Ruprecht-Karls-Universität
Neuphilologische Fakultät
Institut für Computerlinguistik

Bachelorarbeit

23.03.2017

Automatisierte Layoutformatierung für Printmedien mit CSS

„CSS paged media module level 3“ in der Anwendung im Vgl. zu XSL-FO

Erstbetreuer: Prof. Dr. Andreas Witt
Zweitbetreuer: Manuel Montero Pineda
Zweitgutachterin: Prof. Dr. Katja Markert

Svenja Lohse (7. Semester)
Römerstr. 131
69126 Heidelberg
Tel: 0162 8620613
Email: S.Lohse@stud.uni-heidelberg.de

HF: Computerlinguistik (75%)
NF: Bildungswissenschaft (25%)

Abstract

Books and other printed media are still a favorite way to save and exchange knowledge and thoughts. These contents have to get enriched with meta data to be logically, usefully and clearly stored and representated. Through syntactical and semantical connections linguistic meta data enables specific analysis, extraction and processing amounts of data. Also the layout of texts, informations etc. works with these meta data and is important for understanding the content. Cross media publishing is interested in designing layouts for different output media but with the same or few programming languages.

This thesis explores empirically the work ability of CSS level 3 (CSS3), especially CSS paged media module level 3, for creation of print layouts in practice in contrast to XSL-FO. How is it working and what is CSS3 missing to get a more professional result?
Data base is MARC-XML from the „Deutsche Nationalbibliothek" datashop, which is processed by XSLT to a HTML5 document. On this CSS3 properties, grouped by typographical concepts, are tried out rendered by Antenna House Formatter.
The subject combination of computer science (XML technologies), publishing (layout, cross media publishing) and linguistics (enrichment of content with markup languages) makes further research and developement of automatical CSS layout processing for printed media interesting.

The study comes to the conclusion that CSS3 has a lot of things to improve before managing professional print layouts like XSL-FO does. But simple documents and simple, mean formatting requests are executed easily and fastly. The person developing and controlling CSS stylesheets doesn't have to be a professional layout designer to get decent output. More browser compatibility would be helpful so more user could test scenarios (like „digital first" process, from website to print) and give feedback. This could be used for evaluation and advancement of CSS3.
In conclusion, CSS3 is improvement-needy but also upgradeable and suitable for future of professional print layout creation.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AH(F)	Antenna House (Formatter)
CALS	Computer-Aided Acquisition and Life-Cycle Support
CMDI	Component Metadata Initiative
CSS(3)	Cascading Style Sheets (Module Level 3)
CSS3BG	CSS Backgrounds and Borders Module Level 3
CSS3BREAK	CSS Fragmentation Module Level 3
CSS3COL	CSS Multi-column Layout Module
CSS3FONTS	CSS Font Module Level 3
CSS3GCPM	CSS Generated Content for Paged Media Module
CSS3IMAGE	CSS Image Values and Replaced Content Module Level 3
CSS3LIST	CSS Lists and Counters Module Level 3
CSS3PAGE	Paged Media Module Level 3
CSS3SELECT	Selectors Level 3
CSS3SPEECH	CSS Speech Module
CSS3TABLE	CSS Table Module Level 3
CSS3TDECO	CSS Text Decoration Module Level 3
CSS3TEXT	CSS Text Module Level 3
CSS4COLOR	CSS Color Module Level 3
DC(MES)	Dublin Core (Metadata Element Set)
DNB	Deutsche Nationalbibliothek
FOP	Apache Formatting Objects Processor
IMDI	ISLE Metadata Initiative
ISO	International Organization for Standardization
IVZ	Inhaltsverzeichnis

MARC	Machine-Readable-Cataloging
PDF	Portable Document Format
PI	Processing Intstruction
PS	Photoshop
SGML	Standard Generalized Markup Language
SVG	Scalable Vector Graphics
TEI	Text Encoding Initiative
W3C	World Wide Web Consortium
(X)HTML(5)	(Extensible) Hypertext Markup Language (Version 5)
XML	Extensible Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language
(XSL-)FO	(XSL-)Formatting Objects
XSLT	XSL Transformation

Typografische Konventionen

`Nichtproportionalschrift`

Codebegriffe oder -ausschnitte innerhalb des Fließtexts

Hinterlegung

Zusammenhängender Code, meist abgegrenzt vom Fließtext

Farbkennzeichnung orange bzw. blau

Hervorhebung zu beachtender XSL(-FO) bzw. CSS-Codeformulierungen

[1]

Verweis auf Codezeile im Fließtext

[1] bzw. [1]

Verweisziel in XSL-FO- bzw CSS-Code

1 Einleitung

„Schrift ist – nach der verbalen Sprache – nach wie vor das bedeutendste Kommunikationsinstrument.“ (Günter Gerhard Lange, 1978)

„Die ganze Kunst der Sprache besteht darin, verstanden zu werden.“ (Konfuzius, 551-479 v.Chr.)

Die niedergeschriebene Sprache ist ein mächtiges Hilfsmittel der Kommunikation, wie Herr Lange, ein deutscher Typograf und Lehrer, anmerkte. Damit kann Wissen weitergegeben werden, es können Gefühlswelten und Verhalten beschrieben, ausgetauscht und reflektierend betrachtet werden. In der Schule als Biologiebuch, am Arbeitsplatz als Projektbericht, in der Freizeit in Form von Romanen oder Nachrichten, überall sind Schriften im Alltag zu finden.
Doch Konfuzius‘ Ausspruch umgekehrend deutend kann man feststellen, dass die geschriebenen Worte nichts nutzen, wenn man sie nicht lesen oder verstehen kann. So kann der durchschnittliche Deutsche eher weniger mit chinesischen Schriftzeichen anfangen, aber auch wenn er die Italienische Schrift lesen kann, würde er die Bedeutung der Sätze nicht verstehen (sollte er die Sprache nicht gelernt haben). Und auch in der eigenen Sprache kann es zu Miss- oder Unverständnissen kommen, wenn z.B. der „klischeehafte“ Arzt eine unleserliche Handschrift besitzt oder der Sitznachbar in der Schule zu klein schreibt.
Daher ist es wichtig, dass die Schriftdokumente auch leserlich und ansprechend aussehen, sodass die Informationen darin genau so herübergebracht werden können, wie sie gemeint sind.

1.1 Ausgangssituation und Herausforderung

Ob Unterhaltung, Lernhilfe oder Informationsquelle – Bücher sind selbst in unseren digitalisierten Zeiten immer noch ein beliebtes Medium, um Gedanken und Informationen zu teilen.
Inhalte müssen mit zusätzlichen Daten angereichert werden, um sie logisch, sinnvoll und übersichtlich abspeichern und darstellen zu können. Diese sprachlichen Metadaten ermöglichen es, syntaktische und auch semantische Zusammenhänge herstellen zu können. Dadurch wird die gezielte Analyse, Extraktion und Verarbeitung der Datenmengen bewerkstelligt.

Auch das Aussehen von Texten, Informationen etc. baut auf diesen strukturellen Metadaten auf und ist ausschlaggebend für das Verständnis dieser. Sprachliche Informationen können aufgegriffen und durch verschiedene Programme umgesetzt werden.
So erhält nicht nur ein gedrucktes Buch, sondern auch die Digital- und Onlineversion neben einer logischen Struktur auch ein übersichtliches und ansprechendes Layout, das die Lesbarkeit und die Attraktivität für den Leser steigert.

Das Cross-Media-Publishing beschäftigt sich mit der Überführung von textlichen Objekten zu Webseiten, E-Books und Druckausgaben. Für „effiziente Workflows und schnelle und

qualitätsgesicherte Mehrfachverwertungen“[1] werden medienneutral aufbereitete Daten benötigt, die–am besten auf gleicher Formatbasis, z.B. XML oder (X)HTML, vorliegen. Medienneutral bedeutet hier, dass eine Trennung von Inhalt, Struktur und Gestalt vorliegt und man so flexibel und unabhängig Inhalte gleich gut für bestimmte Ausgabeformen verarbeiten kann.[2]
Auf den vorliegenden Daten wird eine Transformationssprache angewandt, um effizient das passende Outputformat zu kreieren.
Webseiten und viele E-Book-Formate arbeiten bereits mit HTML und formatieren mit CSS. Printprodukte (z.B. Print-PDF) werden bisher allerdings mit Hilfe von Satzsystemen (wie z.B. InDesign) oder XSL-FO erstellt.

XSL-FO ist ein W3C-Standard für die Formatierung von Daten für den Printbereich. Mit XSLT (*Extensible Stylesheet Language Transformation*) kann man z.B. (XML-)Daten einer Datenbank direkt zu Text-, HTML- oder anderen XML-Dateien aufbereiten. Der Standard FO (*Formatting Objects*) erlaubt die vollautomatische Erstellung von Druckdaten wie von Online-Dokumenten bis hin zu gedruckten Handbüchern mit komplexer Seitengestaltung, um sie dann automatisiert auszugeben, z.B. als PDF. XSL-FO ist verbreitet in der Publishingbranche und gilt dank großem Funktionsumfang als mächtiges Werkzeug, aber sie ist ebenfalls komplex und dadurch kompliziert zu überblicken. Außerdem wurde die Weiterentwicklung dieser XML-Technologie nach Version 1.1 eingestellt, die W3C-Arbeitsgruppe dazu wurde geschlossen.

CSS paged media soll mit seiner einfachen Syntax und Sprache eine Alternative für Printmedienformatierung bieten (vor allem für Crossmedia Publishing). *Cascading Stylesheets* (CSS), ursprünglich als eine Erweiterungssprache von (X)HTML zur Trennung von Struktur und Inhalt entwickelt, erlaubt das externe Anlegen von Formatvorlagen, die referenzierend das Layout bestimmen. Die Idee hierbei ist, dass Daten und Formatierung getrennt sind, sodass eine Änderung der Formatierung sich dokumentübergreifend und zentral auf die Darstellung der Daten auswirkt. *CSS Paged Media Module Level 3* (CSS3PAGE) wurde speziell für Druckausgaben entwickelt und ist Teil der Modulvielfalt der Version *CSS Level 3* (CSS3). Bisher wird diese Version allerdings nur von kostenpflichtigen Formatierern, wie im Fall dieser Arbeit von *Antenna House Formatter*, unterstützt und das auch noch nicht gänzlich. Außerdem ist die Benutzung von CSS3 für Printzwecke noch nicht weit verbreitet bzw. nicht umfassend in der Anwendung getestet.

