Lesekopf muss auch rückwärts laufen) arbeiten, um eine korrekte Ableitung zu erzeugen.

Für die Programmierung eines universellen Übersetzers für Tag-Sprachen auf Basis eines Kellerautomaten bietet sich folgender Ausweg an:

XML als kontextfreie Sprache

Fügt man die Menge aller möglichen Tags der Menge der Terminalzeichen hinzu und schreibt alle auftauchenden Tags in die Symboltabelle und definiert für jedes Tag eine kontextfreie Produktion, so erhält man doch eine kontextfreie Grammatik, auf deren Basis mit den bekannten Methoden alle gültigen Satzformen abgeleitet werden können. Die Menge der Tags muss dazu endlich sein[122]. XML-artige Sprachen lassen sich dann

[122] Theoretisch erlaubt XML unendlich viele Tags. Fordert man, dass alle Tags terminale Symbole sind (siehe [Wen08]), hätte XML also keine endliche Grammatik. Zwar ist jede Teilmenge von A* eine formale Sprache, es existieren aber

mit einem Kellerautomaten erkennen und gehören zur Klasse der kontextfreien Sprachen[123]. Ein entsprechender Beweis findet sich in [Wen08], Satz 4.2.8.

Die Grammatik von XML wird von einigen Bedingungen beeinflusst, die formlos definiert sind, beispielsweise die Bedingungen für die Wohlgeformtheit WFC (Well-formedness Constraints). Die WFC sind syntaktische Regeln für die Sprache. Dazu kommen die Gültigkeitsbedingungen VC (Validity Constraints), die überwiegend semantische Eigenschaften beschreiben, die ein XML-Dokument zusätzlich erfüllen muss.

Beide Bedingungen sind in natürlicher Sprache und nicht als Produktionen spezifiziert, ihre formale Prüfung fällt also schwer. Insgesamt ist der Standard dadurch besser lesbar und einfacher zu verstehen. Die vielen im Markt produktiven und zuverlässigen XML-Systeme und –Prozessoren, die entscheiden können, ob ein Wort ein wohlgeformter XML-Ausdruck ist, beweisen gewissermaßen experimentell, dass XML eine formale Sprache mit endlicher Grammatik ist.[Rot03]

Die Sprache L_5 abstrahiert auch einen Standardfall für die semantische Analyse in Compilern bekannter Programmiersprachen: Bezeichner müssen vor ihrer ersten Benutzung deklariert werden. Das erste w in wcw kann als Deklaration des Bezeichners w aufgefasst werden, das zweite w als dessen Verwendung.

Aus der Tatsache, dass L_5 nicht kontextfrei ist, folgt direkt, dass auch Programmiersprachen wie C oder Pascal nicht kontextfrei sind, da in beiden Sprachen Bezeichner vor ihrer Verwendung deklariert werden müssen und beliebig lang sein können. Daher folgt nach der syntaktischen die semantische Analyse, um dennoch die Korrektheit eingegebener Programme beurteilen zu können.

XTG

In Kapitel 15 wurde bereits kurz auf die Besonderheiten der Tagstruktur in der Sprache des DTP-Systems Quark XPress eingegangen. Im Folgenden zur Veranschaulichung ein paar Auszüge aus einem möglichen Quark XPress-Dokument mit XPress-Tags (XTG):

```
@paragraphstyle1:<B>fett<B><I> kursiv<B> fett und kursiv
@paragraphstyle2:<@characterstyle1>Formatierung durch einen
Absatzstil
<B><I>einzelne öffnende und schließende Tags<B><I>
<BI>kombinierte öffnende und schließende Tags<IB>
```

nur abzählbar viele Sprachen, die mit Grammatiken beschreibbar sind. Mit einer künstlichen Nebenbedingung, dass man beispielsweise nur Tags zulässt, die in der Länge beschränkt sind, erhält man wieder eine endliche Grammatik.

123 Genauer: XML-artige Sprachen gehören zu der Klasse der balancierten Sprachen. Diese liegen in der Klasse der deterministisch kontextfreien Sprachen, diese wiederum in den kontextfreien Sprachen. Die zugehörigen Grammatiken sind balancierte Grammatiken. [Wen08]

```
<BI>spezielles Tag: schließt mehrere öffnende Tags<P>
```

Quelltext IV-2: Auszüge aus einem XTG-Dokument

Das Beispiel demonstriert einige typische Probleme des XTG-Formats:

- XTG verwendet die bereits eingeführten „Toggle Tags“ zur Auszeichnung von Typographie, öffnende und schließende Tags besitzen also die gleiche Syntax. Liefert der Scanner das Token „fett“, so weiß der Parser nicht, ob es sich um den Beginn oder das Ende der entsprechenden Formatierung handelt. Eine korrekte Interpretation kann nur anhand des Kontextes entschieden werden.

- Ein spezifisches Tag kann durch verschiedene Tags geschlossen werden.

- Mehrere Tags können in einem Paar spitzer Klammern (‚< >‘) zusammengefasst werden.

- Ein einzelnes Tag oder die Kombination mehrerer Tags kann durch ein anderes Tag geschlossen werden, welches nicht notwendigerweise Teil des öffnenden Tags sein muss, aber kann.

Eine Grammatik für XTG müsste also kontextsensitive Produktionen der Art:

$$u\alpha v \rightarrow uwv \in P \text{ mit } |uwv| \geq 1 \text{ und } \alpha \in N \text{ und } u, w, v \in V^*$$

enthalten, um die Korrektheit aller möglichen Satzformen zu überprüfen und wäre damit eine Chomsky-Grammatik vom Typ 1.

Die Sprache XTG enthält also syntaktische Konstruktionen, die sich nicht durch kontextfreie Grammatiken beschreiben lassen, die aber für reale Programmiersprachen sehr wichtig sind.

Um solche kontextsensitiven Eigenschaften zu überprüfen, gibt es prinzipiell mehrere Möglichkeiten. Man kann einmal zur Festlegung der Syntax den mächtigeren Grammatiktyp (nämlich kontextsensitiv) verwenden. Dann werden aber automatische Analyseverfahren (Algorithmen) äußerst ineffizient. Daher ist dieser Weg eigentlich nur von theoretischem Interesse. Man kann aber auch die Syntax durch eine kontextfreie Grammatik definieren und dann durch zusätzliche Einschränkungen bestimmte, grammatisch korrekte Satzformen wieder ausschließen. Diese zusätzlichen Einschränkungen überprüfen nun solche syntaktischen Eigenschaften, die ein Satz in XTG haben soll, die aber nicht durch kontextfreie Produktionen der Form

$$P \subseteq N \times V^*$$

ausdrückbar sind. Wie oben bereits erwähnt, werden solche zusätzlichen Bedingungen in der semantischen Analyse beispielsweise bei der Nutzung von Variablenbezeichnern verwendet. In einem PASCAL-Programm etwa muss eine Variable deklariert werden, bevor sie verwendet wird. Eine solche Fernwirkung kann man aber nicht durch eine

kontextfreie Grammatik erzwingen. Ähnliches gilt für die Typkonsistenz in Ausdrücken und bei Wertzuweisungen (nach [Mer97]).

Semantik der Toggle-Tags

Für die semantische Analyse der XTG-Fragmente wurde der XTG-Prozessor um einen Mechanismus erweitert, der die Vorkommen von Tags zählt und diese in einem Hash ablegt. Die Tag-Namen bilden die Schlüssel des Hashes, die Hash-Werte zeigen an, ob das jeweilige Tag aktiv oder inaktiv ist. Daraus ergibt sich die Semantik der „Toggle-Tags“ wie folgt:

$$[\![\text{tag}]\!]_c = \#(< \text{tag} >, c) \ \ mod\ 2) \stackrel{\text{def}}{=} \begin{cases} 0, & < \text{tag} > \text{ an} \\ 1, & < \text{tag} > \text{ aus} \end{cases}$$

Das Tag `<tag>` wird im Kontext `c` gezählt. Bei einem Auftreten dieses Tags wird die bisherige Anzahl durch 2 dividiert, der Rest definiert, ob das Tag in TML als öffnendes oder als schließendes Tag geschrieben wird.

RTF

Ebenfalls problematisch für die syntaktische Analyse war das Rich Text Format (RTF). Im Folgenden ein beispielhafter Auszug:

```
{\rtf1
{\pard text example \par}
{\pard{\b hello world in bold} {\i and in italic.}\par}
{\pard{\b nesting of {\expndtw120 formattings}}\par}
}
```

Quelltext IV-3: Quelltextauszug aus einem RTF-Dokument

Die Probleme im RTF-Format waren teilweise mit den XTG-Problemen vergleichbar:

Den in einem RTF-Dokument benötigten Fonts werden im Kopfbereich des Dokumentes natürlichen Zahlen zugeordnet.[124] Im Inhaltsbereich des Dokumentes erfolgt die Zuordnung von Textabschnitten zu Fonts dann über die Nummern. Da die TML-Engine nur Inhaltsabschnitte aus einem Dokument betrachtet und keinen Einblick in den Header besitzt, musste eine Lösung gefunden werden, den richtigen Font in der TML-Version mitzuführen.

[124] Die Zuordnungen sind potenziell sehr umfangreich, da typographische Varianten von Fonts jeweils einer eigenen Nummer zugeordnet werden.

- RTF benutzt für Maßangaben eine spezielle Einheit mit dem Namen „twips“. TML speichert Maßangaben in dem weithin genutzten Punkte-Format. Die Umrechnung zwischen twips und Punkt führte zu Rundungsfehlern. Um bei mehrfachen Transformationen die Veränderung der Semantik zu verhindern, werden die twips-Werte gekapselt und bei der Transformation von TML nach RTF wieder reaktiviert.

- Benamte Tags werden durch unbenamte geschweifte Klammern wieder geschlossen. Als Lösung für die syntaktische Analyse wurde eine Erweiterung implementiert, welche die Gruppenebenen und die aktuell geöffneten Befehle auf einem Stack ablegt[125].

- RTF fasst mehrere Formatdefinitionen in Gruppen zusammen. Um daraus wohlgeformtes TML zu erzeugen, muss der RTF-Builder diese Gruppen in einzelne Befehle auftrennen.

Semantikerhalt zwischen unterschiedlichen Sprachklassen

Bereits in den ersten Konzepten für ein Verfahren zur Übersetzung von typographischen Metadaten wurde festgehalten, dass Transformationen zwischen den interessanten Formaten potenziell mehrfach bidirektional durchführbar sein sollten, dass aber dennoch die typographische Semantik des Ursprungsdokument nicht verändert oder verloren werden darf.

Es musste ein allgemein tragfähiges Verfahren entwickelt werden, welches alle speziellen Formatierungsangaben mit unbekanntem und nicht abschätzbarem Inhalt aus allen Quellformaten berücksichtigen kann. Zur Umsetzung wurden verschiedene Ideen diskutiert. Die erste Idee war ein allgemeines „Müll“-Tag, welches die nicht übersetzten Formatierungsanweisungen zwischen dem öffnenden und schließenden Tag einklammert. Nach ersten Tests wurde dieser Ansatz wieder verworfen, da der Inhalt teilweise eigene Markups enthielt und Probleme bei der Validierung der TML-Dokumente verursachte.

Als Alternativen wurden u.a. `CData`-Abschnitte oder Kommentare untersucht. Aufgrund der einfachen Umsetzung und der Tatsache, dass Kommentare in den meisten bekannten Sprachen vorhanden sind, wurden schließlich Kommentare – umgeben von einem speziellen Tag – als Containerformat umgesetzt.

Beispiel:

```
<code type="xtg"><!-- <"CMYK",N,S,4,N,0.3,0,0.25,0.4>--
></code>
```

[125] In vereinfachter Form funktioniert dieser Stack wie die Namensraumverwaltung für Bezeichner in klassischen imperativen Programmiersprachen.

An diesem Beispiel ist zu sehen, dass dem umgebenden Tag noch ein Attribut mitgegeben wird, welches das Ursprungsformat des im Kommentar gekapselten Codes angibt. Diese Kapselung nicht definierter Tags findet im TML-Prototyp gleichfalls in HTML und RTF statt.

In HTML wurde zunächst ebenfalls das Code-Tag verwendet. Um aber den Anwendern des Cross Media Publishing-Systems, welche Textinhalte in einem browserbasierten Editor bearbeiten, diese gekapselten Informationen visuell zugänglich zu machen, wurde schließlich das `<img>`-Tag verwendet und die unbekannten Formatierungsangaben im `title`-Attribut abgelegt. Des Weiteren war es noch nötig, das Code-Tag mit einem Typ zu versehen. Damit wurde es möglich, zu unterscheiden, ob der Inhalt in einem Code-Tag aus einer unbekannten typographischen Information aus einem Quellformat stammt oder ob der Code in TML zwar bekannt ist, aber keine Transformation nach HTML definiert wurde.

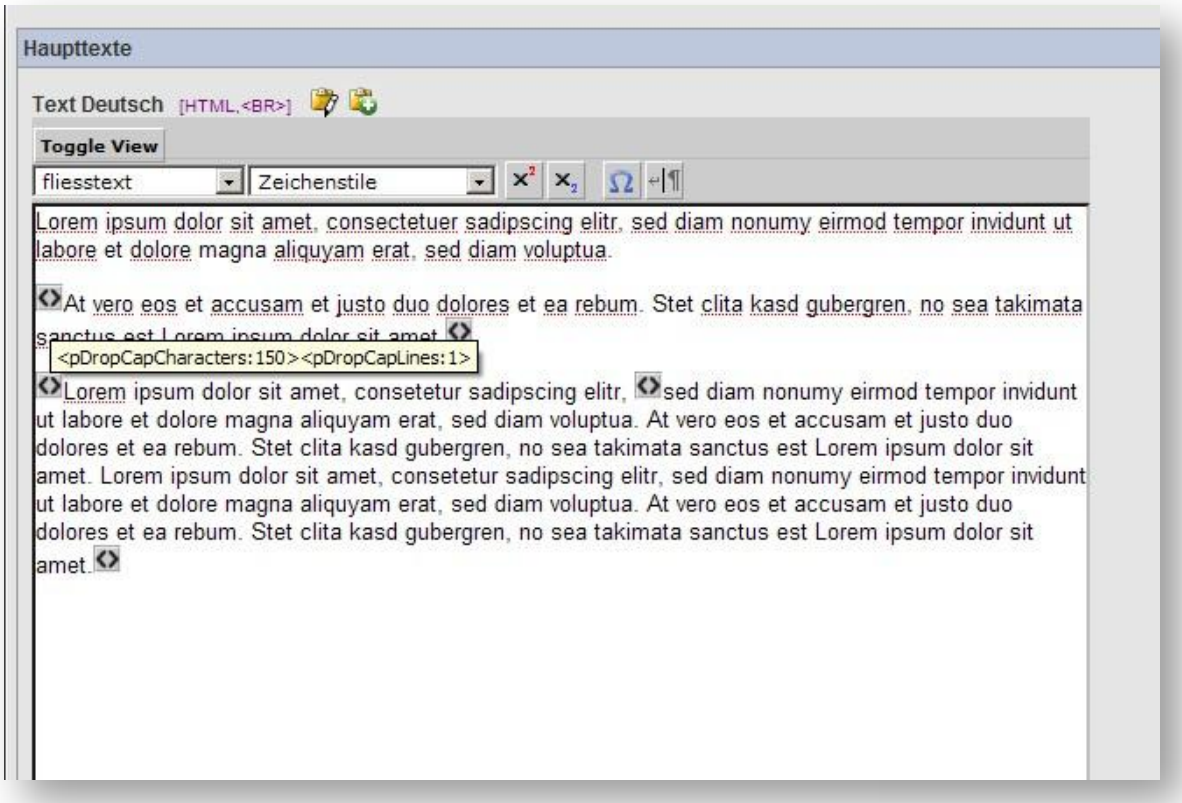

ABBILDUNG IV-10: CLOSED-TAG-DARSTELLUNG IM HTML-BASIERTEN EDITOR DES REDAKTIONSSYSTEMS INTERRED PRINT.

Um den Lesefluss eines Redakteurs im Frontend des Redaktionssystems nicht zu stören, werden komplexe, nicht übersetzte typographische Metadaten über eine kleine Grafik visualisiert. Bewegt der Redakteur die Maus über eine dieser Grafiken, zeigt der Browser das title-Tag an, in welchem das originale Format der Metadaten zur Information für den Redakteur ausgegeben wird, siehe Abbildung IV-10.

Das System bietet an dieser Stelle entsprechend kompetenten Mitarbeitern die Möglichkeit, den Quellcode des Inhaltes direkt zu bearbeiten. Theoretisch könnten Anwender auch auf diesem Weg Einfluss auf die Typographie nehmen, praktisch dürften nur Wenige in der Lage sein, den spezifischen Code der externen Anwendung direkt zu schreiben.

Stilmapping

Wie der erste praktische Einsatz des TML-Prototypen aufgezeigt hat, kommt der Übersetzung von Absatz- und Zeichenstilen eine besondere Bedeutung zu. Man kann davon ausgehen, dass in den externen Ausgabemedien und –objekten die Mächtigkeit der Stilmengen jeweils unterschiedlich ist. Um bei der bidirektionalen Transformation zwischen dem zentralen TML-Format und den externen Formaten die individuellen Stile der Objekte zu unterstützen, müssen die Stilmengen in TML die Obermenge über alle externen Stile bilden.

Definition Anzahl der Stildefinitionen in TML

Sei $k:=$ Anzahl der externen Objekte und $SD(O):=$ die Menge der Stildefinitionen im Objekt O, dann gilt:

$$SD_{TML} = \bigcup_{i=1}^{k} SD_i(O_i)$$

Dennoch kann es in diesem Fall zu einem Verlust an Semantik kommen, da ein externes Objekt mit einer geringeren Mächtigkeit der Stilmenge keine Information über die ursprünglichen Stilzuordnungen behält. Ein dem Code-Tag (siehe voriger Abschnitt) vergleichbarer Mechanismus zum Erhalt semantischer Metadaten wurde für das Stilmapping nicht implementiert.

Beispiel:

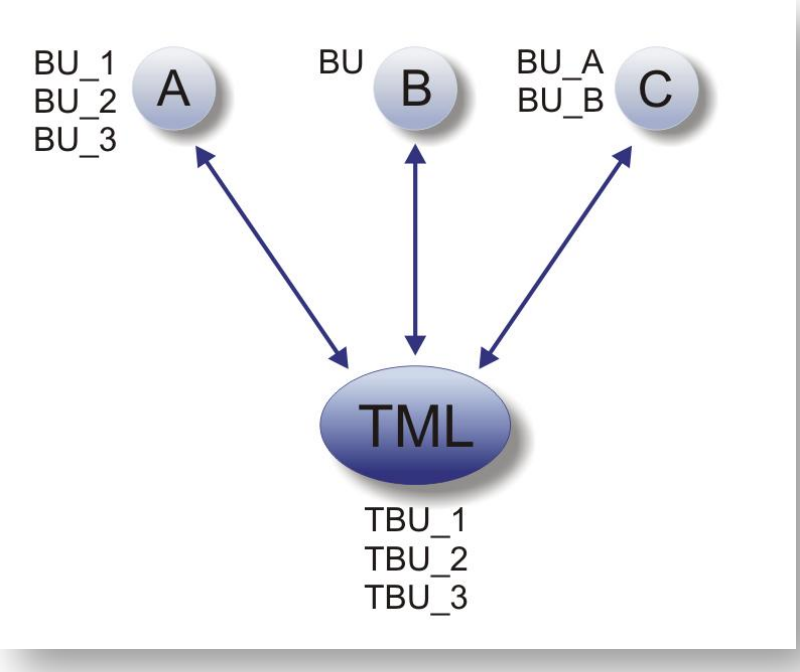

ABBILDUNG IV-11: BEISPIEL FÜR DAS MAPPING-PROBLEM DER BILDUNTERSCHRIFTENSTILE

In einer Installation werden zwischen den drei externen Objekten[126] A, B und C Inhalte ausgetauscht und die zugehörigen Stile transformiert. Im Objekt A sind für Bildunterschriften die drei Stile $\{BU\_1, BU\_2, BU\_3\}$ vorgesehen, für Objekt B wird nur ein Stil für Bildunterschriften $\{BU\}$ verwendet, Objekt C verwendet zwei: $\{BU\_A, BU\_B\}$

Die Anzahl der notwendigen Stildefinitionen im zentralen TML-Format wird durch das mächtigste externe Format bestimmt, in diesem Fall Objekt A mit drei Stilen.

Geht man davon aus, dass die Inhalte zwischen allen drei Objekten übertragen werden sollen, so muss der Nutzer definieren, wie eine geeignete Abbildung zwischen den Stilen aussehen soll. Ein Beispiel zeigt folgende Tabelle:

[126] Es ist hier irrelevant, ob es sich um Print- oder um Online-Objekte handelt, da Stile in allen berücksichtigten Formaten Verwendung finden.

von\nach	A	B	C
A	°	BU_1→BU BU_2→BU BU_3→BU	BU_1→BU_A BU_2→BU_B BU_3→BU_A
B	BU→BU_3	°	BU→BU_A
C	BU_A→BU_3 BU_B→BU_2	BU_A→BU BU_B→BU	°

Daraus konfiguriert der Administrator bei der Installation die Abbildungen in Richtung TML und von TML in Richtung der externen Formate.

Von Extern nach TML:

nach\von	A	B	C
TML	BU_1→TBU_1 BU_2→TBU_2 BU_3→TBU_3	BU→TBU_3	BU_A→TBU_3 BU_B→TBU_2

Von TML nach Extern:

von\nach	A	B	C
TML	TBU_1→BU_1 TBU_2→BU_2 TBU_3→BU_3	TBU_1→BU TBU_2→BU TBU_3→BU	TBU_1→BU_A TBU_2→BU_B TBU_3→BU_A

Wie an diesem Beispiel schon leicht zu ersehen ist, können durch den Verzicht auf den vollständigen Erhalt der Semantik nicht alle denkbaren Abbildungen realisiert werden. Würde der Anwender bei einer Übertragung von B nach A beispielsweise das Mapping $BU \rightarrow BU\_2$ wünschen, so wäre das mit dem TML-System nicht abbildbar, da ein Mapping $BU \rightarrow TBU\_2$ notwendig wäre um auf diesem Weg BU_2 in A zu erreichen, aber von TBU_2 kann nicht BU_A erreicht werden, was ebenfalls gewünscht war.

Während der Entwicklung des TML-Prototypen wurde dieser Mangel als vermutlich rein theoretisches Problem eingeschätzt, da ein je nach Transformationsrichtung unterschiedliches Mapping eher unwahrscheinlich erschien. Der o.a. Wunsch BU aus B in Richtung A nach BU_2 zu transformieren und in Richtung C nach BU_A erscheint unlogisch, da BU_2 und BU_A nicht untereinander sondern jeweils auf andere Formate transformiert werden, siehe Abbildung IV-12:

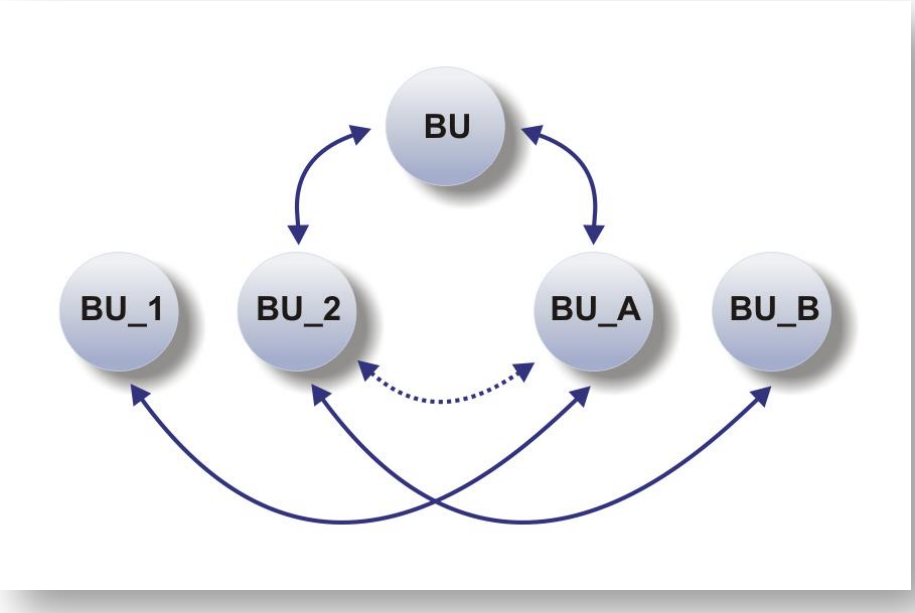

ABBILDUNG IV-12: BILDEN DIE STIL-TRANSFORMATIONEN ZWISCHEN DEN OBJEKTEN EINEN ZYKLUS, LÄSST SICH DAS OHNE ANREICHERUNG VON SEMANTIK MIT DER TML-ENGINE ABBILDEN.

Es war davon auszugehen, dass die Transformation einzelner Stile über alle Objekte hinweg immer einen Zyklus bilden würde, was durch die Erfahrungen in der Praxis bestätigt wurde.

Teil V

Ergebnisse

Kapitel 15

KOSTENREDUKTION

Teil II erläutert die wirtschaftlichen Vorteile der Mehrfachnutzung von Content im Rahmen eines Cross Media Managements in Verlagen. Technische Voraussetzung für eine erfolgreiche Umsetzung ist die medienneutrale Speicherung der Inhalte. Doch vieles, was diesen Anspruch erfüllen will – oft wird mit dem Hinweis auf das Speicherformat XML der Anspruch als erfüllt gesehen – erfordert in der Praxis aufwändige manuelle Anpassungsprozesse.

Strebt nun ein Medienunternehmen im Rahmen von verschiedenen Diversifikationsstrategien die Mehrfachverwertung von Content an, so entstehen Kosten nach den in Kapitel 4 definierten Kostenarten. Folgende Fälle sind zu unterscheiden:

Konvergenzgetriebene intermediäre Diversifikation

Die technologische Konvergenz der klassischen Medienmärkte im Zeitungs- und Zeitschriftenbereich und der modernen Online-Medienmärkte erfordert die Übertragung der Inhalte zwischen technisch bisher isolierten Systemen.

Erfahrungen des Autors aus der Verlags-Praxis zeigen, dass im Jahr 2008, neben diversen XML-Schnittstellen zwischen beteiligten Systemen, selbst in größeren Unternehmen die einzelnen Inhalte noch per manuellen Copy&Paste-Orgien zwischen inkompatiblen Anwendungen übertragen werden. Bei anschließenden Modifikationen in einem der Systeme ist entweder eine manuelle nachträgliche Anpassung notwendig, oder man nimmt in Kauf, dass die Inhalte divergieren.

Diese Medienbrüche sind also nur teilautomatisiert, häufig unidirektional und verlaufen unter dem Verlust der typographischen Metadaten. In Abbildung II-2[127] wird beispielhaft visualisiert, wie mit individuell programmierten Transformationen zwischen und innerhalb von technischen Medienmärkten diese Medienbrüche automatisiert werden können. Wie in Kapitel 4 gezeigt wurde, liegt die Anzahl und damit die Kosten für diese Transformationen in $O(n^2)$.

Ein Redaktionssystem mit zentraler, medienneutraler Datenhaltung und einer Zwischensprache, die alle notwendigen semantischen Informationen übersetzt, reduziert das quadratische Wachstum der zu programmierenden Übersetzer auf ein lineares Wachstum mit $O(n)$. Genau das leistet die entwickelte Architektur mit ihren sprachspezifischen Übersetzern und ihrer zentralen und medienneutralen typographischen Markup Sprache TML.

[127] Betrachtet man die Doppelpfeile als jeweils zwei gerichtete Kanten zwischen zwei Knoten, ergibt sich die Komplexität eines vollständigen, einfachen und gerichteten Graphen.

Die grundlegende Idee einer Zwischensprache ist eine klassische Strategie in der Informatik, um die Komplexität bei der Übersetzung zwischen Sprachen und Formaten zu reduzieren, beispielsweise durch die Java Virtual Machine als zentraler Knoten zwischen den analytischen Compilerteilen und den Compilerteilen für die Synthese.

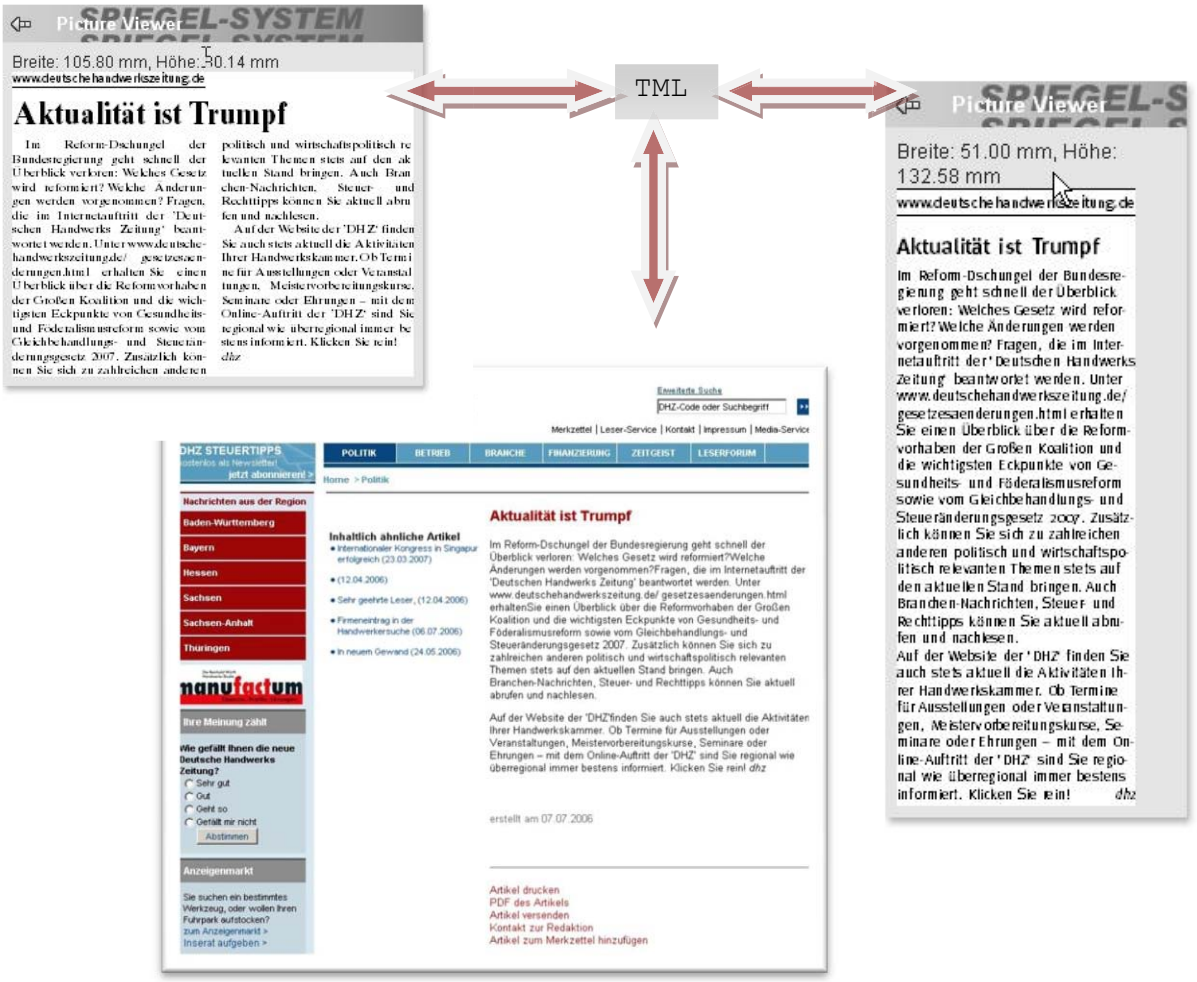

ABBILDUNG V-1: TML ALS ZWISCHENSPRACHE FÜHRT ZU EINEM NUR NOCH LINEAREN WACHSTUM DER ANZAHL DER TRANSFORMATIONEN.