[1] Götz/Ott, S.15
[2] O.V. (o.J.): „Ohne sie geht nichts: Medienneutrale Daten“

1.2 Ziel und Vorgehen

Die vorliegende Arbeit soll CSS paged media und weitere CSS3-Module für die Printmedienformatierung in der Anwendung erforschen, d.h. Kombinationen von Eigenschaften und Funktionen unter realistischen Vorgaben erproben.
Wie wird praktisch mit CSS ein Printmedium formatiert? Was fehlt der Sprache noch zur verbesserten Anwendungsmöglichkeit?
Unter Beachtung dieser Fragen sollen mögliche typografische Konzepte umgesetzt, die momentane praktische Anwendbarkeit bewertet und dadurch Ausbaumöglichkeiten herausgefunden werden.

Anhand von MARC-XML-Daten des „Deutsche Nationalbibliothek"-Archivs soll empirisch getestet werden, wie gut die Verarbeitung der Metadaten zu einem anschaulichen Layout gelingt. Zuerst werden die MARC-XML-Daten verschiedener Zeitschriftentiteldaten mit XSLT zu HTML5 umgewandelt, um von dort aus dann verschiedene Layoutschritte bzw. Eigenschaften von CSS anwenden zu können. Viele typografische Mittel entstehen durch Kombinationen von CSS-Eigenschaften, welche erprobt werden sollen.

Letztendlich werden die Konzepte der Umsetzung mit CSS durch Angehensweisen mit XSL-FO verglichen, wobei das Augenmerk nicht auf der Erklärung des XSL-FO-Codes liegt, sondern dieser nur beispielhaft gezeigt wird. XSL-FO formatiert dasselbe Ausgangsformat (MARC-XML) zum gleichen Zielformat (PDF) und das mit sehr ähnlichen Eigenschaften und Funktionen. Dadurch ist ein Vergleich der Elemente in der Anwendung möglich.

1.3 Motivation für die Computerlinguistik

Diese Abhandlung betrachtet die Verarbeitung von sprachlichen Metadaten zu anschaulichem Printlayout. Es findet eine empirische Untersuchung einer vorhandenen Technologie zur Metadatenverarbeitung in einem bisher noch relativ offenen Anwendungsbereich (Cross-Media-Publishing bzw. Print-Publishing) statt.
Durch die Kombination vom Fachbereich Informatik (XML-Technologie), Verlagswesen (Layoutformatierung, Cross-Media Publishing) und Linguistik (Anreicherung von Inhalten mit Metadaten durch Markup-Sprachen) entsteht eine interessante Vermengung, die zur Erforschung und Weiterentwicklung der automatisierten Layoutformatierung von Printprodukten durch CSS anregt.

2 Daten und Vorprozess

Im Verlauf der Arbeit werden Metadaten im XML-Format (genauer MARC-XML) von einer XML-basierten Programmiersprache (XSLT) zu einem weiteren Markup-Format (HTML) verarbeitet.

2.1 Metadaten in XML

Extensible Markup Language (kurz XML) ist ein offener W3C-Standard, um Daten zu serialisieren, d.h. sie zu kodieren und sie mit syntaktischen und semantischen Informationen anzureichern. Durch das „Prinzip der Trennung von Inhalt, Struktur und Gestalt“[3] wird die Speicherung, die Organisation, die Instandhaltung, der Austausch sowie die Wiederverwendung dieser Daten erleichtert.
Spezielle semantische Auszeichnungen und Codierungen werden verwendet, um textliche, numerische oder auch andere Daten (Grafiken, Tondaten etc.) zu strukturieren.
Mit XML können komplexe strukturelle Informationen durch ein einfaches text-basiertes, also sprachliches Format repräsentiert, erweitert und angepasst werden. XML-Dokumente besitzen eine Baumstruktur, mit Elementknoten und Verzweigungen, und können so Hierarchien erstellen, die auch einfach wieder durchforstet werden können.[4]

Als Metadaten werden, wie bereits in der Einleitung erwähnt, zusätzliche sprachliche Daten über Inhalte bezeichnet - „Daten über Daten“[5] ist eine beliebte Definition. Diese Daten sind maschinenlesbar und enthalten u.a. syntaktische Annotationen, semantische Informationen und weitere Eigenschaften von (linguistischen) Ressourcen. Was Daten zu Metadaten macht, das wird allerdings hauptsächlich vom Verwendungszweck bestimmt.

Im Bibliothekswesen hat z.B. ein Buch in der Datenbank u.a. Metadaten über den Titel, die Autoren, den Herausgeber und Angaben zu Zeit und Ort der Veröffentlichung. Ebenso sind aber auch strukturelle Informationen über die Seitenabfolge eines zusammenhängenden Werks, Layoutinformationen und Navigationsstrukturen wie Suchindexe oder Inhaltsverzeichnisse Metadaten.

Die unterschiedlichsten Inhalte können die unterschiedlichsten Metadaten besitzen, so wird bspw. für einen Korpus die Länge gespeichert, für eine Ton-/Sprachaufnahme das Datum und zu linguistischen Annotationen der Grund, wofür die Annotationen bestimmt sind. In Anbetracht dessen gibt es so viele verschiedene Vorschläge für Metadatenformate. Es verhält sich ähnlich

[3] Götz/Ott, S.15
[4] Cole/Han, S.4
[5] Stührenberg, S.233

zur maschinellen Übersetzung: Ist etwas in einer Sprache vorhanden, so kann es übersetzt werden; wenn ein Wort oder eine grammatische Form nicht erlernt wurde, kann es auch nicht verarbeitet werden. Analog ist es mit Metadatenformaten. Manche Metadaten können in dem einen oder anderen Metadatenkonzept nicht umgesetzt und eingepflegt werden, da für die Information passende Strukturen oder Elementcontainer fehlen. Deswegen können linguistische Metadaten je nach Sinn und Zweck der Anwendung in verschiedenen Formaten untergebracht werden.

Metadaten können entweder generisch entstehen (z.B. Dublin Core) oder für bestimmte Domänen entwickelt werden.[6] Die Art und Weise, wie Metadaten gespeichert werden, ist entscheidend für die Effizienz der Weiterverarbeitung. Es gibt viele verschiedene Formate, um die gewünschten Informationen zu speichern. Auszeichnungssprachenformate wie z.B. XML werden hierbei gerne verwendet.[7]

Im Laufe der Zeit hat sich die Computerspeicherkapazität so stark entwickelt, dass man heute mit sehr großen, digitalen Korpora arbeiten kann. Dadurch wurden Metadaten allerdings immer wichtiger, um den Überblick über die großen Datenmengen sowie ihrer vieler Annotationen zu erhalten und den Zugriff bzw. die Analysemöglichkeit für bestimmte Aufgabenstellungen zu unterstützen.[8]

XML-Formate spielen daher auch eine Rolle bei der Effizienz der Datenspeicherung. XML-Metadaten können verschiedene Formen annehmen, wie u.a. Annotationen (z.B. bei TEI; Kommentare oder Part-of-Speech/PoS), Katalogisierungsdaten (z.B. bei MARC; Titel, Autor, ISBN), Querverweise.[9]

Die standardisierte, von Darstellung und Verarbeitung unabhängige und austauschbare Datenspeicherung mit XML bietet eine einfache Interoperabilität. [10]

2.2 Ausgangsformat MARC-XML und andere linguistische Metadatenformate

2.2.1 MARC21 und MARC-XML

Ein XML-Format für die Speicherung von Metadaten ist MARC-XML, das Ausgangsformat der in dieser Arbeit verwendeten Daten. Was dieses Format ist und wofür es von Nutzen ist, soll nun dargelegt werden.

MARC-XML ist ein erweiternder XML-Framework des Marc21-Formats, das vom „Library of Congress“ in den sechziger Jahren als Kombination der bereits existierenden britischen und amerikanischen „Machine-Readable-Cataloging“-Formate (kurz: MARC) entwickelt wurde.

[6] Stührenberg, S.233
[7] Ahmed/River-Moore et. al, S.11
[8] Stührenberg, S.233
[9] Ahmed/River-Moore et. al, S.13f
[10] O.V. (2004): „Vorteile von XML”

MARC erleichtert die Erstellung und den Austausch von computergestützten Katalogdatensätzen und ist neben der Speicherung von bibliografischen Daten auch für u.a. Norm-, Klassifikations-, Bestandsdaten sowie Community-Informationen nutzbar.[11]

MARC wurde ursprünglich auf Magnetbändern gespeichert und ausgetauscht, weshalb die Struktur recht umständlich gestaltet werden musste. MARC-XML bietet einen flexiblen und effizienteren Weg für die Erstellung, Lieferung und den Austausch von bibliografischen Metadaten und stellt den Zugriff zu bibliotheksverwaltenden Ressourcen bereit.[12]

Während die Zeichenposition maßgebend in der MARC-Struktur ist (Speicherung als Sequenzen), können strukturelle Elemente in MARC-XML durch Markupzeichen („<…>“) codiert und so Einträge erweitert oder geändert werden.

Grob umrissen besteht ein MARC-XML-Dokument aus einer Wurzel (***<record>*** oder ***<collection>***) und darin enthaltenen Feldern (***<controlfield>*** oder ***<datafield>***), die u.a. durch `tag`-Attribute spezifiziert werden (*s. Anhang: Code 1*).
Eine ***<collection>*** besitzt mehrere ***<record>***–Elemente, und diese wiederum beschreiben jeweils ein Werk. Ein ***<record>*** wird durch genau ein darin enthaltenes Kopfelement (***<leader>***) beschrieben, daraufhin folgen Kontrollfelder und Datenfelder. Datenfelder beinhalten ein oder mehrere Unterfelder (***<subfield>***), definiert durch semantisch-bedeutungsvolle `code`-Attribute. In den Unterfeldern ist letztendlich der eigentliche Inhalt enthalten.

Durch das `tag`-Attribut mit dem MARC-Code können die Datenfelder unterschieden werden, denn diese bestimmen, welche Informationen das Datenfeld enthält. So steht z.B. die Zahl ‚245‘ für den Titel eines Werks.[13]

Mit MARC-XML ergibt sich eine bessere Tranformationsmöglichkeit (durch Wiederverwendung), Präsentation (durch simple Umformung in XHTML) und Analyse (durch die XML-Baumstruktur).

Die in dieser Abschlussarbeit verwendeten Beispieldaten sind aus der Datenbank der „Deutsche[n] Nationalbibliothek“. Aus dem Metadatenshop der Homepage[14] wurden frei verfügbare Titeldaten von Zeitschriften nach dem Stichwort ‚Linguistik‘ und dem Zeitraum ‚ab 2000‘ gefiltert und als MARC-XML-Daten heruntergeladen. Nach welchen Parametern die Daten ausgewählt wurden, ist für den weiteren Vorgang aber letztendlich nicht relevant.

---

[11] Welsh/Batley, S.131
[12] Cole/Han, S.67
[13] Ebd., S.75
[14] DNB Metadatenshop: https://portal.dnb.de/metadataShop.htm (Stand: 22.03.17)

Neben diesem genannten MARC-Metadatenformat gibt es noch viele weitere XML-basierte Formate, die auf ihren jeweiligen in der Linguistik verorteten Zweck und Schwerpunkt abgestimmt sind.