Intramediäre Diversifikation

Bei der Mehrfachverwertung von Inhalten innerhalb einer Mediengattung sind ebenfalls Anpassungen an den Daten notwendig. Zwei Fälle sind hierbei zu unterscheiden:

1. Objektabhängige Typographie
2. Geometrieabhängige Typographie

Objektabhängige Typographie

Inhalte können innerhalb einer Mediengattung in verschiedenen Objekten wiederverwendet werden. Online-Inhalte können beispielsweise auf unterschiedlichen Portalen eines Medienunternehmens vollständig oder teilweise mehrfach ausgegeben werden. Im Druckbereich kann zum Beispiel ein Artikel aus einem Fachmagazin in einem anderen Magazin mit ähnlich geartetem Fachgebiet wiederverwendet werden oder in einem Themenspecial aus dem gleichen Haus mit anderen Inhalten zusammengestellt werden.

Dabei ist es wahrscheinlich, dass sich die Typographie zwischen den Zielobjekten unterscheidet. Ein Fließtext in Objekt A ist beispielsweise mit der Schrift `Times New Roman 10 pt` formatiert, in Objekt B wird der gleiche Fließtext in der Schriftart `Garamond 11 pt` ausgegeben. Bei der Übertragung des Inhalts muss die Schriftart entweder manuell oder per automatisierter Transformation angepasst werden. Die hierbei entstehenden Aufwände sind Nutzungs- bzw. Platzierungskosten. Die Definition dieser Kosten findet sich auf Seite 40. Auch hier wurde gezeigt, dass diese Kosten quadratisch mit der Anzahl der verwendeten Absatz- und Zeichenstile wachsen, da für jedes 2er-Tupel aus Quell- und Zielformat eine eigene Transformation programmiert werden muss.

Mit dem Ansatz der Ausgabeobjekt-unabhängigen Zwischendarstellung der Typographie in TML und den separaten Übersetzungsregeln für jedes Objekt, konnte die Komplexitätsklasse der zu konfigurierenden Übersetzungsregeln von $O(n^2)$ auf $O(n)$ reduziert werden. Im ersten Kunden-Testbetrieb zeigte sich, dass diese theoretische Reduktion einen hohen praktischen Nutzwert hat. Bei diesem Verlag existierte ein Katalog, in dem mehr als 200 Absatzformate definiert waren. Die realistische Anzahl der zu definierenden Regeln wäre vermutlich deutlich vom vollständigen Graphen entfernt, aber auch wesentlich höher als mit dem Ansatz der Zwischendarstellung, gewesen.

Die sprachspezifischen Prozessoren und Builder führen also zu einer Komplexitätsreduktion von $O(n^2)$ auf $O(n)$, wenn man n als die Anzahl der medienspezifischen, externen Anwendungen betrachtet. Zusätzlich reduziert die medienneutrale Formulierung der Typographie in TML als Zwischendarstellung und die Übersetzung von objektspezifischen Absatz- und Zeichenstilen von und nach TML ebenfalls die Laufzeit von $O(n^2)$ auf $O(n)$, wobei n hier die Anzahl der zu berücksichtigenden Stile benennt.

Geometrieabhängige Typographie

Geometrische Metadaten zu einem Inhalt sind grundsätzlich medienabhängige Metadaten und wurden deshalb im ursprünglichen Konzept der TML nicht berücksichtigt[128].

Die Erfahrung aus dem ersten Praxistest in einem Verlag zeigte jedoch, dass die Grenze zwischen medienabhängigen Geometriedaten und medienneutralen typographischen

[128] Siehe auch „Modellierung der primären Entitäten“ auf Seite 104.

Daten nicht klar zu ziehen ist, sondern Fälle von geometrieabhängiger Typographie auftreten können.

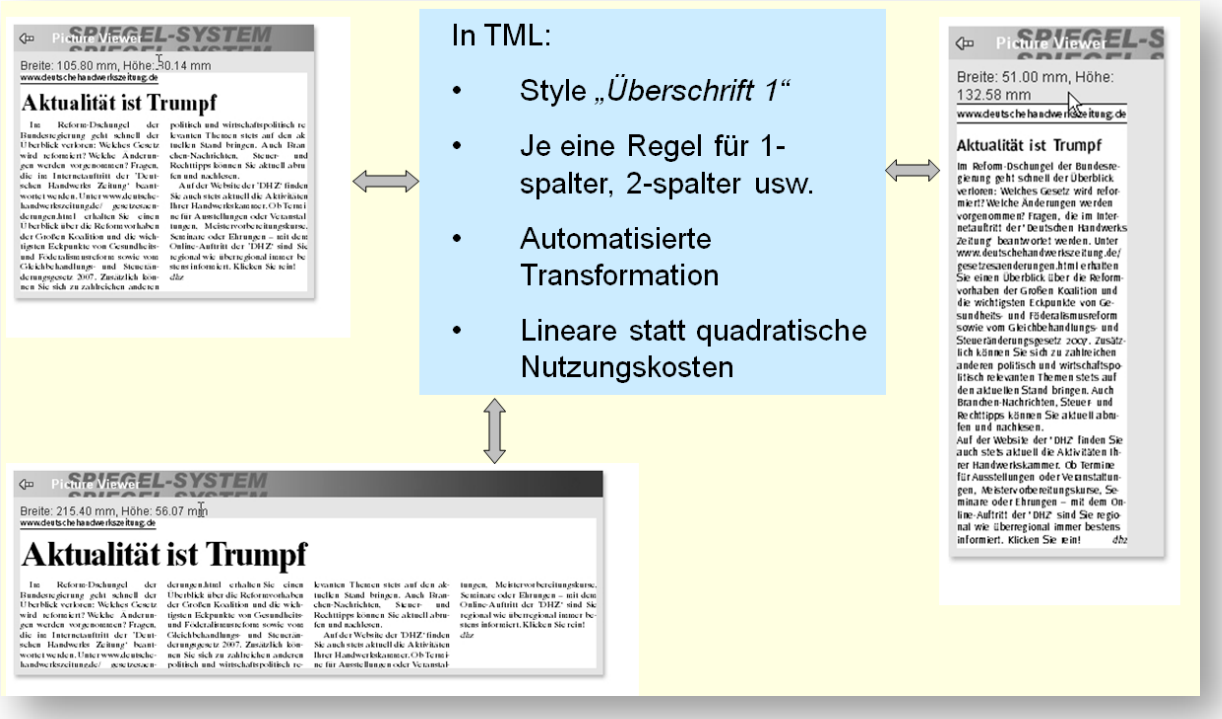

ABBILDUNG V-2: EINE ÜBERSCHRIFT IDENTISCHER LÄNGE WIRD JE NACH SPALTIGKEIT DES ZUGEHÖRIGEN FLIEßTEXTES IN UNTERSCHIEDLICHER GRÖßE FORMATIERT, TROTZ EINES IN ALLEN FÄLLEN IDENTISCHEN ABSATZSTILS FÜR DIE ÜBERSCHRIFT.

Bei der Platzierung eines Inhaltes[129] in unterschiedlichen Objekten oder an unterschiedlichen Orten in einem Objekt, kann es häufig zu Änderungen an der Geometrie kommen. Ein im ursprünglichen Layout zugewiesener Absatzstil einer Überschrift muss dann eventuell anders attributiert werden, damit die Schriftgröße der Überschrift wieder zum Erscheinungsbild des Artikels ‚passt'. Entweder erfolgt die neue Attributierung manuell und muss dann in jedem weiteren Fall manuell wiederholt werden oder es wird eine automatische Anpassung (Transformation) dafür programmiert. Es entstehen wiederholte oder einmalige Nutzungskosten (Platzierungskosten). Abbildung V-2 zeigt ein Beispiel für einen Artikel, der in drei verschiedenen Varianten verwendet werden soll: als 1-Spalter, 2-Spalter und als 4-Spalter.

Während der Erstellung des Textes wird der Überschrift ein Absatzstil zugeordnet. Je nach Organisation des Workflows erfolgt die Zuordnung des Stils direkt durch den Redakteur oder später durch einen Layouter. Es steht ein Satz von Stilen für Überschriften zur Verfügung, aus denen in Abhängigkeit von der Länge der Überschrift ein passender

[129] In diesem Fall ein Zeitungsartikel.

Stil zugeordnet wird[130], beispielsweise „`Überschrift_2`“. Unabhängig von der angestrebten Nutzung des Inhaltes durch den Verlag soll diese Stilzuordnung nur einmal erfolgen. Eine Änderung der Stilzuordnung ist nur noch bei einer inhaltlichen Änderung der Überschrift nötig, wenn sich dabei die Textlänge signifikant ändert[131].

Die Menge der Definitionen eines Absatzstils für die Überschriften sind also 2er-Tupel, bestehend aus Stilname und Spaltenanzahl des Zielformates.

Beispiel:

Stil/Spaltenzahl	1-Spalter	2-Spalter	3-Spalter	4-Spalter
Überschrift_1	Arial, 16 pt	Arial, 36 pt	Arial, 48 pt	Arial, 72 pt
Überschrift_2	Arial, 12 pt	Arial, 18 pt	Arial, 34 pt	Arial, 48 pt
Überschrift_3	Arial, 10 pt	Arial, 12 pt	Arial, 18 pt	Arial, 26 pt

Bereits mit drei Stildefinitionen für ein einziges Inhaltselement (Überschriften), welches potenziell in Vorlagen mit vier verschiedenen Spaltenanzahlen verwendet werden kann, ergeben sich daraus bereits:

$$\binom{4}{2} * 2 = 12$$

zu programmierende Transformationen pro Absatzstil, also insgesamt $3 * 12 = 36$ Transformationen nur für die Überschriftenstile.

Durch eine Erweiterung der Datenstruktur der Transformationsregeln und eine Anpassung der TML-Engine ergab sich hier die Möglichkeit, eine weitere Optimierung der Nutzungskosten zu erreichen. Die Normalisierung der Relation zwischen Absatzstil und Spaltenanzahl führte ebenfalls zu einer Reduktion auf ein lineares Wachstum der zu konfigurierenden Transformationsregeln. Im o.a. Beispiel wären nur noch 12 statt 36 Transformationen zu konfigurieren[132].

---

130 Gleiches gilt für Dachzeilen, Sub-Headlines, etc.

131 Auch hier wäre natürlich eine Automatisierung der Stilauswahl anhand von Textlängenintervallen möglich, die subjektive Einschätzung der ‚richtigen‘ Schriftgröße durch einen Menschen wird für ein gutes Layout aber bevorzugt.

132 Bei einem bisher immer angenommenen gerichteten Graphen müssten es eigentlich 24 Transformationen sein. An dieser Stelle genügt aber eine Berücksichtigung der Spaltenanzahl von TML in eine externe Sprache.

Kapitel 16

ERFAHRUNGEN AUS DER PRAXIS

A wealth of information creates poverty in attention.

(Herbert Simon)

Während der Forschungsarbeiten zu dieser Arbeit wurde die Leistungsfähigkeit des TML-Prototypen in einer redaktionellen Umgebung in einem Verlag mit ca. 100 Redakteuren getestet. Inhalte wurden dort primär für Print-Medien erstellt.

Die Wiederverwendung von Inhalten innerhalb einer Medienart belief sich in einem Beobachtungszeitraum von 12 Monaten auf weniger als 1 Prozent. Die in Kapitel 4 analysierten und definierten Nutzungskosten, die bei der Transformation zwischen Zielobjekten entstehen, waren in diesem Fall nicht relevant. Sie hätten auch ohne die Verbesserung durch TML die Wertschöpfungskette nicht nennenswert belastet. Während des Verfassens dieser schriftlichen Arbeit wurde eine modifizierte Variante bei weiteren Medienunternehmen eingeführt. Dort liegt nach ersten Erkenntnissen die Wiederverwendungsquote innerhalb einer Mediengattung bei ca. 5 Prozent, der Aufwand für die initiale Konfiguration der TML-Regeln dürfte sich in diesem Fall schneller amortisieren.

Im Gegensatz dazu betrug die Wiederverwendung von Print nach Online im ersten Testszenario 97 Prozent im gleichen Zeitraum. Allein die Tatsache, dass dieser Medienbruch durch TML vollständig automatisiert werden konnte, erlaubte dem Verlag die Entwicklung neuer gewinnbringender Wertschöpfungsprozesse.

Ein zweiter Aspekt der Nutzungskosten war die Umplatzierung von Inhalten innerhalb eines Objektes. Er trat unerwartet im Rahmen dieser Testinstallation auf und wurde nachträglich in die Konzeption von TML integriert. Natürlich ist dieser Aspekt nur relevant, wenn nach der Prozessvariante „Content vor Layout" produziert wird. Die Möglichkeit, einen Artikel automatisiert von beispielsweise einem zweispaltigen in ein vierspaltiges Layout zu transformieren, ist hier von großer Bedeutung.

Abschließend wurde der signifikante Fall der Wiederverwendung von Inhalten zwischen verschiedenen Print-Publikationen betrachtet, beispielsweise Serien, Sonderausgaben oder Line Extensions[133]. Im vorliegenden Testszenario konnte die Wiederverwendung in Serien vernachlässigt werden (0,1%). Der Grad der Wiederverwendung in Sonderausgaben war dagegen mit 90% sehr hoch, aber die geringe Erscheinungshäufigkeit von einer Ausgabe pro Jahr hat die Bedeutung trotzdem reduziert.

[133] Line Extension beschreibt die Übertragung einer Marke auf ein neues Produkt innerhalb der gleichen Produktkategorie. Beispielsweise ein neues Sportwagenmagazin, welches unter der Marke eines bekannten Automagazins veröffentlicht wird.

Im Gegensatz dazu war das Modell der Line Extensions sehr interessant. Im betrachteten Szenario existierten zwar keine Line Extensions, aber in einer späteren Testumgebung in einem anderen Unternehmen wurde eine Wiederverwendungsquote von 25% festgestellt.

Ein weiterer Ansatz, der die Bedeutung einer medienneutralen Typographie unterstreicht und in jüngerer Zeit immer populärer geworden ist, ist der sogenannte ‚Newsroom'. Redaktionsteams produzieren hier Inhalte mit dem Gedanken der medienunabhängigen Verwendung, besondere Eigenschaften wie sprechende Hyperlinks oder Seitenverweise, die sich nur in einem Medium abbilden lassen, werden vermieden. Alle Prozesse ab der Content-Erfassung benötigen für die effiziente Übertragung in ein Zielmedium eine neutrale Typographie mit automatisierter Übersetzung.

Kapitel 17

FAZIT

Zu Beginn des Jahres 2002 begannen die ersten Gespräche zwischen der Fachgruppe Programmiersprachen der Universität Siegen unter Leitung von Prof. Merzenich und der InterRed GmbH über die Herausforderungen in einem relativ jungen Anwendungsfeld, das man mit „Single Source – Multi-Media Publishing" umschreiben konnte. Ein neuer Begriff für diese Anwendungsfeld trat damals häufiger im Kontext von Content Management Systemen auf: „Cross Media Publishing[134]".

In den meisten Medienunternehmen war die parallele Erfassung, Verwaltung, Steuerung und Distribution von Inhalten für die Zielmedien Print und Online ein historisch gewachsener Tatbestand. Neue und umfassendere Diversifikationsstrategien über Mediengrenzen hinweg und daraus resultierende neue Wertschöpfungsketten waren damals bereits intensiver Gegenstand der Forschung. Heute, gegen Ende des Jahres 2008, hat das strategische Unternehmensziel Cross Media Publishing die operative Praxis auf breiter Front erreicht. In nur wenigen Jahren hat sich der wenig hinterfragte Parallelbetrieb von Redaktions- und Publishingsystemen für Print und Online zu einem kostentreibendem Dilemma ausgewachsen, welches zumal kaum zusätzliches kreatives Potenzial bietet und nur eine Folge technischer Unzulänglichkeiten ist.

Der Wunsch nach integrierten Systemen mit nur einer Datenquelle und nur einmaliger Erfassung von Content hat zu zahlreichen Erweiterungen etablierter Anwendungssysteme geführt. Von der einfachen Textverarbeitung für Endbenutzer bis zur komplexen Unternehmenssoftware wird heute das Beherrschen von medienübergreifender Publikation und das ‚medienneutrale' Speichern von Inhalten fast schon selbstverständlich erwartet.

Getrieben werden diese Änderungen der Unternehmensstrategien und die daraus folgende technische Entwicklung von zahlreichen Studien und Veröffentlichungen über das geänderte Mediennutzungsverhalten quer durch alle Nutzerschichten. Die vor dem Platzen der ersten Internet-Blase Ende der 1990er Jahre häufig gehörte Theorie, dass Print in ein paar Jahren keine Rolle mehr spielen würde, hat sich zwar mittlerweile als falsch herausgestellt, aber am Umkehrschluss festzuhalten, dass Online nur ein ‚nice to have' ist, ist in fast allen Bereichen heute keine Zukunftsperspektive mehr.

Eine Analyse der European Interactive Advertising Association bestätigte Ende 2007, dass die deutsche Medienwirtschaft gerade eine historische Wende erlebt hatte: Erstmals haben die Konsumenten mehr Zeit im Internet verbracht als vor dem Fernseher.[135] Jens

---

[134] Oft wird auch die alternative Schreibweise „Crossmedia Publishing" verwendet. Im Google Battle liegen beide interessanterweise fast gleich auf.

[135] http://www.emar.de/emar/NL/news/artikel/2007/11/52301/index.html

Schröder[136] präsentiert auf retromedia.de fein säuberlich den Nekrolog der Publikums- und Special Magazine von 2006 bis 2008, mit steigender Tendenz. Starben 2006 noch 82 Magazine, waren es 2007 schon 86 und 2008 schloss mit (mindestens) 93 Print-Produkten, die in diesem Jahr ihr Leben ausgehaucht haben[137].

Arbitron und Edison Media Research veröffentlichen seit 1998 Studien zum Thema Medianutzung in den USA. In ihrer Studie „Internet and Multimedia 2006“[138] wird hervorgehoben, dass die Nutzer die Zeit für ihre Mediennutzung nicht beliebig ausdehnen können. Immer mehr Medien müssen sich die Zeit pro Tag, die ein Mensch in sie investiert, teilen und das Internet kannibalisiert dabei vermehrt die klassischen Medien. Treibend ist hier der Wunsch nach dem „On-Demand Media“, der Möglichkeit, Medien unabhängig von Zeit und Ort bei Bedarf sofort nutzen zu können.

Hugo E. Martin berichtet in seinem lesenswerten Medienblog[139] vom VDZ Vertriebsforum 2006[140] über die wenig erfreulichen Entwicklungen und Aussichten beim Einzelverkauf von Zeitungen und Zeitschriften (siehe Abbildung V-3).

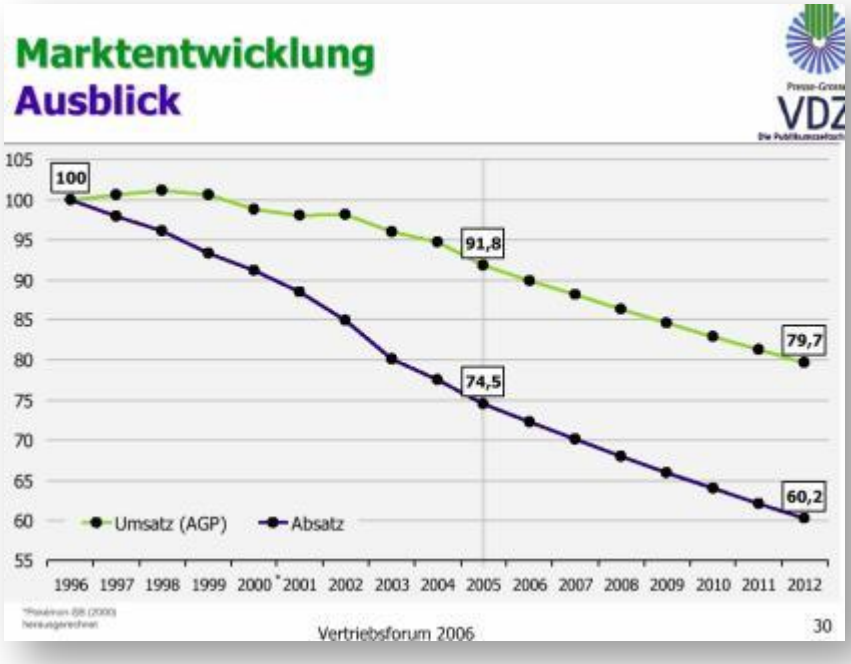

ABBILDUNG V-3: STATISTIK BIS 2005 UND PROGNOSE VON UMSATZ UND ABSATZ EINZELVERKAUF ZEITUNGEN UND ZEITSCHRIFTEN (QUELLE: VDZ)

136 http://popkulturjunkie.de/

137 http://www.retromedia.de/?p=225

138 http://www.arbitron.com/downloads/im2006study.pdf

139 http://hemartin.blogspot.com/

140 http://hemartin.blogspot.com/2006/09/vertriebsforum-2006-jede-menge-grund.html

Die Veränderung der Mediennutzung ist unbestreitbar und sie verläuft in einem vergleichsweise atemberaubenden Tempo. Die notwendigen Weichenstellungen im Management der Medienunternehmen zur Erschließung neuer Umsatzpotentiale und die Entwicklung neuer Geschäftsmodelle kann dabei nicht unabhängig von den verfügbaren Technologien betrachtet werden. Sie sind teilweise Ergebnis neuer Geschäftsmodelle, teilweise aber auch Katalysator für die Entwicklung neuer Modelle.

Technologien, die den teuren und fehleranfälligen Parallelbetrieb von medienspezifischen Anwendungen vermeiden, sind mittlerweile zahlreich vorhanden. Da etablierte Industriestandards vor allem im Bereich der Druckmedienerstellung nicht ohne weiteres ersetzt werden können, war es nötig, die Übertragung der Inhalte zwischen den beteiligten Systemen zu automatisieren. Das ist heute weitgehend gelungen, fast immer spielt XML dabei eine wichtige Rolle. Auch das medienneutrale Management von Bilddaten ist theoretisch im Wesentlichen gelöst. Als Datenmaster wird ein Bild in der höchsten verfügbaren oder benötigten Auflösung gesichert, kleinere und schlankere Formate können daraus abgeleitet werden. Skalierungen und eventuell nötige Farbraumtransformationen lassen sich komplett automatisieren.

Für das Layout und die Typographie von Textinhalten gab es bisher nur automatisierte Verfahren, die für bestimmte Kommunikationspartner individuell erstellt wurden. Es war Ziel dieser Arbeit, ein generelles Übertragungsverfahren für bestehende Industriestandards wie RTF, Quark XPress, Adobe Indesign etc. zu entwickeln. Dazu waren einige Vorüberlegungen notwendig. Die layouttechnische Seitenbeschreibung ist im Grunde ja medienspezifisch, so dass die Transformation von Formatierungen über Mediengrenzen hinweg nicht per se notwendig erschien. Die Formatierungen lassen sich in zwei Klassen abgrenzen:

- Als medienspezifisch kann man die Geometrie eines Dokumentes bezeichnen, beispielsweise Seitenformat, Ränder, Anordnung von Bildern, etc.

- Die Typographie von Überschriften, Fließtexten, also der einzelnen Textinhalte, ist fast immer medienneutral. Ein in einem gedruckten Artikel gefettetes Wort soll auch bei einer Online-Ausgabe gefettet werden. Die Fettung ist ein Teil der Dokumentsemantik. Die Grenze zwischen Geometrie und Typographie ist dabei nicht strikt.

Ein automatisierter Austausch von Dokumenten ohne Verlust der aufwändig erstellten Typographie würde den Ansatz der medienneutralen Publikation weiter verbessern. Die Ziele dieser Arbeit lassen sich wie folgt zusammenfassen:

Nach dem abstrakt formulierten Wunsch, typographische Metadaten in einem crossmedialen Produktionsszenario zu berücksichtigen, war zunächst die Analyse der zu erwartenden Prozesse und der sich daraus ergebenden Problemstellungen notwendig. Anschließend wurde die Eignung bestehender Technologien für die Lösung dieser Problemstellungen untersucht.

Mit der Entwicklung der typographischen Markupsprache TML, einer XML-basierten Sprache zur medienneutralen Formulierung von typographischen Metadaten und einer Transformationsanwendung, die auf Basis variabler Grammatiken typographische Metadaten gezielt übersetzen kann, wurde eine Serviceanwendung geschaffen, die sich relativ

einfach in bestehende Produktionsszenarien einbinden lässt. Voraussetzung ist ein zentrales Redaktions- oder Publishingsystem, welches die Abläufe zentral koordiniert. Dabei konnten die in der Analysephase identifizierten Komplexitätsklassen an mehreren Stellen von einem quadratischen zu einem linearen Wachstum reduziert werden.

Unter der Voraussetzung, dass Print- und Online-Objekte mit einer gewissen Erscheinungsfrequenz produziert werden und die Objektpalette in einem Medienunternehmen die Mehrfachverwertung von Content als wertschöpfendes Geschäftsmodell attraktiv macht, können durch den Einsatz der automatisierten Transformation der Typographie mit Hilfe der TML-Engine die Kosten für die wiederholte Applikation der Typographie reduziert und die Qualität der Ergebnisse gesteigert werden.

Noch während der Arbeiten konnten erste Erfahrungen bei einem praktischen Einsatz in einem größeren Verlagsunternehmen gesammelt und für die weitere Entwicklung berücksichtigt werden. Die Architektur ist so ausgelegt, dass Übersetzer für weitere Sprachstandards an klar definierten Schnittstellen einfach ergänzt werden können.

Kapitel 18

AUSBLICK

Bereits während der ursprünglichen Konzeption der Transformationsengine wurden zur Reduktion der Komplexität bewusst zwei typographisch wichtige Inhaltstypen ausgeblendet: Formeln und Tabellen werden bisher nicht unterstützt.

- Für die medienneutrale Beschreibung von Tabellen hat sich noch kein Standard durchgesetzt. Ein umfangreiches Modell bietet das ursprünglich vom US-amerikanischen Verteidigungsministerium definierte „CALS Table Model“[141]. Der ursprüngliche Standard ist unter der OASIS zuletzt 1995 veröffentlich worden. Später hat die Docbook-Initiative die Weiterentwicklung der CALS-Tabellen-DTD[142] übernommen.

- Das Hauptproblem war die fehlende oder mangelhafte Unterstützung von Tabellen in den DTP-Systemen. Jeder Anbieter verfolgt hier einen eigenen Ansatz, der sich zudem nicht immer über das jeweilige XML-basierte Austauschformat serialisieren lässt. Für die Transformation von Tabellen müssen aktuell individuelle Übersetzer programmiert werden.

- Als Standard für die Darstellung (und für die Bedeutung) einer Formel kann MathML[143] angesehen werden. MathML wird mittlerweile von den bekannten Office-Paketen unterstützt und kann von manchen Internetbrowsern direkt dargestellt werden, für die anderen Browser existieren Plugins. Die beiden wesentlichen DTP-Anwendungen Adobe Indesign und Quark XPress bieten zwar keine native Unterstützung für MathML, sie kann aber mit Erweiterungen von Drittanbieteren nachgerüstet werden.

Eine Erweiterung von TML um die Möglichkeit der Transformation von Formeln scheint naheliegend, da mit MathML ein weitgehend akzeptierter Standard vorliegt.

Die DTP-Anwendungen enthalten noch weitere Funktionen wie Objektstile, Ebenen, Gruppierungen, die zum Teil die Typographie eines Dokumentes beeinflussen. Eine Berücksichtigung bei der Transformation zwischen DTP-Systemen wäre wünschenswert.

Ein weiteres, bisher nicht gelöstes Problem trat spezifisch bei der Bearbeitung von DTP-Dokumenten in Indesign auf. Nachträgliche, individuelle Formatierungen an einem bereits über einen Stil formatierten Text, werden additiv oder subtraktiv im Namen des Stils angegeben. Angenommen, eine Überschrift sei mit einem Stil formatiert, so dass die

---

[141] http://oasis-open.org/specs/tm9502.html

[142] http://www.docbook.org/xml/4.2/calstblx.dtd

[143] http://www.w3.org/Math/

Schrift größer und gefettet wurde. Wenn jetzt durch einen Grafiker beispielsweise von einem Wort in der Überschrift die Fettung entfernt wird, dann verändert Indesign den Stilnamen durch die Subtraktion der Änderung.

Vorher:

```
<style1>Das ist die Überschrift</style1>
```

Nachher:

```
<style1>Das ist <style1-bold>die</style1-bold> Überschrift</style1>
```

Für das XTG-Format ist dieses Problem einfacher zu lösen, da dort die bekannten Toggle-Tags eingesetzt werden, für die bereits eine Behandlung implementiert ist.