2.2.2 Dublin Core

Das *Dublin Core Metadata Element Set*[15] (DCMES oder auch einfach Dublin Core bzw. DC) ist seit 1995 verfügbar – übrigens benannt nach der Stadt Dublin in Ohio, nicht Irland – vorgestellt von der DCMI (Dublin Metadata Initiative) als „Jedermanns Metadatenstandard für das Web“[16]. Nach dem MARC-Format sei DCMES der meist benutzte (einfache) deskriptive Metadaten-Standard in der Bibliotheks-Domäne, so Timothy W. Cole und Myung-Ja Han, die Autoren des Buches „XML for Catalogers and Metadata Librarians“.

Bibliotheksressourcen, z.B. Texte und andere digitale Ressourcen, können durch Dublin Core ganz einfach erkundet werden. Dublin Core ist heutzutage in zwei Standards, dem festgelegten ‚*Simple (unqualified) Dublin Core*‘ oder als ‚*Qualified Dublin Core*‘, erhältlich. Ersterer enthält 15 Kernelemente: *identifier, format, type, language, title, subject, coverage, description, creator, publisher, contributor, rights, source, relation, date*.[17] Eine eindeutige Identifizierung des Dokuments (z.B. ISBN) wird mit *identifier* gespeichert, die Sprache des Dokumentinhalts wird durch *language* codiert usw. Diese sind unveränderlich und werden im *Qualified Dublin Core* mit drei weiteren Entitätsklassen erweitert.[18]

Das Konzept ist so allgemein und grundlegend, sodass es für fast alle Datenarten verwendet werden kann, aufbauend auf einfachen RDF-Implementationen. So wird die Beschreibung und Erforschung von Bibliotheksressourcen auch für Laien möglich. Außerdem lässt sich DCMES nicht nur in XHTML einbetten, sondern ist auch mit vielen weiteren Metadatenstandards kompatibel, da diese Obermengen oder Erweiterungen von Dublin Core sind.[19]

Durch das dokumentähnliche Aussehen, das für den nicht-professionellen Metadatennutzer entwickelt wurde, ist es aber oft zu simpel, um volle Effektivität bei der Beschreibung von digitalen Ressourcen zu erreichen. Viele u.a. Bibliotheken verwenden DC als Basis-Metadatenset für eine breite Palette digitaler Sammlungen. Auf DC bauen obendrein viele reichere Metadatenstandards (z.B. TEI) auf.[20]

---

[15] Offizielle Webseite der DCMI: http://dublincore.org (Stand: 22.03.17)
[16] Übersetzt von: Cole/Han, S.98
[17] DCMI (2012)
[18] Miller, S.95
[19] Stührenberg, S.234
[20] Miller, S.95f

2.2.3 IMDI

Die *ISLE Metadata Inititative* (*IMDI*) entwickelte ein Metadatenschema für multimodale und multimediale Sprachressourcen. Dieses ist zweigeteilt, aufgrund der verschiedenen Ressourcentypen, in eine Spezifikation für Sitzungsmetadaten (*Metadata Elements for Session Descriptions*, IMDI Part 1) und die Katalogmetadatenspezifikation (*Metadata Elements for Catalogue Descriptions*, IMDI Part 1B).

IMDI Part 1B wird für die Katalogisierung von multimodalen Sprachressourcen verwendet. Damit werden publizierte Korpora auf Katalogebene beschrieben. Diese Metadaten könnten auch in Dublin Core übertragen werden, da nicht viele detaillierte Informationen über die Bestandteile der Korpora aufgenommen werden.[21]

Sitzungsmetadaten (IMDI Part 1) wiederum beschreiben u.a. einen nicht-publizierten Text oder eine Aufzeichnung mit deren Annotationen,[22] d.h. nicht nur die gesamten Korpora können implementiert und durchsucht werden, sondern auch individuelle Ressourcen innerhalb dieser Korpora. So können detailliertere Anfragen gestellt werden, doch dazu wird eine stärker erweiterbare Elementmenge für Metadaten benötigt.[23]

IMDI baut aber nicht auf Dublin Core auf, sondern richtet sich nach den Bedürfnissen vieler Sprachwissenschaftler bei der Katalogisierung multimodaler Daten (z.B. auditive, textuelle, sprachliche oder visuelle Ressourcen).[24]

Dieses Metadatenformat ist nicht nur für die Ressourcenfindung, sondern auch von Vorteil für die Verwertung und Verwaltung großer Korpora. Durch verknüpfte Hierarchien wird das Durchsuchen und Verwalten der Daten unterstützt.[25] Möchte man also in einen Korpus hineingehen und dort Metadaten abspeichern und erforschen, so ist bevorzugt IMDI zu verwenden.

2.2.4 TEI

Eine der „dienstälteste[n] Spezifikation[en] zur Strukturierung textueller Informationen“[26] ist die der *TEI Guidelines for Electronic Text Encoding and Interchange*, die seit ihrer Version P4 von 2002 auch auf einer XML-Grundlage aufbaut (zuvor SGML). Aktuell ist seit 2007 Version P5.

---

[21] IMDI (2009): „Metadata Elements for Catalogue Descriptions”
[22] Stührenberg, S.237f
[23] IMDI (2009): „Metadata Elements for Session Descriptions”
[24] Stührenberg, S.237
[25] Ebd.
[26] Ebd., S.123

Der Fokus der TEI-Guidelines liegt hauptsächlich auf der Anwendung in den Geisteswissenschaften. Die Kodierung von gedruckten Korpora und die Annotation linguistischer Informationen ist durch die sachbezogenen (optionalen) Module des erweiterbaren XML-Schemas der Guidelines möglich.[27] Jegliche Textarten und Sprachen können vom Anwender (z.B. Philologe) individuell kodiert und annotiert werden[28], da die TEI Richtlinien dem Nutzer einerseits genügend Freiraum für eigene Textumsetzungen, je nach Forschungsinteressen, gibt, andererseits aber ein einheitliches Vokabular und eine einheitliche Syntax der Metasprache bereitstellt. Durch die Standardisierung wird die Austauschbarkeit der Daten zwischen Wissenschaftlern gewährleistet.

Es gibt Elemente für u.a. Datenstrukturen, Annotationen von Gedichten, Dramen und Wörterbüchern, gesprochene Sprache, Katalogisierungsinformationen bibliografischer Daten etc.[29] Version P5 bietet u.a. auch einen Standard zur Handschriftenbeschreibung.[30] Bibliografische Daten lassen sich z.B. im sogenannten *TEI Header* (*teiHeader*), genauer in der file description (*fileDesc*) dokumentieren. Dabei können andere Metadaten aus Standards wie Dublin Code oder MARC mit eingebettet werden (mit *xenoData*).[31]

2.2.5 CMDI

Eine weitere Technologie zur Metadatenbeschreibung ist der Framework *Component Metadate Infrastructur* (CMDI) von CLARIN. Wie der Name schon sagt, ist das eine komponentenbasierte Art, Metadaten darzustellen, indem u.a. bestehende Metadatenentwürfe (wie z.B. TEI und IMDI) unter einem Dach zusammengeführt werden, die dann auch erweitert und modifiziert werden können.

Je nach Bedürfnissen des Nutzers können Datenkategorien, Komponenten oder ganze Metadatenprofile, d.h. fertige Metadatenformate, individuell ausgewählt werden. Auch bei CMDI besitzt der Wissenschaftler, der die Metadatenbeschreibung anlegt, viel Freiheit bei der eigenen Umsetzung, es ist aber trotzdem auch eine Standardisierung möglich. Selbst wenn Elemente unterschiedlich benannt werden, die Datenkategorien sind dieselben und werden per PI (*Persistant Identifier*), einer Art Datenkategorienregistierung von ISOcat[32], mit den verwendeten Elementen verknüpft. [33]

---

[27] TEI (2015)
[28] Stührenberg, S.123
[29] TEI (2016): „P5: Guidelines for Electronic Text Encoding and Interchange"
[30] Ebd., „10 Manuscript Description"
[31] Ebd., „2 The TEI Header"
[32] Offizielle Webseite von ISOcat: http://www.isocat.org/ (Stand: 22.03.17)
[33] CLARIN (o.J.)

Die Verbindung der verschiedenen Metadatenstandards bzw. -entwürfe ist sehr praktisch bei vielfältigen und großen Projekten, aber nicht immer wird alles benötigt. In manchen Anwendungsfällen reicht auch einfach eines der Formate aus, um anwendungsspezifische Bedürfnisse umzusetzen.

2.3 HTML5 als Zielformat

Nun soll aus dem Metadatenformat ein *HTML5*-Dokument hergestellt werden, das im weiteren Prozess für die CSS-Formatierung verwendet wird.

HTML (Hypertext Markup Language) ist eine Markupsprache, eine textbasierte Auszeichnungssprache, die zur Strukturierung und zur Anreicherung semantischer Auszeichnungen von Dokumentinhalten (Texte, Bilder, Hyperlinks) verwendet wird.

Warum wird ausgerechnet in HTML(5) transformiert? Man kann zwar auch andere XML-Strukturen verwenden, allerdings bietet der W3C-Standard HTML neben der Kompatibilität mit CSS noch weitere Vorteile, um es zur weiteren Verarbeitung zu verwenden. Der wohl größte besteht darin, dass HTML eine Markupsprache ist, die unter Programmierern als Websprache bekannt und weit verbreitet ist.[34]

Version 5 des HTML-Formats enthält zudem logische und semantische Strukturelemente für einfaches und gängiges Printprodukt (*article*, *section*, *aside*, *nav* etc.). Dieses Format ist zwar nicht immer komplex genug für jedes Produkt, aber für einfache Printdokumente reicht es völlig aus.[35]

Zusätzlich ist bei den meisten Formatierern ein fundamentales CSS-Stylesheet hinterlegt, das schon das allgemeine HTML-Format umsetzen kann. Tabellen, Listen, Grafiken, Überschriften, Absätze und Weiteres werden automatisch als solche dargestellt, sodass man viel schneller und einfacher Details anpassen kann, ohne sich erst mit der zeitaufwändigen Umsetzung oben genannter Elemente auseinandersetzen zu müssen. Sollte etwas anders umgesetzt werden, als in diesem CSS Stylesheet beschrieben, so kann man es einfach im eigenen Programm überschreiben.