Bis zum Abschluss dieser schriftlichen Arbeit wurde die Transformation Engine bei weiteren Medienunternehmen in Betrieb genommen. Es hat sich gezeigt, dass eine weitere Reduktion der Komplexität für den praktischen Einsatz von Vorteil wäre. Formatierungen werden im Wesentlichen nur über Absatz- und Zeichenstile vorgenommen, die Konfiguration kann sich auf den Umgang mit Stilen beschränken. Dann könnte auch die Struktur der Regelhaltung vereinfacht werden. Die Speicherung in einer Datenbank könnte durch einfache, textbasierte Datenstrukturen ersetzt werden, die in Textdateien gespeichert werden.

Teil VI

Anhänge

Umgesetzte TML-Auszeichnungselemente

Wurzelelement: <typoml>

Umklammert den gesamten TML-Code.
Gruppe: typography
Attribute: keine

**Zeilenumbruch:
**

Bewirkt einen Zeilenumbruch im Text.
Gruppe: special
Attribute: keine

Kapselung <code>

Kapselt unbekannten Quellcode.
Gruppe: special
Attribute: type (any, xtg, tml)

Tabulator <tab>

Stellt einen Tabulator dar.
Gruppe: special
Attribute: align (right)

Tabulator <bed>

Stellt einen breaking-em-dash dar.
Gruppe: special
Attribute: keine

Kursive Darstellung: <i>

Stellt Text kursiv dar.
Gruppe: characterstyle
Attribute: keine

Fette Darstellung: <b>

Stellt Text fett dar.
Gruppe: characterstyle
Attribute: keine

Schrift: <font>

Zeichnet die Schrift aus.
Gruppe: characterstyle

Attribute: face (<string>)
size (<integer>)
cpg (<string>)

Kerning: <kern>

Gibt für die folgenden zwei Zeichen das Kerning an.
Gruppe: characterstyle
Attribute: letterspacing (<decimal>)

**Klasse: **

Umklammert eine Klassendefinition.
Gruppe: characterstyle
Attribute: class (<string>)
cpg (<string>)

**Style: **

Umschließt eine Styledefinition.
Gruppe: paragraphstyle
Attribute: class (<string>)
cpg (<string>)
leftindent (<decimal>)
firstline (<decimal>)
rightindent (<decimal>)
leading (<decimal>)
spacebefore (<decimal>)
spaceafter (<decimal>)
locktobaseline (<string>)
language (<string>)

Schnittstellen der TML-Engine

Alle Transformationen werden mit der Subroutine `process()` des Moduls TransformationEngine gestartet. Abhängig vom gewählten Regelschema und der gewählten Richtung wird der Prozessor der jeweiligen Ursprungssprache aufgerufen.

Handelt es sich beim Quellformat nicht um TML, so wird vom jeweiligen Prozessor-Modul der Eingabestring verarbeitet und der resultierende TML-String zurückgeliefert. Optional besteht auch die Möglichkeit, sich den Plain-Text zurückgeben zu lassen.

Handelt es sich beim Quellformat um TML, so wird der Eingabestring zunächst an den TML-Prozessor übergeben. Dieser zerlegt den TML-String in Tags und übergibt diese einzeln an das Builder-Modul der gewünschten Zielsprache. Das entwickelte Kernmodul trägt den Namen TransformationEngine.pm und wird vom Redaktionssystem aus aufgerufen.

TransformationEngine.pm

Das Modul `TransformationEngine.pm` dient als Containermodul, welches den jeweils zu benutzenden sprachspezifischen Parser aufruft. Ein bereits existierendes Datenbankhandle wird von hier aus an die Datenbankschnittstelle übergeben.

Externe Schnittstellen

process()

Die Subroutine `process()` dient dem Aufruf der Transformationsengine.

Aufruf[144]

`process()` erwartet als Parameter einen Hash. Für gewisse Schlüssel existieren Default-Werte, die benutzt werden, wenn dieser Schlüssel beim Aufruf nicht benutzt wird.

Schlüssel	Standardwert	Beschreibung
DEBUG	0	Ausgabe von Debug-Informationen
VALIDATE	0	Bei TML-Erzeugung sinnvoll
PLAIN	0	Zusätzlich Rückgabe von Plain-Text, nur bei TML-Erzeugung möglich
OS	Windows	Zu verwendendes Betriebssystem bzgl. Codepages
INPUT	keiner	Das Dokument/Fragment im Quellformat
SCHEMA	keiner	Das zu verwendende Regelschema

Rückgabe

- `$output`

[144] Die Reihenfolge der Parameter entspricht im Folgenden immer der internen Reihenfolge der Übergabe- und Rückgabewerte.

- Enthält eine Referenz auf einen Hash. Dieser enthält als Schlüssel Formatbezeichnungen und als Werte die Rückgabefragmente in den entsprechenden Formaten.

- `$errCode`

- Gibt an, ob die Transformation erfolgreich war.

- `$errDesc`

- Enthält eine Meldung zur Erläuterung von $errCode.

Gemeinsame Konzepte der Builder-Module

Die sprachspezifischen Builder arbeiten mit dem TMLProcessor-Modul zusammen, welches die TML-Daten zerlegt und dann an die jeweiligen Builder übergibt.

Da die Builder-Module die Eingabedaten tag-weise und nicht wie die Prozessor-Module als einzelnen String erhalten, wird vor jeder Transformation die `init()`-Subroutine aufgerufen. In dieser werden alle globalen Variablen initialisiert, die für das Zusammenspiel von TMLProcessor und Builder wichtig sind. Findet der TML-Prozessor ein öffnendes TML-Tag, so wird dieses an die Subroutine `startingTag()` des Builders übergeben. Hier wird nun per Fallunterscheidung und mit Hilfe einer Anfrage an die Datenbank das korrekte Äquivalent in der Zielsprache ermittelt. Bei `<div>`- und `<span>`-Tags werden die Attributwerte des Attributs „class“ zusätzlich gegen die Stil-Datenbank geprüft, um den korrekten Stil-Namen zu ermitteln.

Übergebener Plain-Text wird über die Subroutine `plainText()` behandelt. Die Routine `comment()` behandelt Code-Tags aus TML. Dies ist möglich, da der Code-Tag in TML immer innerhalb eines Kommentars steht und keine anderen Kommentare in TML vorhanden sind.

TML-End-Tags werden ebenso wie Start-Tags in einer eigenen Subroutine behandelt, nämlich in der `endingTag()`-Subroutine. Hier wird analog zur `startingTag()`-Subroutine für jedes übergebene schließende Tag das Äquivalent in der jeweiligen externen Sprache gesucht.

Interne Schnittstellen

init()

Die Subroutine `init()` setzt die Variablen auf bestimmte Werte zurück, welche vor der Transformation mit Standardwerten belegt werden müssen. Dies gewährleistet, dass die Builder auch unter mod_perl fehlerfrei funktionieren. Weiterhin wird hier das zu verwendenden Regelschema übergeben.

Aufruf

- `$ruleset`

- Enthält das bei der Transformation zu benutzende Regelschema.

comment()

Diese Funktion wird aufgerufen, falls der TML-Prozessor einen Kommentar im TML-Eingabestring identifiziert hat. Hierbei handelt es sich um gekapselten, unbekannten Code, der sprachspezifisch weiterverarbeitet werden muss.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.
- `$ruleset`
- Enthält das zu verwendende Regelschema.
- `$comment`
- Enthält den gefundenen Kommentar.

Rückgabe

- Enthält den verarbeiteten Kommentar. Diese Verarbeitung ist abhängig vom zu generierenden Zielformat und dem Wert des type-Attributes des Code-Tags.

startingTag()

Diese Funktion wird aufgerufen, falls der TML-Prozessor einen Start-Tag im TML-Eingabestring vorgefunden hat, die eigentliche Transformation in die Zielsprache findet also hier statt. Weiterhin werden hier die verwendeten Styles überprüft und unbekannter Code gekapselt.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.
- `$ruleset`
- Enthält das zu verwendende Regelschema.
- `$startingTag`
- Enthält den gefundenen Start-Tag
- `@attributeList`
- Enthält die zu diesem Tag zugehörigen Attribute.

Rückgabe

- Enthält den transformierten Start-Tag.

endingTag()

Diese Funktion wird aufgerufen, falls der TML-Prozessor einen End-Tag im TML-Eingabestring vorgefunden hat, die eigentliche Transformation in die Zielsprache findet also hier statt. Falls der vorangegangene Start-Tag gekapselt wurde, muss auch der betroffene End-Tag hier gekapselt werden.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.
- `$ruleset`
- Enthält das zu verwendende Regelschema.
- `$endingTag`
- Enthält den gefundenen End-Tag

Rückgabe

- Enthält den vollständig transformierten End-Tag.

plainText()

Diese Funktion wird aufgerufen, falls der TML-Prozessor reinen Text im TML-Eingabestring identifiziert hat. Hier wird dann auch die Bestimmung der Codepage anhand der verwendeten Stile und Schriftart durchgeführt, sowie die Transformation des Textes in eine andere Codepage vollzogen.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.
- `$plainText`
- Enthält den zu transformierenden Plain-Text.

Rückgabe

- Enthält den vollständig transformierten Nutztext.

Gemeinsame Konzepte der Processor-Module

Alle Prozessor-Module haben die bezüglich ihres Aufrufs und ihrer Rückgabewerte identische `process()`-Subroutine als gemeinsame Basis.

Interne Schnittstellen

process()

Die `process()`-Subroutine wird von dem Modul `TransformationEngine.pm` aufgerufen und stellt somit die gemeinsame Schnittstelle der sprachspezifischen Prozessor-Module zur Verfügung.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.
- `$input`
- Enthält die zu transformierenden Daten.
- `$ruleset`
- Enthält das zu verwendende Regelschema.
- `$os`
- Spezifiziert das verwendete Betriebssystem, da dieses für die Codepage-Umwandlung benötigt wird.
- `$plain`
- Flag, welches angibt, ob reiner Text zurückgeliefert werden soll.
- `$debug`
- Flag, welches angibt, ob Debug-Informationen ausgegeben werden sollen.
- `$validate`
- Flag, welches angibt, ob der erzeugte TML-Code gegen das TML-Schema verifiziert werden soll.

Rückgabe

- Enthält einen Zeiger auf die Hash-Tabelle, welche die Ergebnisse der Transformation enthält.

TMLProcessor.pm

Der TML-Prozessor liest TML ein und startet gemäß den übergebenen Parametern die sprachspezifischen Builder, welche für die eigentliche Transformation in die jeweiligen Exportformate zuständig sind.

Interne Schnittstellen

comment()

Diese Funktion wird aufgerufen, falls das Modul `XML::Parser` einen Kommentar im TML-Eingabestring identifiziert hat. Daraufhin werden die entsprechenden Funktionen des durch die Übergabe von `process()` bestimmten Builders aufgerufen. Das Ergebnis wird schließlich an den Anfang des Ausgabearrays angehängt.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.

Rückgabe

- Enthält den gefundenen Kommentar.

Seiteneffekte

- Hängt das Ergebnis der Transformation an das Array `@outputs` an.

startingTag()

Diese Subroutine wird aufgerufen, falls das Modul `XML::Parser` einen Start-Tag im TML-Eingabestring identifiziert hat. Daraufhin werden die entsprechenden Funktionen des durch die Übergabe von `process()` bestimmten Builders aufgerufen. Das Ergebnis wird schließlich an den Anfang des Ausgabearrays angehängt. Weiterhin wird hier das Array der gefundenen Attribute auf das einheitliche Format `(Attributname / Attributwert / Attributtyp)` gebracht.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.

Rückgabe

- `$startingTag`
- Enthält das gefundenen Start-Tag.
- `$attribs`

- Enthält die zum Tag zugehörigen Attribute.

Seiteneffekte

- Hängt das Ergebnis der Transformation an das Array `@outputs` an.

endingTag()

Diese Subroutine wird aufgerufen, falls das Modul `XML::Parser` einen End-Tag im TML-Eingabestring identifiziert hat. Daraufhin werden die entsprechenden Funktionen des durch die Übergabe von `process()` bestimmten Builders aufgerufen. Das Ergebnis wird schließlich an den Anfang des Ausgabearrays angehängt.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.

Rückgabe

- Enthält das gefundene End-Tag.

Seiteneffekte

- Hängt das Ergebnis der Transformation an das Array `@outputs` an.

plainText()

Diese Subroutine wird aufgerufen, falls das `XML::Parser`-Modul nur reinen Text im TML-Eingabestring vorgefunden hat. Daraufhin werden die entsprechenden Funktionen des durch die Übergabe von `process()` bestimmten Builders aufgerufen. Das Ergebnis wird schließlich an den Anfang des Ausgabearrays angehängt.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.

Rückgabe

- Enthält den gefundenen Text.

Seiteneffekte

- Hängt das Ergebnis der Transformation an das Array `@outputs` an.

XTGBuilder.pm

Die Vorgehensweise und die Subroutinendeklarationen des XTGBuilder-Moduls entsprechen der oben beschriebenen allgemeinen Vorgehensweise der Builder.

XTGProcessor.pm

Dieses Modul leistet die Transformation von XTG nach TML. Neben der reinen Parserfunktionalität werden von diesem Modul auch die Anfragen an die Datenbankschnittstelle gestellt.

Interne Schnittstellen

process()

Diese Routine stellt die Schnittstelle des Parsers zum TransformationEngine-Modul dar, sie wird von der Transformationsengine aufgerufen, um dem Parser die zu transformierenden Daten zu übergeben.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.
- `$input`
- Enthält die zu transformierenden Daten als String.
- `$schema`
- Das Transformations-Schema.
- `$os`
- Spezifiziert das verwendete Betriebssystem, da dieses für die Codepage-Umwandlung benötigt wird.
- `$plain`
- Flag, welches angibt, ob reiner Text zurückgeliefert werden soll.
- `$debug`
- Flag, welches angibt, ob Debug-Informationen per STDERR ausgegeben werden sollen.
- `$validate`

- Flag, welches angibt, ob der erzeugte TML-Code gegen das TML-Schema verifiziert werden soll.

Rückgabe

- Gibt eine Referenz auf einen Hash zurück, wie er bei der Beschreibung von TransformationEngine angegeben ist.

handleUnknownTag()

Diese Subroutine dient der Behandlung von unbekannten Tags. Diese werden, falls sie nicht in der Datenbank vorhanden sind, von einem Code-Tag eingekapselt.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.
- `$unknownTag`
- Enthält den unbekannten Tag.

Rückgabe

- Falls der Tag in der Datenbank hinterlegt ist, enthält dieses Skalar den gefunden Tag, ansonsten die oben beschriebene Code-Kapselung.

toggle()

Diese Subroutine ändert den Zustand von Bezeichnern, die nur genau zwei Zustände kennen, um zwischen diesen bei mehrfachem Auftreten eines Bezeichners zu wechseln.

Aufruf

- `$key`
- Enthält das Zielfeld des umzustellenden Zustandes.

Seiteneffekte

- Der Zustand des entsprechenden Tags wird in den Hashes angepasst.

resetToDefault()

Diese Subroutine setzt sämtliche toggle-Tags auf ihre Standardwerte zurück.

Seiteneffekte

- Sämtliche in den entsprechenden Hashes gespeicherte Zustände von Bezeichnern werden zurückgesetzt.

resetToPlain()

Diese Subroutine setzt sämtliche toggle-Tags zurück, welche durch `P` geschlossen werden.

Seiteneffekte

- Alle betroffenen, in den entsprechenden Hashes gespeicherten Zustände von Bezeichnern werden zurückgesetzt.

resetByNewline()

Diese Subroutine setzt die Zustands-Hashes auf ihren Urzustand zurück.

Seiteneffekte

- Alle betroffenen, in den entsprechenden Hashes gespeicherten Zustände von Bezeichnern werden zurückgesetzt.

handleLine()

Diese Subroutine behandelt eine Eingabezeile. Da eine Zeile in XTG einem Absatz entspricht, wird der Inhalt einer Zeile nach entsprechender Transformation in `div`-Tags eingeschlossen.

Aufruf

- `$line`
- Enthält die zu behandelnde Eingabezeile.

Seiteneffekte

- Die Funktion `resetByNewline()` wird aufgerufen.
- Das TML-Pendant der jeweiligen XTG-Zeile wird an `$output` angehängt.

wrapPlainTextInTags()

Diese Subroutine übergibt Plain-Text an das Codepageconversion-Modul und verpackt das Ergebnis in alle nötigen öffnenden und schließenden Tags, die an der entsprechenden Textstelle aktiv sind.

Aufruf

- `$plainText`
- Enthält den zu behandelnden Plain-Text.

Seiteneffekte

- Die Variable `$output` wird durch das Codepageconversion-Modul umgewandelt und um die benötigten öffnenden und schließenden Tags erweitert.

getOpenDiv()

Diese Subroutine liefert das öffnende `div`-Tag eines Absatzes mit allen dazugehörigen Attributen.

Aufruf

- `$line`
- Enthält die zu behandelnde Zeile.

Rückgabe

- Enthält das gesuchte Tag.

handleDivContent()

Diese Subroutine übernimmt den Inhalt eines XTG-Absatzes, entfernt die Anteile, die bereits in das öffnende `div`-Tag geflossen sind und übergibt den Rest an das entsprechende Parser-Objekt.

Aufruf

- `$line`
- Enthält die zu behandelnde Zeile.

Seiteneffekte

- Es werden andere Subroutinen aufgerufen, welche den Inhalt von `$output` verändern.

getParaStyle()

Diese Subroutine übernimmt eine XTG-Zeile und liefert nach einer Datenbankabfrage einen Absatzstil zurück.

Aufruf

- `$line`
- Enthält die zu behandelnde Zeile.

Rückgabe

- Zurückgegeben wird der für die Einbettung in ein `div`-Tag vorbereitete Absatzstil.

getStarPAtts()

Diese Subroutine übernimmt eine XTG-Zeile, übergibt sie an das für `*p`-Tags zuständige Parser-Objekt und liefert das Ergebnis zurück.

Aufruf

- `$line`
- Enthält die zu behandelnde Zeile.

Rückgabe

- Enthält die zurückgelieferten Attribute.

openTag()

Diese Subroutine fügt einem übergebenen öffnenden Bezeichner äußere Klammern hinzu.

Aufruf

- `$plain`
- Enthält den zu umklammernden Tag.

Rückgabe

- Es wird der eingeklammerte String zurückgegeben.

closeTag()

Diese Subroutine fügt einem übergebenen schließenden Tag äußere Klammern hinzu.

Aufruf

- `$plain`
- Enthält den zu umklammernden Tag.

Rückgabe

- Es wird der eingeklammerte String zurückgegeben.

handleEntitySpecialCharacter()

Diese Subroutine übernimmt ein Sonderzeichen und liefert das entsprechende Entity zurück.

Aufruf

- `$entitySpecialChar`
- Enthält das übergebene Sonderzeichen.

Rückgabe

- Es wird das entsprechende Entity zurückgegeben.

HTMLPreprocessor.pm

Dieses Modul konvertiert verschiedene HTML-Formate in ein einheitliches Eingabeformat für das HTMLProcessor-Modul. Insbesondere werden `<p>`-Tags in `<div>`-Tags umgewandelt, Attributwerte werden in Anführungszeichen gesetzt, alle Tags werden in Kleinbuchstaben zurückgeliefert und es findet eine Leerzeilenbehandlung statt.

Interne Schnittstellen

process()

Diese Subroutine stellt die Schnittstelle des HTMLPreprocessors-Moduls nach außen dar. Sie wird vom HTMLProcessor aufgerufen, um den Eingabedatenstrom in ein einheitliches Format umzuwandeln.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.

- `$input`
- Enthält die zu transformierenden Daten als String.

Rückgabe

- Enthält die transformierten Daten als String.

start()

Diese Subroutine wird aufgerufen, falls das Modul `HTML::Parser` ein öffnendes Tag gefun-den hat.

Aufruf

- `$t`
- Enthält das gefundene Tag-Element.
- `$attribs`
- Enthält die Attributnamen und Werte des gefundenen Tag-Elementes als Paare in einem Hash.

Seiteneffekte

- Das transformierte Tagelement wird mit den transformierten Attribut-Name/Werte-Paaren an die globale Variable `$output` angehängt.

end()

Diese Subroutine wird aufgerufen, falls das Modul `HTML::Parser` ein schließendes Tag gefunden hat.

Aufruf

- **`$t`**
- Enthält das gefundene Tag-Element.

Seiteneffekte

- Das transformierte Tagelement wird an die globale Variable `$output` angehängt.

text()

Diese Subroutine wird aufgerufen, falls das Modul `HTML::Parser` Plain-Text gefunden hat.

Aufruf

- `$text`

- Enthält den gefundenen Plaintext.

Seiteneffekte

- Die Plain-Textdaten werden ohne Transformation an die globale Variable `$output` angehängt.

p2div()

Diese Subroutine wird von den Routinen `start()` und `end()` aufgerufen, falls ein Tag gefunden wird.

Aufruf

- **`$t`**

- Enthält den gefundenen Tag-Namen.

Rückgabe

- Falls der Tagname `<p>` ist, so enthält der Rückgabewert ein `<div>`; anderenfalls wird der ursprüngliche Tagname wieder zurückgeliefert.

HTMLBuilder.pm

Die Vorgehensweise des HTMLBuilder-Moduls entspricht bis auf die Hilfsroutine `isSingle-Tag()` der oben beschriebenen allgemeinen Vorgehensweise der Builder.

Interne Schnittstellen

isSingleTag()

Diese Subroutine überprüft anhand eines vordefinierten Arrays, ob es sich bei dem übergebenen Start-Tag um einen Single-Tag handelt.

Aufruf

- **`$tag`**

- Enthält das zu prüfenden Tag.

Rückgabe

- 1, falls es sich um einen Single-Tag handelt,

- 0, wenn das geprüfte Tag kein Single-Tag ist.

sortAttributes()

Diese Hilfsfunktion dient dazu, die von der Datenbank zurückgegebenen Attribute in die benötigte Reihenfolge zu bringen.

Aufruf

- `$mixedAttributes`
- Enthält die ungeordnete Rückgabe der Datenbankanfrage.

Rückgabe

- `($tag, @htmlAttributes, @cssAttributes)`
- Array, welches die Attribute in der benötigten Reihenfolge enthält.

HTMLProcessor.pm

Die Hauptfunktion dieses Moduls ist die korrekte Transformation von wohlgeformtem HTML nach TML unter Beachtung der durch die Datenbank definierten Regeln. Das eigentliche Parsen wird mit Hilfe des `HTML::Parser`-Moduls realisiert, hierfür werden spezielle Funktionen definiert, welche bei Events aufgerufen werden und diese anschließend verarbeiten.

Interne Schnittstellen

comment()

Diese Subroutine wird aufgerufen, falls der HTML-Prozessor einen Kommentar im Eingabestring identifiziert hat. Dieser wird dann einfach in das TML-Dokument übernommen.

Aufruf

- **`$comment`**
- Enthält den gefundenen Kommentar.

Seiteneffekte

- Der gefundene Kommentar wird an den Anfang des Arrays `$output` angehängt

startingTag()

Diese Subroutine wird aufgerufen, falls der HTML-Prozessor einen Starttag im HTML-Eingabestring identifiziert hat, die eigentliche Transformation nach TML findet also hier statt. Außerdem kümmert sich die Routine um die Behandlung von in Image-Tags gekapselten unbekannten Code, überprüft den verwendeten Stil und startet von hier aus die Anfrage an die Datenbank.

Aufruf

- `$startingTag`
- Enthält den gefundenen Start-Tag.
- `@attribs`
- Enthält die zum Tag zugehörigen Attribute.

Seiteneffekte

- Der gefundene Kommentar wird an den Anfang des Arrays `$output` angehängt

endingTag()

Diese Subroutine wird aufgerufen, falls der HTML-Prozessor einen End-Tag im HTML-Eingabestring identifiziert hat, die eigentliche Transformation nach TML findet also hier statt. Außerdem wird hier überprüft, ob das vorliegende HTML wohlgeformt, bzw. verifizierbar ist und ob der dazugehörige Start-Tag gekapselt wurde. Außerdem wird von hier aus die Anfrage an die Datenbank gestartet.

Aufruf

- `$endingTag`
- Enthält das gefundene End-Tag.
- `@attribs`
- Enthält das gefundene End-Tag ohne Slash.

Seiteneffekte

- Der gefundene Kommentar wird an den Anfang des Arrays `$output` angehängt

plainText()

Diese Subroutine wird aufgerufen, falls der HTML-Prozessor reinen Text im HTML-Eingabestring vorgefunden hat. Weiterhin wird hier die Codepage des Textes bestimmt und der Text in die Zielcodepage umgewandelt.

Aufruf

- `$plainText`
- Enthält den gefundenen Text.

Seiteneffekte

- Der gefundene Kommentar wird an den Anfang des Arrays `$output` angehängt

isSingleTag()

Diese Subroutine überprüft anhand eines vordefinierten Arrays, ob es sich bei dem übergebenen Start-Tag um einen Single-Tag handelt.

Aufruf

- **`$tag`**
- Enthält das zu prüfende Tag.

Rückgabe

- 1, falls es sich um einen Single-Tag handelt,
- 0, wenn das geprüfte Tag kein Single-Tag ist.

RTFBuilder.pm

Der RTF-Builder hat die Aufgabe, aus vorhandenen TML-Daten RTF-Snippets zu erzeugen. Snippets sind in diesem Fall keine vollständigen RTF-Dokumente, sondern Dokumentenfragmente, die, zusammengefügt mit einem RTF-Template, ein vollständiges Dokument ergeben. Die Subroutinendefinitionen des RTF-Builders sind mit denen der anderen Builder identisch und werden dort beschrieben.

RTFProcessor.pm

Der RTF-Prozessor muss auf die verschiedenen Möglichkeiten der Formatierung von RTF eingehen. Zur Auflösung der einzelnen RTF-Tokens wird das Perl-Modul `RTF::Tokenizer` genutzt, welches das Dokument in die verschiedenen auftretenden Tokens unterteilt. Die von der Transformationsunit aufgerufene `process()`-Subroutine startet den Tokenizer, der seinerseits wiederum die für die jeweilige Token-Art zuständige Subroutine aufruft. Am Ende wird dem entstandenen TML in der `process()`-Routine noch ein Header hinzugefügt.

Eine Gruppe bezeichnet im Folgenden eine Passage, welche durch geschweifte Klammern begrenzt ist und bezieht sich meist auf einen auch logisch abgeschlossenen Teil der zu trans-formierenden Daten. Eine solche Gruppe setzt in Abhängigkeit von den einzelnen gefundenen RTF-Kontrollzeichen und der daraus

resultierenden Datenbankanfrage eine von zwei Variablen. Befindet sich der Tokenizer nicht innerhalb eines deklarierten Absatzes, so wird das gefundene Tag direkt in die Output-Variable geschrieben.

Befindet sich der Tokenizer allerdings innerhalb eines `<div>`-Tags, muss der Output zuerst gesammelt werden, um auf unterschiedliche Reihenfolgen der Tags zwischen RTF und TML reagieren zu können. Zusätzlich muss bei `<div>`- und `<span>`-Tags noch auf die Gültigkeit der Stil-Namen geachtet werden, um diese ggf. mit Hilfe der Stil-Datenbank wieder zurück zu konvertieren.

Interne Schnittstellen

addCollectedToOutput()

Diese Subroutine fügt die gesammelte Ausgabe am Ende eines Absatzes der `$output`-Variablen hinzu.

Seiteneffekte

- `$collectingOutput` wird mit `$output` konkateniert.
- `$divAttributes` wird ausgelesen und zurückgesetzt.
- `$curStyle` wird zurückgesetzt.

handleGroup()

Die Subroutine `handleGroup()` zählt entweder die Anzahl der sich öffnenden Gruppen oder behandelt schließende Gruppen mit Hilfe der Routine `handleClosingGroup()`.

Aufruf

- `$argument`
- Wenn eine Gruppe geöffnet wird, muss hier eine 1 übergeben werden, wenn eine Gruppe geschlossen wird eine 0.

Seiteneffekte

- Die globale Variable `$groupIndex` wird inkrementiert.

handleClosingGroup()

Diese Subroutine dient der Behandlung von schließenden RTF-Gruppen.

Seiteneffekte

- `$tagStackName` (Tag-Stack) wird dekrementiert.
- `$tagStackNumber` (Tag-Stack) wird dekrementiert.

- `$curFontStyle` (Style-Mapping-Stack) wird dekrementiert.
- `$curFontStyleName` wird dekrementiert (Style-Mapping-Stack).
- `$countNotOpenSpans` wird dekrementiert, wenn der behandelte offene Span nicht geschlossen wird.

handleControl()

Die Subroutine `handleControl()` behandelt RTF-Kontrollzeichen, was im Allgemeinen öffnende Tags oder maskierte Sonderzeichen sein können. Diese Sonderzeichen werden manuell abgefangen und ggf. durch die Datenbank-Sonderzeichenbehandlung ersetzt. Falls es sich bei dem Kontrollzeichen um eine öffnendes Tag handelt, wird die Routine `handleOpeningTags()` aufgerufen

Aufruf

- `$argument`
- Enthält das zu behandelnde Kontrollzeichen.
- `$parameter`
- Enthält die Parameter des zu verarbeitenden Kontrollzeichens.

Seiteneffekte

- `$output` und `$plainOutput` werden ggf. erweitert.

handleOpeningTags()

Diese Subroutine wird von `handleControl()` aufgerufen und startet anhand des gefundenen Tags eine Abfrage an die Datenbank. Für Absätze muss vorher noch eine Sonderbehandlung durchgeführt werden, damit die verschiedenen Varianten eines neuen Absatzes erkannt werden können. Auch `<div>`-Attribute müssen gesondert behandelt werden, d.h. alle RTF-Kontrollzeichen, die zu Attributen des `<div>`-Tags werden, werden vorher gesondert in der Variable `$divAttributes` zwischengespeichert, um später korrekt zurückgeschrieben werden zu können.

Aufruf

- `$argument`
- Enthält das zu behandelnde Kontrollzeichen.
- `$parameter`
- Enthält die Parameter des zu verarbeitenden Kontrollzeichens.

- `$ignoreControls`
- Toggle-Wert, welcher dazu dient, - und -Tags von der Behandlung durch die Routine auszuschließen.
-

Seiteneffekte

- `$output` und `$plainOutput` werden ggf. erweitert.
- Tag-Stack wird gegebenenfalls erweitert.
- Toggle-Wert von `collectingOutput` kann geändert werden.

handleFields()

In der Subroutine `handleFields()` werden gefundene RTF-Felder behandelt, welche der Verarbeitung von unbekannten Tags dienen. Nach einer erneuten Überprüfung werden die Inhalte schließlich in TML-Code-Tags überführt.