Ein weiterer Punkt, der für HTML5 spricht, ist die gemeinsame Grundlage bei crossmedialen Publikationen. Denn wie bereits beschrieben wurde, ist HTML (meistens) die Basis von E-Book-Formaten und Webseiten. Ein Inhaltsformat, eine Transformationsprache aber mehrere Outputformate, so das Wunschziel des Cross-Media-Verlagswesens. „Digital first“ ist hier ein

[34] Kleinfeld/Sanders
[35] W3Schools: „HTML5 Introduction”

Stichwort – es bedeutet, dass man aus digitalen Quellen wie z.B. Webseiteninhalten Printprodukte erstellen kann. Dafür benötigt man eine standardisierte Semantik, wie sie HTML5 bietet.[36]

2.4 Überführung/Transformation der Ausgangsdaten mit XSLT

Um von MARC-XML zu HTML5 zu kommen, muss eine Transformationssprache zwischengeschaltet werden. Bevor aber genauer darauf eingegangen wird, was XSLT ist, werden die Gründe dargelegt, wieso überhaupt ein solcher Vorprozess vonnöten ist.

Leider lässt sich mit CSS nur sehr wenig Inhalt und kaum zusätzliche Strukturen generieren. D.h. Verzeichnisse (z.B. Inhaltsverzeichnisse), Register, zusätzlicher Text u.ä. und auch eine veränderte Elementreihenfolge müssen vor dem eigentlichen Layoutprozess aufgebaut und eingepflegt werden. Diesen Umstrukturierungsprozess könnte man ebenso mit Javascript o.ä. durchführen, da die Daten aber ein XML-Format aufweisen, bietet sich die XSLT-Benutzung an. XSLT agiert auf XML-Baumstrukturen und wandelt sie um.

Extensible Stylesheet Language (XSL) nennt man eine Sammlung an Programmiersprachen, die üblicherweise nach Regeln des XML-Standards aufgebaut sind (eine Ausnahme ist z.B. *XPath*) und u.a. XML-Dokumente verarbeiten bzw. gestalten.[37]

XSL Transformation (XSLT) ist eine dieser Sprachen - eine funktionale Programmiersprache zur Transformation von XML-Dokumenten. Aufbauend auf der logischen Baumstruktur eines XML-Dokuments kann man in XSLT-Stylesheets Umwandlungsregeln definieren, die ein oder mehrere Dokumente in ein beliebiges (X)ML- oder Textausgabeformat umwandeln. Das geschieht mithilfe von *XPath*, einer Abfragesprache, mit der man alle Knoten im XML-Dokument genau adressieren und somit darin navigieren kann.[38] Durch XSLT-Prozessoren, welche auch in den meisten modernen Webbrowsern (u.a. Opera, Firefox und Internet Explorer) integriert sind, werden die Anweisungen des XSLT Stylesheets verarbeitet. Teile des Dokumentbaums können u.a. übertragen, umstrukturiert, vervielfacht oder gelöscht werden.

Die Elemente eines Dokuments werden mit XSLT nacheinander in sogenannten „templates“ abgefragt bzw. aufgerufen und darin dann modifiziert. Der Rahmen des neuen Dokuments stellt einen Aufruf des Wurzelelements dar (`<xsl:template match="/">`), innerhalb dem der Grundaufbau bestimmt werden kann. So kann für jedes dieser Elemente beschrieben werden, wie und wo es im resultierenden Outputdokument auftauchen soll. Z.B. kann eine Transformation in ein (sehr) einfaches HTML-Dokument beschrieben werden (*s. Anhang: Code 2*), das die

[36] Kleinfeld (2013)
[37] W3Schools: „XSLT Introduction“
[38] Montero/Krüger, S.41

<title>–Elemente im Überschriftenelement **<h1>** und den Rest der originalen Daten einfach nur in einem Absatzelement darstellt (**<p>**).[39]

Eine mögliche Ausgabesprache von XSLT zur Layoutformatierung stellt zudem *FO* (Formatting Objects) dar, welches auf XSLT abgestimmt wurde und im folgenden Kapitel genauer beschrieben wird.

Mit XSLT werden nun die MARC-XML-Daten für den Zweck der CSS-Formatierung umgewandelt. Abschnitte, wie u.a. die Titelseite, der Willkommenstext, das Inhaltsverzeichnis, das Register und Überschriften werden generiert. Es wird bestimmt, welche Elemente nicht angezeigt werden und die Elementreihenfolge wird verändert. Außerdem wird alles in eine valide HTML5-Struktur gebracht.

3 Überblick XSL-FO und CSS

3.1 XSL-FO

XSL-Formatting Objects (FO) ist wie XSLT ein Mitglied der XSL-Familie und wurde erstmals 2001 als Version 1.0 sowie 2006 in der Version 1.1 als W3C-Recommendation veröffentlicht. In Kombination mit XSLT lässt sich mit FO die komplexe Darstellung von Dokumenten in festen (digitalen) Seitendimensionen oder auch für Druckerzeugnisse formatieren.[40]

Neben den Seitenmaßen lässt sich festlegen, wie Texte, Linien, Bilder, Flächen und andere Elemente einer Seite auf dieser angeordnet werden. Fünf unterschiedliche Unterteilungen in Regionen hat eine Seite des XSL-FO-Konzepts (Kopf- und Fußbereich, linker und rechter Rand und den Inhaltsbereich).
Darüber hinaus können aber auch Seitenabfolgen der Seitenvorlagen bestimmt werden, z.B. ein Doppelseitenlayout mit abwechselnd rechter und linker Seite. Eine detaillierte Gestaltung des Textes, Umbrüche oder typografische Anforderungen u.a. an das Schriftbild kann XSL-FO genauso gezielt umsetzen.

Möchte man nun ein XML-Dokument mit einem XSL-FO-Stylesheet gestalten, so passieren im Hintergrund mehrere Prozesse (*s. Anhang: Abb. 1*). Zuerst wendet ein XSLT-Prozessor die XSL-FO-Transformationsregeln auf das Eingangsdokument (z.B. aus einer Datenbank) an und heraus kommt ein weiteres, statisches XML-Dokument, das mit allen FO-Formatierungseigenschaften und generierten Inhalten angereichert wurde. Dieses „XML-FO“-Dokument wird nun von einem weiteren XSL-Prozessor verarbeitet, der Grafiken einbindet und das Ganze zu einem Output-

[39] Montero (2004)
[40] W3C: „6 Formatting Objects”

Generator weiterleitet, der dann entweder ein PDF-, SVG- oder PS-Dokument erstellt. All das sind für den Endnutzer unsichtbare Schritte und geschehen vollständig automatisiert.[41]

XSL-FO baut in ein transformierendes XSLT-Stylesheet Formatierungselemente ein (z.B. **`<fo:block>`**), verschachtelt so den Inhalt des Dokuments und gestaltet ihn mit einer Vielzahl an Attributen.

Ein wichtiger Punkt, der zur zukünftigen Benutzung von CSS zu Layoutformatierung motiviert, ist, dass nach der XSL-FO-Version 1.1 letztendlich die offizielle W3C working group geschlossen wurde, aufgrund des scheinbar zu komplizierten Konzepts, und somit die Weiterentwicklung eingestellt wurde.

3.2 CSS(3)

Cascading Style Sheets (CSS) wurde in den 1990er Jahren entwickelt mit der Hauptmotivation, die Formatierungsangaben aus dem HTML-Dokument zu lösen. HTML sollte nur den Inhalt bzw. die Semantik beschreiben und strukturieren, während CSS sich mit der Darstellung beschäftigt.[42] So ist es möglich, die Formatierung mehrerer Dokumente zentral und global durch nur ein externes Stylesheet zu ändern.[43] CSS wird zwar hauptsächlich für HTML-Dokumente verwendet, es lassen sich aber auch weitere (X)ML-Formate damit formatieren.

Ein CSS Stylesheet besteht aus einer oder mehreren Regeln, welche sich jeweils in zwei Teile aufspalten: Selektor und Deklarationsblock. Der Selektor weist auf ein Element im Dokument, das man formatieren möchte, und im Deklarationsblock wird das Erscheinungsbild in Form von CSS-Eigenschaften und deren Werten deklariert:[44]

```
p {
        font-size:12pt;
}
```

Code 3: einfache CSS-Deklaration

Durch den HTML/XML-Dokumentbaum kann CSS das Prinzip der Vererbung verwenden. Nachkommenknoten weiter unten im Baum bekommen (fast alle) Eigenschaften von ihren Vorfahren vererbt. Diese können dann im jeweiligen Deklarationsblock der selektierten Nachkommen überschrieben werden. Es gilt: die spezifischeren Selektionen (weiter unten im

[41] Montero/Krüger, S.13f
[42] Mehr dazu siehe: Lie (1994)
[43] W3Schools: „CSS Introduction"
[44] Meyer, S.5f

Baum, detaillierter selektiert) haben bei der Umsetzung Vorrang vor den allgemeineren (Kaskadierung). [45]

Seit 2000 entwickelt das W3C nun eine Version der Technologie, die auch professionelle, XML-basierte Printlayouts produzieren soll. *CSS level 3* (CSS3) befindet sich bis heute allerdings noch großteils im *W3C Working Draft Status* und wurde noch nicht von W3C standardisiert, was zur weiteren Erforschung und Testung der Sprache zu Printzwecken anregt.[46]

Anders als die vorherigen Versionen, besteht diese Version nicht mehr aus einzelnen Spezifikationen, sondern ist in Module unterteilt. Dabei bleibt CSS3 komplett abwärts-kompatibel zu den älteren Versionen. Es gibt mehr als 50 CSS-Module, mit alten und neuen Eigenschaften, die sich aber in verschiedenen Fertigkeitsstufen befinden. Die u.a. zum Einsatz kommenden Module sind:[47]

- CSS Paged Media Module Level 3 (CSS3PAGE)
- CSS Generated Content for Paged Media (CSS3GCPM)
- Selectors Level 3 (CSS3SELECT)
- CSS Multi Column Layout (CSS3COL)
- CSS Fragmentation Module (CSS3BREAK)
- CSS Image Values and Replaced Content Module Level 3 (CSS3IMAGE)
- CSS Text Module Level 3 (CSS3TEXT)
- CSS Fonts Module Level 3 (CSS3FONTS)
- CSS Text Decoration Module Level 3 (CSS3TDECO)
- CSS Lists and Counters Module Level 3 (CSS3LIST)
- CSS Backgrounds and Borders Module Level 3 (CSS3BG)
- u.v.m.