Seiteneffekte

- `$output, $collectingOutput` und `$plainOutput` werden ggf. erweitert.

handlePlainText()

Diese Subroutine führt die zu vollziehenden Codepage-Konvertierungen für Plain-Text zeichenweise durch.

Seiteneffekte

- `$output, $collectingOutput` und `$plainOutput` werden ggf. erweitert.
- `$errDesc` wird bei Fehler geeignet gesetzt.

checkTmlAttributes()

Diese Subroutine überprüft, ob ein übergebenes Attribut im Array `@tmlAttributes` vorhanden ist. Zweck ist es, alle TML-Attribute geschachtelt in RTF zu speichern, um deren genauen Attribut-Werte wieder auflösen zu können.

Aufruf

- `$attribute`

- Enthält den zu prüfenden Attributnamen.

Rückgabe

- `undefined`, falls das Attribut im Array nicht vorhanden ist, sonst den Wert des Attributes.

CodepageConversion.pm

Dieses Modul dient der Behandlung von Codepageinformationen. Es ermöglicht einerseits die Konvertierung eines Eingabestrings in einen entsprechenden Unicodestring, um ihn so in TML zu speichern. Andererseits können Unicodestrings wieder in 8-bit-Zeichen zurück konvertiert werden.

Interne Schnittstellen

toUnicodeString()

Diese Subroutine konvertiert einen übergebenen String anhand der übergebenen Codepage nach Unicode.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.
- `$in`
- Enthält den zu konvertierenden String.
- `$cpg`
- Enthält einen die Codepage beschreibenden String.
- `$fontOrStylename`
- Gibt den Namen des verwendeten Stiles oder der Font an.
- `$fontOrStyle`
- Gibt an, ob es sich um eine Schriftart oder eine Stildefinition handelt.
- `$schema`
- Gibt das zu verwendende Transformationsschema an.

Rückgabe

- Gibt den konvertierten String zurück.

toCodepageString()

Diese Subroutine konvertiert einen übergebenen String von Unicode in einen String aus 8-bit-Zeichen aus der übergebenen Codepage. Unbekannte Entities werden in RTF zusätzlich gekapselt.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.
- `$in`
- Enthält den zu konvertierenden String.
- `$cpg`
- Enthält einen die Codepage beschreibenden String.
- `$fontOrStyle`
- Gibt an, ob es sich um eine Schriftart oder eine Stildefinition handelt.
- `$schema`
- Gibt das zu verwendende Transformationsschema an.

Rückgabe

- Gibt den konvertierten String zurück.

charToNumberedEntity()

Diese Subroutine konvertiert in einem übergebenen Unicode-String alle Einzelzeichen in Unicode-Numbered-Entities. Dabei überspringt die Subroutine alle anderen Zeichen und bereits vorhandene Entities.

Aufruf

- `$string`
- Enthält den zu konvertierenden String.

Rückgabe

- Gibt den konvertierten String zurück.

convertHexToUnicode()

Diese Subroutine wird von `toUnicodeString()` aufgerufen und übernimmt einen Teil der dort beschriebenen Konvertierung.

Aufruf

- `$self`
- Enthält die obligatorische Selbst-Referenz.
- `$input`
- Enthält den zu konvertierenden String.
- `$cpg`
- Enthält einen die Codepage beschreibenden String.
- `$fontOrStylename`
- Gibt den Namen des verwendeten Stiles oder der Font an.
- `$fontOrStyle`
- Gibt an, ob es sich um eine Schriftart oder eine Stildefinition handelt.
- `$schema`
- Gibt das zu verwendende Transformationsschema an.

Rückgabe

- Gibt den konvertierten String zurück.

Schnittstellen des Data Centric Tier

RuleDBI.pm

Das Modul `RuleDBI.pm` stellt die zentrale Schnittstelle zur MySQL-Datenbank zur Verfügung. Diese umfasst neben den Subroutinen für die Abfrage von Regeln auch die Schnittstelle zu den spezifischen Lookup-Tabellen.

Interne Schnittstellen

getOpenTag()

Diese Subroutine erwartet als Aufrufparameter das eigentliche Tag, dessen Attribute, die Richtung der Transformation, die Nummer des zu verwendenden Schemas, sowie eine SubsetNummer. Wird eine passende Regel in der Datenbank gefunden, gibt die Subroutine diese zurück.

Aufruf

- `$options`
- Eine Referenz auf ein Array, dessen erstes Element die Transformationsrichtung, der zweite Wert die Nummer des zu verwendenden Schemas und der dritte die des zu verwendeten Subsets angibt. Falls kein Subset zur Ausgabe benutzt wird, ist als dritter Wert 1, die Nummer des Standardschemas anzugeben.
- `$tag`
- Enthält das gefundene Tag ohne Klammerung.
- `$attributes`
- Enthält eine Referenz auf ein Array mit den zu übergebenden Attributen. Diese sind im Array in der Reihenfolge Name/Wert/Typ abzulegen.

Rückgabe

- Falls eine Regel gefunden wurde, wird in diesem Array das transformierte Tag, gefolgt von eventuellen Attributen zurückgegeben.

getCloseTag()

Diese Subroutine gibt für einen übergebenen schließenden Tag die Transformationsrichtung und ein zu verwendendes Schema für den schließenden Tag auf der Zielseite der Regel zurück, falls zu dem übergebenen Tag eine Regel existiert.

Aufruf

- `$options`
- Eine Referenz auf ein Array, dessen erstes Element die Transformationsrichtung, der zweite Wert die Nummer des zu verwendenden Schemas und der dritte die des zu verwendeten Subsets angibt.
- `$tag`
- Enthält das gefundene Tag ohne Klammerung.

Rückgabe

- Enthält das gesuchte schließenden Tag.

getCloseFromOpenTag()

Diese Subroutine gibt für ein übergebenes öffnendes Tag das schließende Tag der Gegenseite zurück.

Aufruf

- `$tag`
- Enthält das übergebene öffnende Tag.

Rückgabe

- Enthält das gesuchte schließenden Tag.

getRule()

Gibt zu einer angegebenen Regelnummer die gesuchte Zielseite dieser Regel zurück. Falls in der Datenbank Durchschleif-Attribute definiert sind, werden diese mit den jeweiligen loopthrough-Werten ersetzt. Von `getRuleID()` nicht markierte optionale Attribute werden nicht mit zurückgegeben.

Aufruf

- `$ruleID`
- Erwartet die Nummer der gesuchten Regel.

Rückgabe

- Enthält das gesuchte Tag, gegebenenfalls gefolgt von den Attributen des Elements.

getRuleID()

Diese Subroutine liefert in Abhängigkeit von den per **getOpenTag()** übergebenen Werten die Nummer der Regel in der Datenbank zurück, falls eine entsprechende Regel existiert.

Rückgabe

- Enthält die Nummer der gesuchten Regel.

Seiteneffekte

- `@foundLinks` wird mit gefundenen Link-IDs befüllt.
- `%foundLT` wird mit einem Schlüssel/Wert-Paar für jeden gefundenen Loopthrough befüllt.

connectDatabase()

Diese Subroutine stellt die Verbindung mit der MySQL-Datenbank her. Hierzu benutzt die Routine die in `@DATABASE_CONNECTION_PARAMETERS` hinterlegten Verbindungsparameter. Alternativ kann beim Aufruf ein Handle übergeben werden.

Aufruf

- `$dbh`
- Falls dieser Wert übergeben wird, wird das bestehende Datenbankhandle damit überschrieben.

Rückgabe

- Falls die Verbindung hergestellt werden kann, wird 1 zurückgeliefert.

Seiteneffekte

- `$databaseHandle`
- Falls die Verbindung hergestellt werden kann, wird das `$databaseHandle` neu gesetzt.
- `$remoteHandle`
- Falls der Subroutine das Handle von außen übergeben wurde, wird diese Variable auf 1 gesetzt.

disconnectDatabase()

Diese Subroutine beendet die aktuelle Verbindung mit der Datenbank, indem sie die `disconnect()`-Subroutine des aktiven Datenbankhandles aufruft.

Seiteneffekte

- $databaseHandle
- Wird bei Erfolg freigegeben.

Rückgabe

- Bei Erfolg wird eine 0, ansonsten eine 1 zurückgegeben.

setConnectionParameters()

Die Subroutine `setConnectionParameters()` dient dazu, die Standardeinstellungen zum Verbindungsaufbau neu anzugeben.

Aufruf

- `$username`
- Enthält den zu setzenden Benutzernamen.
- `$password`
- Enthält das zu setzende Passwort.
- `$hostname`
- Enthält den zu verwendenden Hostnamen.
- `$databasename`
- Enthält den Namen der zu verwendenden Datenbank.

Seiteneffekte

- `@DATABASE_CONNECTION_PARAMETERS` wird neu gesetzt.

Rückgabe

- Bei Erfolg wird eine 0, ansonsten eine 1 zurückgegeben.

executeSQLCommand()

Diese Subroutine dient der Ausführung eines einzelnen SQL-Kommandos. Alle Resultate dieser Abfrage werden von der Routine serialisiert zurückgeliefert.

Aufruf

- `$sqlCommand`
- Enthält das auszuführende SQL-Kommando als String.

Rückgabe

- Jeder Eintrag im zurückgelieferten Array entspricht einer Zeile aus dem Result-Set der Abfrage, mehrere Reihen von Resultaten werden der Reihenfolge nach an das Array angehängt.

getEntityFromSpecialChar()

Diese Subroutine liefert für ein gegebenes Entity das entsprechende Sonderzeichen in Abhängigkeit von der verwendeten Font oder dem verwendeten Style zurück. Wird kein passender Eintrag gefunden, wird geprüft, ob die Transformation als Default-Regel definiert ist.

Aufruf

- `$specialChar`
- Enthält das übergebene Sonderzeichen.
- `$ruleset`
- Enthält die Nummer des zu verwendenden Regelschemas.

Rückgabe

- `$result[0]`
- Enthält die gesuchte Entity als String.
- `$result[1]`
- Enthält eine 1, falls ein default-Eintrag zurückgeliefert worden ist, sonst wird eine 0 zurückgegeben.

getSpecialCharFromEntity()

Diese Subroutine liefert für ein gegebenes Sonderzeichen das entsprechende Entity in Abhängigkeit vom verwendeten Font oder dem verwendeten Stil zurück. Wird kein passender Eintrag gefunden, wird geprüft, ob die Transformation als Default-Regel definiert ist.

Aufruf

- `$entity`
- Enthält das übergebene Entity.
- `$ruleset`
- Enthält die Nummer des zu verwendenden Regelschemas.

Rückgabe

- `$result[0]`
- Enthält das gesuchte Sonderzeichen.
- `$result[1]`
- Enthält eine 1, falls ein default-Eintrag zurückgeliefert worden ist, sonst wird eine 0 zurückgegeben.

getCodepage()

Diese Subroutine liefert in Abhängigkeit von einer übergebenen Schriftart und Betriebssystemkennung die dazu eingetragene Codepage aus der dafür vorgesehenen Lookup-Tabelle zurück.

Aufruf

- `$font`
- Enthält den Namen einer Schriftart.
- `$operatingSystem`
- Enthält die die Bezeichnung eines Betriebssystems.

Rückgabe

- Enthält die Beschreibung der gesuchten Codepage.

getTargetStyle()

Diese Subroutine konvertiert einen bestimmten Stil-Namen aus dem Quellformat in den entsprechenden Stil-Namen des Zielformates.

Aufruf

- `$schema`
- Enthält das Schema in dem nach dem Stylenamen gesucht wird.
- `$richtung`
- Enthält die Transformationsrichtung.
- 0 steht für die Transformation nach TML
- 1 steht für die Transformation aus TML heraus
- `$styleName`
- Enthält den Style-Namen aus dem Quellformat als String.

Rückgabe

- Enthält den Stil-Namen des Zielformates als String.

getRulesetStyles()

Diese Subroutine liefert eine Referenz auf ein Hash zurück, welcher sämtliche in der Datenbank hinterlegten Informationen aller Stile eines Schemas enthält.

Aufruf

- $schema
- Enthält das Schema in dem nach den Styles gesucht wird.

Rückgabe

- Enthält eine Hash-Referenz, welche auf einen Hash verweist, welcher die Namen, Beschreibungen und Definitionen der Stile des übergebenen Schemas enthält.

getFontID()

Diese Subroutine liefert zu einer übergebenen RTF-Font-ID den passenden Namen zurück, falls dieser in der Datenbank hinterlegt ist.

Aufruf

- `$fontID`
- Enthält die ID der gesuchten Schriftart.

Rückgabe

- Enthält den Namen der gesuchten Schriftart.

getFontName()

Diese Subroutine liefert zu dem übergebenen Namen eines in einem RTF-Dokument benutzten Font die passende ID zurück, falls diese in der Datenbank hinterlegt ist.

Aufruf

- `$fontName`
- Enthält den Bezeichner der gesuchten Schriftart.

Rückgabe

- Enthält den gesuchten Identifier.

flushCache()

Diese Subroutine dient der teilweisen Entleerung des Hashes, welcher die zwischengespeicherten SQL-Anfragen enthält.

Aufruf

- `$hashRef`
- Enthält die Referenz auf den zu entleerenden Hash.

Seiteneffekte

- Der per Referenz übergebene Zwischenspeicher wird über den Aufruf von `flushByHits()` und `flushByTimestamp()` teilweise geleert. Wieviel Prozent des Speichers erhalten bleiben, wird mit Hilfe der Konstante `$cacheRescueFactor` angegeben.

flushByHits()

Diese Subroutine wird von `flushCache()` aufgerufen und ist für das Löschen der Einträge bezüglich des Kriteriums der Anfragetrefferanzahl zuständig.

Aufruf

- `$nrToDelete`
- Enthält die Anzahl der zu löschenden Einträge.
- `$hashRef`
- Enthält die Referenz auf den zu entleerenden Hash.

Seiteneffekte

- Der per Referenz übergebenen Zwischenspeicher wird um die angegeben Anzahl von Einträgen verkleinert, wobei die Einträge mit der größten Trefferanzahl erhalten bleiben.

flushByTimestamp()

Diese Subroutine wird von `flushCache()` aufgerufen und ist für das Löschen der Einträge bezüglich des Kriteriums der Aktualität zuständig.

Aufruf

- `$nrToDelete`
- Enthält die Anzahl der zu löschenden Einträge.
- `$hashRef`
- Enthält die Referenz auf den zu entleerenden Hash.

Seiteneffekte

- Der per Referenz übergebene Zwischenspeicher wird um die angegebene Anzahl von Einträgen verkleinert, wobei die Einträge mit dem neuesten Zeitstempel erhalten bleiben.

printCache()

Diese Subroutine gibt den Inhalt des Zwischenspeichers formatiert auf die Standardausgabe aus.

Aufruf

- `$hashRef`
- Enthält die Referenz auf den zu entleerenden Hash.

getHitFactor()

Diese Subroutine gibt das Verhältnis der Zwischenspeicher-Anfragen zu den dabei erzielten Treffern zurück, indem es die entsprechenden Skalare ausliest.

Rückgabe

- Enthält das Verhältnis von Anfragen zu Treffern.

mask()

Diese Subroutine dient der Maskierung und Demaskierung von Schlüsselwörtern und Backslashes in Eingabefeldern, welche per SQL-Kommando an die Datenbank geschickt werden.

Aufruf

- `$direction`
- 0, falls das übergebene Array maskiert werden soll.
- 1, falls das übergebene Array demaskiert werden soll.
- `@incoming`
- Enthält die zu behandelnden Zeichenketten.

Rückgabe

- Enthält die maskierten/demaskierten Zeichenketten in einem Array.

Externe Schnittstellen

cacheThresholdReached()

Diese Subroutine wird von außen aufgerufen und teilt dem aufrufenden Prozess mit, ob der Schwellwert für die Zwischenspeicherentleerung erreicht worden ist.

Rückgabe

- 0, falls der Schwellwert noch nicht erreicht ist.
- 1, falls der Schwellwert erreicht oder überschritten worden ist.

remoteCacheFlush()

Diese Subroutine wird von außen aufgerufen und führt die Subroutine `flushCache()` aus.

Seiteneffekte

- Der Zwischenspeicher wird aufgrund des Aufrufs von `flushCache()` teilweise entleert.

remoteCacheReset()

Diese Subroutine wird von außen aufgerufen und entleert den Zwischenspeicher vollständig, indem der enthaltene Hash komplett geleert wird.

Seiteneffekte

- Der Zwischenspeicher wird vollständig geleert.

LITERATURVERZEICHNIS

Online-Quellen

[Aas08] **Aas, Gisle; Chase, Michael A.** HTML::Parser. *CPAN*. [Online] 2008. [Cited: 12 23, 2008.] http://search.cpan.org/~gaas/HTML-Parser-3.59/Parser.pm.

[Ado08] **Adobe.** Extensible Metadata Platform (XMP). *Adobe XMP*. [Online] 2008. [Cited: 12 4, 2008.] http://www.adobe.com/products/xmp/.

[Ant07] **Antenna House.** Professional Formatting Solutions. *XML and XSL-FO to PDF with High Quality SVG, MathML and Multilingual Support for over 50 Languages.* [Online] 09 11, 2007. [Cited: 09 11, 2007.] http://www.antennahouse.com/.

[Apa07] **Apache Software Foundation.** Apache FOP. *The Apache XML Graphics Project*. [Online] 11 18, 2007. [Cited: 11 21, 2007.] http://xmlgraphics.apache.org/fop/index.html.

[Arg06] **Argast, Thomas.** Ein Online-Tutorial. *Elektronisch publizieren im PDF-Format*. [Online] Universitätsbibliothek Freiburg, 16. 08 2006. [Zitat vom: 29. 11 2007.] http://www.freidok.uni-freiburg.de/doku/tutorial/.

[Bec04] **Becker, Klaus.** Theorie formaler Sprachen. *Informatik Bildungsserver Rheinland-Pfalz*. [Online] 2004. http://informatik.bildung-rp.de/fileadmin/user_upload/informatik.bildung-rp.de/Weiterbildung/pps.

[Bei07] **Beinert, Wolfgang.** *Das Lexikon der westeuropäischen Typographie.* [Online] [Zitat vom: 13. 08 2007.] http://www.typolexikon.de/.

[Bos96] **Bosak, Jon; Sun Microsystems.** DSSSL Online Application Profile. *ibiblio*. [Online] University of North Carolina at Chapel Hill, 08 16, 1996. [Cited: 12 23, 2008.] http://www.ibiblio.org/pub/sun-info/standards/dsssl/dssslo/do960816.htm.

[Cla98] **Clark, James.** ISO/IEC 10179:1996. *DSSSL*. [Online] 9 11, 1998. [Cited: 5 12, 2006.] http://www.jclark.com/dsssl/.

[Die03] **Diekert, Volker.** Dyck-Sprachen und Syntax-Analyse. *Teilbibliothek Mathematik/Informatik*. [Online] 25. 6 2003. [Zitat vom: 24. 11 2008.] http://wwwbib.informatik.tu-muenchen.de/Stroehlein/Berichte/dyck/dieckert.pdf.

[Eng03] **English, Joe.** COST. *COST Home Page*. [Online] 08 02, 2003. [Cited: 04 26, 2007.] http://www.flightlab.com/cost/.

[Fei07] **Feig & Partner.** Marktübersicht. *Contentmanager.de*. [Online] [Zitat vom: 09. 11 2007.] http://www.contentmanager.de/itguide/marktuebersicht.html.

[Jam07] **Jamra, Mark; Peloquin, Jamie.** TypeCulture. *A Digital Type Foundry and Academic Resource*. [Online] [Cited: 11 28, 2007.] http://www.typeculture.com.

[Mal04] **Malloy Incorporated.** *Malloy Quartely*. [Online] 2004. [Cited: 10 24, 2007.] http://www.malloy.com/pdf/quarterly/1402-spring04.pdf.

[Mic09] **Microsoft.** XML Paper Specifications. [Online] 2009. [Zitat vom: 6. 1 2009.] http://www.microsoft.com/germany/siteservices/xps/default.mspx.

[Nit07] **Nitro PDF Inc.** The PDF User Community. *Planet PDF*. [Online] [Cited: 11 26, 2007.] http://www.planetpdf.com.

[Obj07] **Object Management Group.** Unified Modeling Language (UML), Version 2.1.1. *OMG*. [Online] 02 05, 2007. [Cited: 11 08, 2007.] http://www.omg.org/cgi-bin/apps/doc?formal/07-02-05.pdf.

[PDF08] **PDF/A Competence Center.** PDF/A FAQ. *Das PDF/A Competence Center*. [Online] 2008. [Zitat vom: 2. 12 2008.] http://www.pdfa.org/doku.php?id=pdfa:faq.

[Pre97] **Prescod, Paul.** Introduction to DSSSL. [Online] 07 24, 1997. [Cited: 05 22, 2006.] http://www.prescod.net/dsssl/.

[Typ07] **Punchcut.** Typographic Collaboration. *Typophile*. [Online] Punchcut. [Cited: 11 28, 2007.] http://typophile.com/.

[Sch08] **Schenk, Jörg;.** PDF-X, der PrePress-Standard. *PDF-X.de*. [Online] 2008. [Zitat vom: 2. 12 2008.] http://www.pdf-x.de/GRUNDLAGEN_FAQ_s/home.html.

[Ser08] **Sergeant, Matt; Cooper, Clark; Wall, Larry.** XML::Parser. *CPAN*. [Online] 2008. [Cited: 12 23, 2008.] http://search.cpan.org/~msergeant/XML-Parser/Parser.pm.

[The05] **The Unicode Consortium.** Unicode Homepage. [Online] 2005. http://www.unicode.org/.

[Vir07] **Virk, Rizwan.** The Seven Deadly Sins of XML Publishing. *xDox Converter Desktop*. [Online] CambridgeDocs. [Cited: 11 02, 2007.] http://www.cambridgedocuments.com/resources/whitepapers/sevendeadlysins.htm.

[W3C00] **W3C.** Extensible Markup Language (XML) 1.0 (Second Edition). *W3C Recommendation*. [Online] October 6, 2000. [Cited: 11 10, 2007.] http://www.w3.org/TR/2000/REC-xml-20001006.

[W3C06] —. Extensible Stylesheet Language (XSL) Version 1.1. *W3C Recommendation*. [Online] 12 5, 2006. [Cited: 11 10, 2007.] http://www.w3.org/TR/xsl/.

[Wik04] **Wikipedia.** XSLT. *Wikipedia*. [Online] [Zitat vom: 9. September 2004.] http://de.wikipedia.org/wiki/XSLT.

Print-Quellen

[Aho88] **Aho, Alfred V., Sethi, Ravi and Ullman, Jeffrey D.** *Compilers: Principles, Techniques, Tools.* Reading : Addison-Wesley, 1988. ISBN 0-2011-0088-6.

[Aho73] **Aho, Alfred V. and Ullman, Jeffrey D.** *The Theory of Parsing, Translation and Compiling (Band 1: Parsing).* s.l. : Prentice Hall, 1973. ISBN 0-1391-4564-8.

[Bae87] **Baeseler, Frank und Heck, Bärbel.** *Desktop Publishing.* Hamburg : McGraw-Hill Book Company, 1987. ISBN 3-89028-090-0.

[Bau90] **Baumann, Hans D. und Klein, Manfred.** *Desktop Publishing: Typografie, Layout.* Niedernhausen : Falken-Verlag, 1990. ISBN 3-8068-4330-9.

[Bec02] **Becker, M. und Bramann, Klaus-W. (Hrsg.).** Cross Media Publishing. *Verlagslexikon.* Input-Verlag, 2002, Bd. 1. Auflage.

[Beh00] **Behme, H und Mintert, S.** *XML in der Praxis.* München [u.a.] : Addison Wesley, 2000. ISBN 3-8273-1636-7.

[Box01] **Box, Don, Skonnard, Aaron und Lam, John.** *Essential XML.* München : Addison-Wesley, 2001. ISBN 3-8273-1769-X.

[Bur03] **Burke, Sean M.** *RTF Pocket Guide.* Sebastopol : O'Reilly Media, 2003. ISBN 0-596-00475-3.

[Chr03] **Christ, Oliver.** *Content-Management in der Praxis.* Berlin [u.a.] : Springer-Verlag, 2003. ISBN 3-5400-0103-4.

[Duß73] **Dußler, Sepp und Kolling, Fritz.** *Moderne Setzerei.* Pullach : Verlag Dokumentation Saur KG, 1973. ISBN 3-7940-8703-8.

[Ehl03] **Ehlert, Lars H.** *Content Management Anwendungen.* Münster : Westfälische Wilhelms-Universität Münster, 2003. Dissertation.

[Fri01] **Fritsche, Hans Peter.** *Cross Media Publishing.* Bonn : Galileo Press, 2001. ISBN 978-3934358461.

[Gei02] **Geiger, Markus.** *Internetstrategien für Printmedienunternehmungen.* Lohmar [u.a.] : Josef Eul Verlag, 2002. ISBN 3-8993-6010-9.

[Gen78] **Genzmer, Fritz und Kuchel, Jürgen H.** *Umgang mit der schwarzen Kunst: vom Ms. zum fertigen Druckerzeugnis.* Berlin : Schiele & Schön, 1978. ISBN 3-7949-0251-3.

[Gor06] **Gorke, Bastian.** *XML-Datenbanken in der Praxis.* Saarbrücken : bomots-Verlag, 2006. ISBN 3-9393-1619-9.

[Har04] **Harold, Elliotte Rusty.** *XML.* Bonn : Mitp-Verlag, 2004. ISBN 3-8266-0915-8.

[Har041] **Harold, Elliotte Rusty, Means, W. Scott and St. Laurent, Simon.** *XML in a Nutshell.* Sebastopol : O'Reilly, 2004. ISBN 0-596-00764-7.

[Hei01] **Heinrich, Jürgen.** *Medienökonomie.* Wiesbaden : Westdeutscher Verlag, 2001. ISBN 3-5313-2636-8.

[Jac03] **Jackenkroll, Melanie.** *Cross Media Publishing mittels XML - Die Enzyklopädie als Beispiel.* Köln : Fachhochsch., Fak. für Informations- und Kommunikationswiss., Inst. für Informationswiss., 2003. Kölner Arbeitspapiere zur Bibliotheks- und Informationswissenschaft, Band 35.

[Kaz02] **Kazakos, Wassilios, Schmidt, Andreas und Tomczyk, Peter.** *Datenbanken und XML.* Berlin [u.a.] : Springer Verlag (Xpert.press), 2002. ISBN 3-540-41956-X.

[Kel02] **Kelter, Udo.** *Datenbanksysteme I.* Siegen : Universität Siegen, 2002.

[Kel03] —. *Transaktionen.* Siegen : Universität Siegen, 2003.

[Keu03] **Keuper, Frank und Hans, René.** *Multimedia-Management.* Wiesbaden : Gabler, 2003. ISBN 3-409-11926-4.

[Koc04] **Koch, Sven und Nieland, Stefan.** *Content- und Cross-Media-Management in Verlagsunternehmen.* Aachen : Shaker-Verlag, 2004. ISBN 3-8322-2481-5.

[Kre88] **Kredel, Lutz.** *Grundlagen des Desktop Publishing.* München : Droemersche Verlagsanstalt Th. Knauf Nachf., 1988. ISBN 3-426-26353-X.

[Kre04] **Kretschmar, Oliver und Dreyer, Roland.** *Medien-Datenbank- und Medien-Logistik-Systeme.* München : Oldenbourg Wissenschaftsverlag, 2004. ISBN 3-486-27494-5.

[Mer02] **Merzenich, Wolfgang Prof. Dr.** *Algorithmen.* Universität Siegen : Scriptum zu Vorlesung, 2002.

[Mer97] —. *Programmiersprachen und Compilerbau.* Universität Siegen : Scriptum zur Vorlesung, 1997.

[Mey06] **Meyer, Oliver.** *aTool - Typographie als Quelle der Textstruktur.* Aachen : Shaker Verlag, 2006. ISBN 3-8322-5011-5.

[Min03] **Mintert, Stefan (Hrsg.).** *XHTML, CSS & Co.* München [u.a.] : Addison Wesley Deutschland, 2003. ISBN 3-8273-1872-6.

[Min02] —. *XML & Co.* München [u.a.] : Addison Wesley Deutschland, 2002. ISBN 3-8273-1844-0.

[Mül02] **Müller-Kalthoff, Björn.** *Cross-Media Management.* Berlin : Springer, 2002. ISBN 3-540-43692-8.

[Pat02] **Patay, Helmut.** *Webprogrammierung mit Perl.* München : Addison-Wesley Verlag, 2002. ISBN 3-8273-2053-4.

[Paw02] **Pawson, Dave.** *XSL-FO.* Sebastopol : O'Reilly, 2002. ISBN 0-596-00355-2.

[Pet98] **Peters, Joachim.** *Bestiarium der Bits 'n' Bytes.* s.l. : Springer Verlag, Berlin und Macup Verlag, Hamburg, 1998. ISBN 3-540-63420-7.

[Rot03] **Rothfuss, Gunther (Hrsg.) und Eisenbiegler, Jörn.** *Content Management mit XML.* Berlin [u.a.] : Springer Verlag, 2003. ISBN 3-540-43844-0.

[Rub88] **Rubinstein, Richard.** *Digital Typography - An Introduction to Type and Composition for Computer System Design.* s.l. : Addison-Wesley Publishing Company, 1988. ISBN 0-201-17633-5.

[Sch07] **Scheifele, Nicola.** Coole Glanzeffekte fallen immer auf. *versio!* Druck-Innovationen, Corporate Publishing, Printbuying, 2007, 04.

[Sch06] **Schumann, Matthias und Hess, Thomas.** *Grundfragen der Medienwirtschaft.* Berlin : Springer-Verlag, 2000, 2002, 2006. ISBN 3-540-29228-4.

[Sju05] **Sjurts, Insa.** *Strategien in der Medienbranche.* Wiesbaden : Gabler, 2005. ISBN 3-4093-2181-0.

[von93] **von Harlessem, Marcus.** *Datenbank.* München : Vogel Verlag, 1993. ISBN 3-8023-1189-2.

[von94] —. *Enterprise Client/Server-Computing.* Würzburg : Vogel Verlag, 1994. ISBN 3-8259-1304-X.

[Wen08] **Wendlandt, Matthias.** *Formalsprachliche Aspekte von XML.* s.l. : Deutsche National Bibliothek, 2008. Dissertation.

[Zsc02] **Zschau, Oliver, Traub, D. und Zahradka, R.** *Web Content Management.* Bonn : Galileo Press, 2002, 2. Auflage. ISBN 3-8984-2157-0.