Einzelne Module können bereits ein weiterentwickeltes Level 4 erreicht haben, gehören aber trotzdem zu CSS3. Die meisten Features für die Drucklayoutierung werden aus CSS3PAGE und CSS3GCPM gewonnen. Damit lassen sich u.a. die Seitengröße, -ausrichtung, Paginierung, Seiten- und Absatzumbrüche, Masterseiten (Formatvorlagen), Kopf- und Fußzeilen bestimmen.[48]

[45] Meyer, S.9
[46] Wolf, S.309
[47] W3C: „2. Cascading Style Sheets (CSS) — The Official Definition"
[48] McKesson (2012)

Erwähnenswert ist übrigens auch die Möglichkeit, mit einem CSS3 Sprachmodul (CSS3SPEECH) die akustische Umsetzung eines Dokuments durch Sprachsynthese zu steuern (Text-To-Speech-System) und so die Barrierefreiheit von Dokumenten zu gewährleisten.[49]

Diese Arbeit setzt ein Maß an Vorkenntnissen und Erfahrungen mit CSS und HTML voraus, da die detaillierte Beschäftigung mit Grundlagen den Forschungsrahmen überschreiten würde. Der Einfachheit halber wird hier außerdem nicht immer CSS3 geschrieben, sondern es ist meistens von CSS die Rede.

3.3 Vergleichbarkeit

Wieso kann man diese zwei Technologien vergleichen? Was unterscheidet sie, was haben sie gemeinsam?

Ein größerer Unterschied beider Technologien ist das Konzept für den Seitenaufbau – XSL-FOs Regionenkonzept und das Boxmodell von CSS (s. *4.1 Grundstruktur*). Außerdem unterscheiden sich beide Technologien darin, wie sie ihre Formatierungselemente einsetzen. XSL-FO nutzt Extra-Elemente, um den eigentlichen Inhalt zu um-/verschachteln (z.B. `<fo:block font-size="11pt"><p></fo:block>`), während CSS (hauptsächlich) die bereits existierenden Elemente mit Eigenschaften in Form von Attributwerten modifiziert (z.B. `<p style="font-size:11pt">`).

Auf den ersten Blick scheinen auch die Arbeitsprozesse nicht viel gemeinsam zu haben, braucht man für XSL-FO nur ein Stylesheet während CSS einen Vorprozess benötigt und dann die Interpretation des HTML mit CSS stattfindet. Doch schaut man genauer hin, bemerkt man, dass auch die XSL-FO-Verarbeitung zuerst eine Art Vorprozess stattfinden lässt. So wird ein statisches „XML-FO“-Dokument erstellt, das sich mit dem HTML-Dokument vergleichen lässt. Beide sind in ihrer Struktur starr und werden von einem Formatierungsprogramm zu einem (in unserem Fall) PDF interpretiert.

In einem ersten Schritt werden in beiden Fällen Inhalte generiert, XSL-FO bindet dabei allerdings auch noch die Formatierungseigenschaften ein, welche bei CSS in einem externen Stylesheet gespeichert werden. Daraufhin findet eine Interpretation des XML-FO- und auch des HTML-Dokuments, zusammen mit dem CSS-Stylesheet, durch einen „Formatierer“ statt. Viele der dabei zu interpretierenden Eigenschaften sind in beiden Technologien ähnlich bis deckungsgleich in ihrer Funktion und ihren Werten. Beide Technologien zur Layoutierung von Printprodukten lassen sich also recht gut in ihrer Umsetzung vergleichen.

[49] W3C: CSS3SPEECH

3.4 Formatiererwahl

Die Interpretation der Styling-Eigenschaften findet sowohl bei XSL-FO wie auch bei CSS mit einem Formatierungsprogramm (*Formatierer* oder *Renderer* genannt) statt, das Größen, Umbrüche, Positionen von Elementen und vieles mehr berechnet. Heraus kommt dann ein printfähiges Outputdokument.

CSS wird meist von Browsern und weiteren kostenlosen Formatierern unterstützt. Viele CSS3-Eigenschaften sind auch bereits in modernen Browsern implementiert.[50] Webbrowser sind auf dynamisches Rendering ausgelegt, nicht unbedingt auf die Interpretation von festgelegten Seitengrößen. Das CSS der Printmedienformatierung ist für die professionelle und qualitativ hochwertige Printlayoutgenerierung auf eine sehr leistungsfähige Formatierungssoftware angewiesen und kann somit leider von noch keinem kostenlosen Renderer vollständig verarbeitet werden.[51]

Für XSL-FO existiert FOP (*Apache Formatting Objects Processor*[52]), ein kostenloser Java-basierter Formatierer, der aus XSL-FO-Stylesheets verschiedene Ausgabeformate (u.a. auch PDF) hervorbringt. Für CSS3 besteht jedoch keine solche Formatierungslösung.

Antenna House bietet eine (nicht gerade kostengünstige) Formatierungssoftware an, den *Antenna House Formatter* (AHF, Version 6.4), die mit CSS paged media bisher am besten umsetzen kann und sich zur professionellen Satzerstellung eignet, auch für XSL-FO.[53] Es gibt aber auch eine kostenlose Evaluationsversion[54] des AHF, mit der die Daten dieser Arbeit getestet werden. AHF bietet selbst auch einige nützliche Erweiterungen für CSS, damit die Printmedienformatierung besser gelingt. Diese und die Konformität der CSS3-Elemente in AHF werden auf der Homepage des Anbieters aufgeführt.[55]

Ein Konkurrent wäre die Rendering-Software *Prince*, deren Ergebnisse aber in der Umsetzung zum Großteil den Ergebnissen von AHF unterlegen ist.[56]

Bei einem der von AHF gerenderten Outputformaten handelt es sich um ein PDF (*Portable Document Format*), welches zum Austausch von fertiggestellten Dokumenten konzipiert wurde. Dieses Format, ursprünglich von Adobe entwickelt, ist heute ein offener ISO-Standard (kostenlos

---

[50] W3Schools: „CSS3 Introduction"
[51] Götz/Ott, S.99
[52] Siehe Apache FOP: https://xmlgraphics.apache.org/fop/ (Stand: 22.03.17)
[53] Götz/Ott, S.99
[54] Siehe AHF kostenlose Testversion: https://www.antennahouse.com/product/ahf60/download.htm (Stand: 22.03.17)
[55] AH (o.J.): „CSS Conformance"
[56] Götz/Ott, S.99

und weit verbreitet), mit dem man Dokumente plattformunabhängig präsentieren und austauschen kann.[57]

Von der Originalsoftware, Hardware oder dem Betriebssystem unabhängiges Darstellen unterscheidet dieses Format von den hier bisher beschriebenen. HTML, XML, XML-FO etc. sind je nach Maschinenumsetzung, d.h. der Interpretation bzw. Implementation des Renderers, verschieden. PDF gewährleistet ein Enddokument, das genau so formatiert ist, wie der Ersteller es bestimmt hat – auch auf anderen Geräten und in gedruckter Variante.

Es existieren inzwischen zwar Programme, die das Modifizieren des PDF-Dokuments (z.B. Unterzeichnen) ermöglichen, aber das Format ist grundlegend so aufgebaut, dass es nicht stark veränderbar sein soll. Der Vorteil, der das PDF-Format so beliebt und nützlich macht, ist eine feste geometrische Seitenbeschreibung, d.h. auch feste Umbrüche.

4 Umsetzung typografischer Konzepte

Um ein Dokument zu einem Print-PDF zu formatieren, werden unterschiedliche typografische Konzepte umgesetzt.

Typografie beschreibt im Grunde den Gestaltungsprozess einer Seite inklusive allem, was darauf zu sehen ist.[58] Dies geschieht durch Kombinationen von typografischen Eigenschaften, die Konzepte bilden. So wird ein Inhaltsverzeichnis bspw. nicht nur mit einer, sondern mehreren Eigenschaften formatiert, die das Seitenformat (Größe, Ränder etc.), Texteffekte (Schriftart, -größe, Hervorhebungen etc.), Umbrüche (Seiten-, Zeilenumbruch und Silbentrennung), Abstände (z.B. zwischen Titel und Ordnungszahl, zwischen Titel und Seitenreferenz, oder Zeileneinzüge), Verweise auf Referenzseiten und vieles mehr betreffen.

Wie die Technologie CSS im Gegensatz zu XSL-FO die Konzepte umzusetzen versucht, soll im Folgenden erläutert werden. Wie funktioniert das Konzept und wie gut ist die Umsetzung? Während die XSL-FO-Umsetzung eher kurz und konzeptuell vorgestellt wird, wird die Theorie der CSS-Umsetzung auch anhand der bereits vorgestellten Beispieldaten gezeigt.

Die Reihenfolge der folgenden Abschnitte und die in der vorliegenden Arbeit verwendeten XSL-FO-Methoden orientieren sich hauptsächlich an der Publikation „XSL-FO in der Praxis“ (2004) von Manuel Montero Pineda und Manfred Krüger.

Innerhalb der einzelnen Abschnitte wird nach der allgemeinen Erklärung des Bereichs in konkrete typografische Konzepte unterteilt, deren Umsetzung sinnvoll wäre. Mit deren Realisierung durch

[57] O.V. (o.J.): „Was ist Adobe PDF?“
[58] U.a. Köhler, S.6f

beide Technologien wird sich dann theoretisch und mit CSS auch konkret und praktisch auseinandergesetzt. Mögliche Problematiken werden so schließlich aufgezeigt und unterschieden, ob es sich um einen Konzeptfehler handelt, d.h. in CSS3 ist eine Funktion oder Eigenschaft nicht vorgesehen und es existiert kein passendes Element, oder um einen Implementierungsfehler des Renderers, d.h. die Programmierung der vorhandenen CSS-Elemente sind noch nicht zufriedenstellend oder gar nicht vorhanden.

Zuerst werden die minimalen Anforderungen der Grundstruktur und dann die allgemeine Seitenformatierung betrachtet. Daraufhin folgen die Formatierungsmöglichkeiten des Inhaltstexts, anderer Inhaltselemente (z.B. Tabellen oder Grafiken) und zum Schluss die Generierung von Inhalten und Strukturen, wie u.a. Kolumnentitel oder Verweise/Referenzierungen.

4.1 Grundstruktur

Zuerst wird die Umsetzung des Seitenaufbaus und minimale Anforderungen an ein Printlayout mit XSL-FO und CSS erläutert und verglichen.

Mit minimalen Anforderung ist gemeint, welche Funktionen und Eigenschaften im Code vorhanden sein müssen, damit eine Datei in ein druckfähiges Print-PDF umgewandelt werden kann. Dieses muss noch keine ästhetische Inhaltsstruktur besitzen, aber die Grundstruktur des Dokuments muss vorhanden sein und auf Seiten abgebildet werden. Das betrifft allgemein Seitengröße, -format und -ausrichtung, den Satzspiegel (die bedruckbare Nutzfläche der Seite) und Einstellungen für die Seitenränder/Stege.

4.1.1 XSL-FO

XSL-FO verwendet ein Regionenkonzept für den Seitenaufbau (*s. Anhang: Abb 2/3*) mit zusätzlichen Einstellungen für eine Seitenreihenfolge.

Für die Definition einer Seite werden hier Angaben für die Seitenbreite (`page-width`), -höhe (`page-height`) und den Rand (`margin-top, -right, -bottom, -left`) benötigt. Diese Eigenschaften werden im Element ***<fo:simple-page-master>***, dem Bereich für die strukturellen Seiten(format)vorlagen, angegeben (**[1]**). Zusätzlich werden als Kindelemente des ***<fo:simple-page-master>***-Elements fünf Druckbereiche definiert, die eine Seite weiter unterteilen, ausgerichtet an unserer gewohnten Schreibrichtung (von links nach rechts, von oben nach unten). Darunter befinden sich der Kopf- und Fußbereich (***<fo:region-before>***, ***<fo:region-after>***),

die linke und rechte Region (**<fo:region-start>,< fo:region-end>**) und der Hauptbereich (**<fo:region-body>**) (**[2]**).[59]

In **<fo:page-sequence>** werden dann die Seitenformatvorlagen mit dem jeweiligen darzustellenden Inhalt verbunden und eine oder mehrere Seitenfolgen erstellt (**[3]**). Auch Seitenformat-Vorlagen werden darin angewendet, dazu aber in Abschnitt *4.2.3 Seitenvorlagen* mehr.

Um mit XSL-FO die minimalen Anforderungen an einen Druckoutput umzusetzen, muss zuerst die Seitenformatvorlage für Inhaltsseiten und deren Auftauchen in der Seitenreihenfolge festgelegt und dann diese mit dem Inhalt verknüpft werden.

Der folgende Code für die Minimalanforderungen ist nur zur Veranschaulichung der Komplexität der Sprache abgebildet und wird nicht detailliert beschrieben:[60]

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:axf="http://
  www.antennahouse.com/names/XSL/Extensions">
  <xsl:output method="xml" version="1.0" indent="yes"/>
  <xsl:template match="/">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
        <fo:simple-page-master margin="2cm 1cm 2cm 2cm" master-name="Inhalt-
Seiten" page-height="297mm" page-width="210mm">          [1]
          <fo:region-body/>        [2]
        </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-reference="Inhalt-Seiten">          [3]
        <fo:flow flow-name="xsl-region-body">
          <fo:block>
            <xsl:apply-templates/>
          </fo:block>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>
</xsl:stylesheet>
```

Code 4: XSL-FO Grundstruktur

4.1.2 CSS [CSS3PAGE]

Während XSL-FO mit einem Regionenkonzept arbeitet, überträgt CSS eine Seite auf Boxen (*s. Anhang: Abb. 4/5*), die jeweils einen Außenabstand (`margin`), einen Rand (`border`), einen

[59] Montero/Krüger, S.67ff
[60] Angelehnt an ebd., S.65

Innenabstand (`padding`) und einen Inhaltsbereich (`content`) besitzen. Die Seite selbst ist auch eine solche Box.

Über eine oder mehrere Seitenboxen („page boxes“), die sich wie Container für all die anderen generierten Boxen verhalten, fließt der Inhalt im Seitenbereich („page area“), eingeschränkt je Seite von den Rändern („margin“). In der Paged-Media-Spezifikation des W3C wird der „margin“ in 16 verschiedene Boxen zerlegt („page margin boxes“), die auch jeweils einen eigenen „margin“, „border“, „padding“ und Inhaltsbereich („content/context“) besitzen. Diese Margin-Boxen werden u.a. für laufende Kopf- und Fußzeilen verwendet, dazu in Abschnitt *4.5.1 Marginalien* genaueres.[61]

Welche Eigenschaften eine Seitenbox besitzen soll, wird mithilfe der **`@page`**-Regel und deren Deklarationsblock festgelegt. Darin können u.a. Default-Größenmaße sowie Eigenschaften von Marginalboxen eingestellt werden. Mit der `size`-Eigenschaft wird der enthaltene Block der Seitenbox spezifiziert, die Höhe, Breite und jeweiligen Ränder werden automatisch berechnet (der normale Fluß eines nicht-modifizierten Blockelements; Ränder fallen mit Randkanten der Seitenbox zusammen, wenn sich die angegebenen Werte überlappen).[62]

Der erste Schritt lautet also, die Seitenstruktur einzustellen – dazu reicht eine simple **`@page`**-Regel, mit einer Größeneinstellung (`size`) (**[4]**). Die restlichen Angaben werden automatisch vom Renderer generiert, wenn sie nicht anders angegeben werden.

```
@page{
  size:A4;          [4]
  margin:2cm 1cm 2cm 2cm;          [5]
}
```

Code 5: CSS Grundstruktur

Die minimalen Anforderungen von CSS für ein erstes Printlayout sind also bei weitem kürzer und einfacher auf den ersten Blick zu durchschauen, als die XSL-FO-Anforderungen – die `margin`-Werte (**[5]**) wären im Prinzip auch nicht unbedingt nötig anzugeben. Es reicht ein **`@page`**-Deklarationsblock, der mit einer XSL-FO-Seitenformatvorlage zu vergleichen ist. Mit CSS müssen viele andere Angaben nicht zusätzlich angegeben werden, um eine Seite zu erstellen. Der Inhalt der Datei wird so automatisch auf fortlaufende Seiten einer bestimmten Größe übertragen, die daran angepasste Seitenränder enthalten.

[61] W3C: CSS3PAGE, #page-model
[62] Ebd., #page-size

4.1.3 Fazit Abschnitt

Hierbei gibt es bisher noch keine Komplikationen bei der gewünschten Umsetzung, vor allem aber zu XSL-FO bei späteren Seitenformateinstellungen schneller an Grenzen stößt, was z.B. Seitenabfolgen angeht. Ein erwähnbarer Vorteil von CSS-Renderern ist, dass viele Default-Einstellungen, z.B. zu Seitenrändern und –ausrichtungen, beim Formatierer hinterlegt sind, sodass automatische Berechnungen geschehen, und der CSS-Code erst bei abweichenden Eigenschaften erweitert werden muss.

Ansonsten unterscheiden sich beide Technologien vor allem durch das Seitenkonzept. XSL-FO hat festgelegte Seitenregionen, die getrennt vom fließfähigen Inhalt eingestellt werden, während CSS pro Seite viele verschiedene Boxen hat, die sich anpassen an die umgebenden Boxen und (beinahe) denselben Aufbau besitzen. Beide Konzepte haben ihre Vorteile, so stellen Regionen klare Strukturen dar, Boxen lassen sich aber einfacher gestalten. Welches Konzept sich als besser erweist, wird sich im Laufe der Arbeit zeigen.

4.2 Seitenformatierung

Nachdem die Grundstruktur geklärt wurde, folgen nun diverse Erweiterungen und Modifikationen der allgemeinen Seitenstruktur. Diese verändern die Position und den Textfluss des Inhalts im druckbaren Bereich.

Für die Umsetzung eines Layouts für Druckerzeugnisse ist die Umbruchsteuerung ein Muss. Gut umsetzbare Seiten-, Spalten-, Zeilenumbrüche, aber auch eine korrekte Silbentrennung sind notwendig, um eine professionelle Gestaltung der Seitenstruktur zu erreichen.

Da unterschiedliche inhaltliche Abschnitte auch anders gestaltet werden sollen (z.B. Inhaltsverzeichnis oder rechte und linke Seiten), müssen sogenannte Muster- oder Masterseiten angelegt werden können. Das sind Seitenformatvorlagen, die auf die Umsetzung verschiedener Darstellungsbedürfnisse eingehen.

4.2.1 Mehrspaltensatz

Soll ein Inhalt nicht nur in einer Spalte pro Seite erscheinen, sondern auf zwei oder mehr Spalten verteilt werden, so benötigt man Seiteneinstellungen, die die Inhaltsbox entsprechend aufteilt. Z.B. für ein Namens-, Stichwort- oder Sachregister am Ende eines Dokuments lohnt sich die Aufteilung des Inhaltsbereichs in einen Mehrspaltensatz (in diesem Beispiel zweispaltig).

Der Spaltenausgleich (die gleichmäßige Verteilung des Inhalts auf der Seite mit dem Ziel gleich hoher Spalten) ist auch ein kritischer Punkt jeder Satztechnologie.

4.2.1.1 XSL-FO

Um mit XSL-FO eine Seite mit zwei nebeneinander befindlichen Spalten darzustellen, vertikal durch eine Trennungslinie getrennt, müssen zusätzliche Eigenschaften in ***<fo:region-body>*** der betreffenden Seitenformatvorlage deklariert werden, nämlich `column-count` (Spaltenanzahl) und `column-gap` (Breite des Zwischenraums zwischen den Spalten) (**[1]**).

Um die Trennlinie zu erstellen, muss eine proprietäre AH-Erweiterung für Linienart und -stärke (`afx:column-rule-style/-width`, **[2]**) verwendet werden, denn das FO-Konzept enthält selbst keine solchen Attribute. Ob andere Renderer eine solche Erweiterung implementiert haben, steht zur Frage.[63]

```
<fo:simple-page-master margin="15mm 20mm 30mm 20mm" master-name="Register" page-
height="297mm" page-width="210mm">
   <fo:region-body margin="5mm 0mm 7mm 0mm" column-count="2" column-gap="4mm"  [1]
    axf:column-rule-style="solid" axf:column-rule-width="1pt"/> [2]
</fo:simple-page-master>
…
<fo:block span="all" [3] color="#FFFFFF" border-after-style="solid" border-after-width="1pt">
  …
</fo:block>
```

Code 6: XSL-FO Mehrspaltensatz

Damit ein Spaltenausgleich gewährleistet werden kann, muss um den Inhalt beider Spalten ein unsichtbarer Block kreiert werden, der sich über alle Spalten zieht (`span="all"`, [3]). Es existiert in XSL-FO auch kein Konzept für den Ausgleich von Spalten.

4.2.1.2 CSS [CSS3COL]

Mit den neuen CSS3COL-Features („Multi Column Layout“) kann jeder Block aufgespaltet werden. Ähnlich wie in FO heißen die CSS-Eigenschaften `column-count`, `column-gap` und `column-width` (Spaltenbreite, obligatorisch) (**[4]**). CSS beinhaltet mit der Spezifikation bereits ein Konzept für Trennlinien zwischen den Spalten (`column-rule`, **[5]**). Diese legen sich über den Spaltenabstand, dadurch wird die Spalte also nicht schmaler oder enger, ist aber auch immer nur mittig abgebildet. Die Trennlinien besitzen fast alle Werte, die auch `border` verwendet (Achtung: ohne `column-rule-style`-Angabe wird nichts angezeigt). [64]

Möchte man eine Überschrift o.Ä. über die gesamte Breite aller Spalten legen, so bekommt das gewünschte (Block-)Element ein `column-span="all"` (**[6]**), wie auch in XSL-FO. Die Eigenschaft trennt den Inhalt in einen „davor“- und „danach“-Block auf, sodass der Block vor der

[63] Montero/Krüger, S.199
[64] W3C: CSS3COL

Überschrift und auch danach jeweils in zwei Spalten dargestellt wird. Das ermöglicht es, lange Textabschnitte zusätzlich durch `column-span` in Zeilen zu unterteilen.[65]

CSS besitzt sogar ein Konzept für den Spaltenausgleich – die `column-fill`-Eigenschaft mit den Werten *`balance`* (verteilt den Inhalt gleich auf alle Spalten) oder *`auto`* (füllt die Spalten von rechts nach links mit Inhalt auf) (**[7]**). Das funktioniert allerdings laut W3C nur bei definierter Spaltenhöhe, die auch höher als die Höhe des darin befindlichen Inhalts sein muss. Außerdem wird angemerkt, dass dieser Effekt in seitenbasierten Medien nur auf der letzten Seite des Mehrspaltenelements auftritt.[66]

```
#register{
   column-count:2;                [4]
   column-gap: 4mm;
   column-rule-style:solid;       [5]
   column-rule-width:1pt;
   column-fill:balance;           [7]
#register h2{
   column-span:all;               [6]
}
```

Code 7: CSS Mehrspaltensatz; Renderergebnis s. Anhang: Abb.6

In der Aufführung zeigt sich auch auf der letzten Seiten ein verbesserter Spaltenausgleich (*s. Anhang: Abb. 7/8*).

4.2.1.3 Fazit Abschnitt

CSS hat scheinbar aus den fehlenden Eigenschaften für Mehrspaltigkeit in XSL-FO gelernt, denn es werden Eigenschaften für Spaltentrennlinien und den -ausgleich bereitgestellt. Die Umsetzung des Ausgleichsattributs ist zwar etwas schwach, in XSL-FO wird der Spaltenausgleich allerdings mit dem AH-Zusatz auch nicht unbedingt besser dargestellt. So liegt es entweder an der AH-Umsetzung oder an der Schwierigkeit im Allgemeinen, einen regelmäßigen und korrekten Spaltenausgleich zu erreichen.

Alles in Allem ist die CSS3COL-Spezifikation nützlich und benutzbar mit ausreichendem Ergebnis.

4.2.2 Umbrucheinstellungen

Seiten-, Spalten-, Bereichs- und Zeilenumbrüche sind wichtige Layouteinstellungen, die die Übersichtlichkeit erhöhen, Abschnitte durch Zusammengehörigkeit von Zeilen oder Absätzen kennzeichnen und letztendlich zur Verteilung des Inhalts über mehrere Seiten hinweg beitragen.

---

[65] Heller (2011)
[66] W3C: CSS3COL, #column-fill

Bei dynamischen Medien sind Seitenumbrüche (u.a.) nicht nötig, da der Inhalt dynamisch fließend z.B. auf einer Webseite dargestellt wird (dynamisch durch den Browser gerendert). Bei Printmedien allerdings existieren feste und starre Seiteneinteilungen. Herausfordernd werden die Umbrucheinstellungen dadurch, dass es Sonderfälle zu beachten gibt:

Eine Absatzkontrolle soll sogenannte „Hurenkinder“ und „Schusterjungen“ (oder auch: „Witwen“ und „Waisen“, aus dem Englischen) vermeiden. Schusterjungen sind ein oder zwei Zeilen, die am Ende einer Seite stehen und dadurch unschön aussehen und der Kontext des restlichen Absatzes fehlt – sie haben sozusagen keine Zukunft. Hurenkinder sind dann die wenigen Zeilen, die am Anfang einer Seite stehen und deren Kontext des zugehörigen Abschnittes aber noch auf der vorherigen Seite zu finden ist – diese Zeilen haben also keine „Vergangenheit“.

Weiterhin sollen manchmal Umbrüche erzwungen und manchmal auf jeden Fall vermieden werden. Die Silbentrennung zählt ebenfalls zu den Umbrüchen, die bestimmt werden müssen.

4.2.2.1 XSL-FO

XSL-FO bietet zur Umbruchsteuerung von Blöcken und Inzeiligem vier Typen von Umbrucheigenschaften:

1. Attribute zur Verhinderung von Schusterjungen und Hurenkindern (`orphans/widows`)
2. Umbruchverhindernde Attribute: `keep-together`, `keep-with-next`, `keep-with-previous`, mit Zusätzen: `.within-line`, `.within-column`, `.within-page`
3. Umbrucherzwingende Attribute: `break-after` und `break-before`
4. Alternative Attribute: `page-break-after`, `page-break-before`, `page-break-inside` (bezieht sich nur auf den Seitenumbruch; Umbruchverhinderung durch Wert *`none`*) [67]

Mit `orphans` oder `widows` und einem Zahlenwert lässt sich bestimmen, wieviele Zeilen denn mindestens zusammenstehen müssen, damit solche unästhetischen Kontextlosigkeiten der Schusterjungen und Hurenkinder vermieden werden können:

```
<fo:block orphans="3"> ...
```

Code 8: XSL-FO Schusterjungen vermeiden

Umbruchverhindernde Attribute legen fest, dass innerhalb von Elementen oder zwischen bestimmten Elementen kein Umbruch stattfinden soll, der die Zeile, Spalte oder Seite betrifft. So kann bspw. eine Überschrift mit dem folgenden Block zusammengehalten werden oder innerhalb eines Zitats ein Seiten- oder Spaltenumbruch vermieden werden. Für diese Umbruchattribute sind jeweils die Werte *`always`* und *`auto`* gültig – *`always`* erzwingt den Effekt und mit *`auto`* wird er

[67] Montero/Krüger, S.104

fakultativ. Setzt man einen Zahlenwert ein, so wird dieser kontextabhängig vom Formatierer gewertet und schwächt den Durchsetzungszwang der Eigenschaft ab:

```
<fo:block keep-together.within-page="always"> …
```

Code 9: XSL-FO Seitenumbruchverhinderung

Wie sich vermuten lässt, haben umbrucherzwingende Attribute die Folge, dass, je nach Wert, vor oder nach einem Block die Spalte (`column`), Seite (`page`), nach einer linken bzw. geradzahligen (`even-page`) oder einer rechten bzw. ungeradzahligen Seite (`odd-page`) umbrochen wird:

```
<fo:table break-after="page"> …
```

Code 10: XSL-FO Seitenumbruch erzwingen

Alternativ zu diesen Umbruchsteuerungen gibt es noch weitere Umbruchattribute die Seite betreffend. Mit jeweils einem `always`- oder `avoid`-Wert lassen sich somit ein Umbruchzwang oder eine Umbruchvermeidung vor, nach oder innerhalb eines Blocks herstellen. Mit den Werten `left` oder `right` lässt sich auch hier bestimmen, ob die dem Umbruch nachfolgende Seite eine linke/geradzahlige oder eine rechte/ungeradzahlige Seite sein soll:

```
<fo:block page-break-before="left"> …
```

Code 11: XSL-FO Seitenumbruch linke Seite

Die automatische Silbentrennung wird aktiviert, indem das Attribut `hyphenate` (Wert `true` aktiviert, `false` deaktiviert) formuliert wird. Anhand des `xml:lang`-Attributs des betreffenden Elements, werden Regeln oder Muster der Trennung angegeben. Die deutsche Silbentrennung entspricht dem Wert `de`. Weitere Modifikationen wie die Bestimmung des Trennzeichens, trennungsverhindernde Angaben oder andere Länder- bzw. Sprachcodes sind möglich. Die deklarierten Attribute (**[1]**) werden auf die untergeordneten Elemente vererbt.[68]

```
<xsl:template match="A">
  <fo:block text-align="justify" hyphenate="true" xml:lang="de">        [1]
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Code 12: XSL-FO Silbentrennung; Montero/Krüger, S.100

AHF nutzt bei der Silbentrennung ein Algorithmenprogramm namens *Hyphenologist* von *Computer Hyphenation Ltd.* Dieses Programm ist eine neue Adaption einer Sammlung von etablierten Wortaufteilungsalgorithmen (z.B. TeX Trennregeln fürs Deutsche[69]). Damit kann eine Silbentrennung in vielen menschlichen Sprachen stattfinden.[70]

---

[68] Montero/Krüger, S.100
[69] TUG: „TeX Hyphenation Patterns", #introduction
[70] Vgl. AH: „Description of Hyphenologist"

4.2.2.2 CSS [CSS3BREAK, CSS3PAGE, CSS3TEXT]

Das Modul CSS3BREAK[71] ermöglicht die gewünschten Umbrüche, die sich auch hier erzwingen oder verhindern lassen. Das CSS-Konzept stellt allerdings ein teilweise simpleres Konzept als das von XSL-FO zur Verfügung.

Zuerst auf die Schusterjungen und Hurenkinder eingehend lässt sich sagen, dass diese ebenfalls mit den Attributen `orphans` und `widows` vermieden werden können – die Wertangabe deckt sich mit der XSL-FO Variante.

```
*{
  orphans:2;
  widows:2;
}
```

Code 13: CSS Schusterjungen und Hurenkinder vermeiden

Die Umbrucheigenschaften `break-after`, `break-before` und `break-inside` können u.a. jeweils allgemein erzwungen (`always`), vermieden (`avoid`) oder fakultativ (`auto`) werden. Mit `(avoid-)page`, `(avoid-)region` oder `(avoid-)column` lässt sich genauer auf die Umbruchart eingehen, für `break-inside` allerdings nur die vermeidenden Werte. Außerdem gibt es für `break-after` und `-before` auch die Möglichkeit zu bestimmen, ob nach dem Umbruch eine linke oder rechte Seite stehen soll (`left`/`right` bzw. `verso`/`recto` bei anders gegenüberliegenden Seiten).

```
#register p{
  break-inside:avoid;
}
#register h2{
  …
  break-after:avoid;
}
```

Code 14: CSS Umbruchverhinderung

Die Seitenumbrüche lassen sich alternativ auch CSS2.1-konform mit `page-break-before` und `–after` ausdrücken, jeweils mit `auto`, `always`, `left`, `right` oder `avoid`:

```
body > section{
  page-break-before:right;
}
h2, h3, h4{
  page-break-after:avoid;
  page-break-inside:avoid;
}
```

Code 15: CSS Seitenumbruchsteuerung

[71] W3C: CSS3BREAK

Zeilen- und Wortumbrüche kann man durch `line-break` oder `word-break` erreichen (CSS3TEXT[72]). Wie streng auf Zeilenumbruchsregeln geachtet werden soll, bestimmen die Werte *`loose`* (geringe Einschränkungen), *`normal`* (gewöhnliche Regelbeachtung) und *`strict`* (strenge Regelung). Eine Anmerkung des W3C ist, dass in einer zukünftigen Modul-Version eine feinere Umbruchkontrolle bei Zeilen- und Wortumbrüchen notwendig sei, um wirklich professionelle Produkte umsetzen zu können.[73] Beim Wortumbruch wird geregelt, wie ohne die Anwendung der Silbentrennung zwischen Buchstaben eines Wortes getrennt werden darf. Bei *`break-all`* kann zwischen jedem Zeichen umbrochen werden, *`normal`* bestimmt die übliche Trennregelung und bei *`keep-all`* sollen die Zeichenpaare beieinander behalten werden.[74]

Wenn mit CSS die automatische Silbentrennung verwendet werden soll, muss für das entsprechende (HTML-)Element das `lang`-Attribut mit der angegebenen Sprache vorhanden sein. Dann lässt sich mit der CSS-Eigenschaft `hyphens` und dem Wert *`auto`* die automatische Silbentrennung anwenden. Weitere Angaben zur erweiterten Silbentrennung gibt es nicht. Die Umsetzung der Silbentrennung ist nicht perfekt, manche Worte würde ein menschlicher Sätzer anders trennen.

```
*{
   hyphens:auto;
}
```

Code 16: CSS Silbentrennung

Mit CSS3GCPM werden auch Umbrucheinstellungen für Fußnoten definiert, falls die komplexe Situation auftritt, dass Verweiszeichen und Fußnotentext nicht auf derselben Seite angezeigt werden können, weil für den gesamten Fußnotentext nicht ausreichend Platz ist. Mit `footnote-policy` (deklariert für das Element, in dem das Fußnotenelement auftritt, nicht für das eigentliche Fußnotenelement) lässt sich ein Seitenumbruch am Zeilen- bzw. Blockanfang des Verweiszeichens bestimmen (Werte: *`line`* bzw. *`block`*).[75]

4.2.2.3 Fazit Abschnitt

Die Umbruchsteuerung in CSS ist begrifflich ähnlich zu XSL-FO, auch wenn es weniger komplex angelegt wurde, und funktioniert mit AHF sehr gut. Auch XSL-FOs `keep-together.within-[line|column|page]` lässt sich mit CSS umsetzen, durch die *`avoid`*-Werte der Attribute `break-after/-before/-inside`.

---

[72] W3C: CSS3TEXT
[73] Ebd., #line-break-property
[74] Ebd., #word-break-property
[75] Götz/Ott, S.90

Schusterjungen und Hurenkinder werden auch nicht immer wie gewünscht vermieden, was aber eine Schwäche von AHF ist und wahrscheinlich in Konflikten mit anderen Umsetzungen von Eigenschaften, die Priorität besitzen, begründet ist.

Auch die Silbentrennung lässt noch Verbesserungen zu. Auch wenn diese Trennung (wie alle anderen) vom Renderer abhängt, ist es nicht unbedingt ein Implementierungsfehler von AHF, dass Silben nicht immer korrekt umbrochen werden. Eher ist es darauf zurückzuführen, dass in der computerlinguistischen, syntaktischen Forschung noch keine Antwort auf die Frage nach einer fehlerfreien automatischen Silbentrennung gefunden wurde.

4.2.3 Seitenvorlagen

Neben der Einstellung einer alle Seiten umfassenden, globalen Default-Seitenformatvorlage können noch weitere „Musterseiten“ (bzw. „Seitentypen“) erstellt werden, die bestimmte Teile des Dokumentinhalts auf ein anderes Layout übertragen. So kann ein Doppelseitenlayout mit rechter und linker Seite entstehen und Inhaltsverzeichnis, Inhalt und Register können jeweils selbstdefinierte Einstellungen für das Format erhalten. Durch Kombinationen von Seitenumbrüchen und Doppelseitenlayout können auch Vakatseiten (leerbleibende Füllseiten) entstehen.

4.2.3.1 XSL-FO

Auch hier werden in XSL-FO wieder Seitenformatvorlagen verwendet, die dann in eine Abfolge gebracht und mit dem Inhalt verknüpft werden. Je nachdem, ob man eine Titelseite, ein Doppelseitenlayout oder eine andere Musterseite erstellen möchte, modifiziert man entweder die Seitenformatvorlagen, die Folgeneinstellungen oder die Verknüpfung mit dem Inhalt.

Für ein Doppelseitenlayout (*s. Anhang: Code 17*) werden zu den Layoutmasterseiten (**`<fo:layout-master-set>`**) jeweils für die linke und für die rechte Seite eines Buches (o.ä.) eine Masterseite mit passendem Format hinzugefügt (**`<fo:simple-page-master>`**). Daraufhin wird die Reihenfolge des Auftretens der Layouteinstellungen für die Inhaltsseiten im Dokument aufgestellt (**`<fo:page-sequence-master>`**): die Masterseiten werden abwechselnd aufgerufen und das Vorkommen so bedingt, dass die rechte auf einer ungeraden Seite liegen soll und die linke Seite auf einer geraden. Die Seitenzahl bestimmt die Geradheit, dazu aber mehr im Kapitel „Marginalien“. Damit nun der gesamte Inhalt der XML-Instanz auch auf dieses Doppelseitenformat gebracht wird, fügt man in der fließenden Inhaltsregion der referenzierten **`<fo:page-sequence>`** den Aufruf aller weiteren Elemente ein.[76]

[76] Montero/Krüger, S.191ff

In Kombination dieser Doppelseiten mit den vorgestellten Umbrüchen, die das Element danach z.B. auf einer rechten Seiten beginnen lassen (u.a. `page-break-after="right"` oder `break-after="odd-page"`), kann automatisch eine sogenannte Vakatseite entstehen. Ist die eigentlich vorgesehene Seite nach dem Umbruch nämlich eine linke Seite, wird eine leere Seite eingefügt, damit der weitere Inhalt auf der rechten Seite fortgeführt werden kann.

Bei einer Titelseite wird ebenfalls eine entsprechende Masterseite erstellt und diese in der Seitenfolge zu Beginn aufgerufen. Weitere Musterseiten, wie für Inhaltsverzeichnisse o.A., können auch als entsprechende Masterseite (natürlich mit anderem `master-name`) umgesetzt werden. Diese müssen dann das passende Element des Inhaltsverzeichnisses in der Seitenfolge **`<fo:page-sequence>`** aufrufen.

Um die Vakatseiten zu modifizieren, kann man in der entsprechenden Seitenfolge mit dem Attribut `blank-or-not-blank="blank"` leere Seiten abfangen und formatieren (**[6]**).[77]

```
<fo:page-sequence-master master-name="Inhalt-Seiten">
  <fo:repeatable-page-master-alternatives>
        ...
        <fo:conditional-page-master-reference
        master-reference="PageMaster.Teil" blank-or-not-blank="blank"/>        [6]
...
```

Code 18: XSL-FO Vakatseite; angelehnt an Montero/Krüger (2004), S.73f

4.2.3.2 CSS [CSS3PAGE, CSS3GCPM]

Für die Erstellung von Titelseiten, Doppellayouts und anderen Musterseiten mit CSS spielen Seitenselektoren bzw. Seiten-Pseudoklassen-Selektoren eine wichtige Rolle. Durch Pseudoklassen wie `:first`, `:left`, `:right` etc. lässt sich u.a. der allgemeine **`@page`**-Befehl zielgerichteter ausführen. Bestimmte Seitentypen eines Dokuments lassen sich damit und mit selbst-definierten Musterseiten modifizieren, sodass komplexe Layouts gestaltet werden können (z.B. wenn für die erste Seite eines Unterabschnitts ein anderes Layout gefragt ist).[78]

Neben der Default-Seitenformatierung (**`@page`**) lassen sich hier auch weitere Musterseiten-Gruppen anlegen, die sich auf bestimmte Bereiche des Dokuments beziehen. Das erreicht man mit CSS über „benannte Seiten“ („named pages“), also **`@page`**-Regeln mit einem Namenszusatz (**`@page name`**, **[7]**), die mit Elementen im Deklarationsblock verknüpft sind (**[8]**).[79]

---

[77] Montero/Krüger, S.73f
[78] W3C: CSS3PAGE, #page-selector-and-context
[79] Ebd., #using-named-pages

```
@page first{                    [7]
   margin: 4cm 4cm 4cm 4cm;
   border:solid 1pt;
}
#first{
   page:first;                  [8]
}
```
Code 19: CSS Seitenvorlage named pages

Möchte man ein Doppelseitenlayout erstellen, benutzt man die Pseudoklassen `:left` und `:right` – wirksam bei **`@page`** und eigenen benannten Seitenregeln. Damit kann man Unterschiede in der Layoutierung zwischen den Seiten festlegen (z.B. jeweils Bindekante/Innenkante breiter). Ob die rechte oder linke Seite gerade bzw. ungerade ist, bestimmt im Grunde der Formatierer und seine Defaulteinstellungen, sollte man nichts Anderes durch die Paginierung (s. *4.5.2.3 Pagina*) bestimmt haben. Bei AHF ist es aber so, dass die Seitenzählung auf einer linken Seite anfängt und somit rechts gerade und links ungerade ist.

```
@page:left{
   padding-left: 1cm;
}
@page:right{
   padding-right: 1cm;
}
```
Code 20: CSS Seitenvorlage Doppelseite

CSS erstellt ebenfalls automatisch eine Vakatseite bei Anwendung einer Umbruchregel mit dem Wert `left` oder `right` (bzw. `verso` und `recto`). Mit der Pseudoklasse `:blank` kann man auch diese Leerseite gestalten, indem man z.B. alle Randinhalte (u.a. Kopf- und Fußzeilen) entfernt oder einen ausgewählten Text einfügt (bspw. „This page was intentionally left blank“). Leider müssen für jede angelegte Musterseite die `:blank` Werte eigens bestimmt werden, denn da diese Seitenregeln spezieller sind als die allgemeine **`@page`**-Regel, gilt die **`@page`**`:blank`-Regel auch nur für die Seiten, die keine eigene Musterseitenlayoutierung haben.[80]

```
@page first:blank{
   border:none;
}
```
Code 21: CSS Vakatseite

Formateigenschaften für eine Titelseite bzw. erste Seite eines Abschnitts lassen sich mit `:first` festlegen. Eine allgemeine Titelseite lässt sich mit **`@page`**`:first` gestalten, nicht zu verwechseln mit der **`first`** genannten Seitevorlage (**`@page first`**). Für eigene Musterseiten lässt sich die jeweilige erste Seite auch mit `:first` formatieren.

[80] W3C: CSS3PAGE, #blank-pseudo

```
@page ivz:first{
   background-color:lightgrey;
}
```
Code 22: CSS erste Seite

Die Pseudoklassenselektoren lassen sich auch kombinieren. So kann z.B. die erste linke Seite durch `@page:left:first` angesprochen werden.

4.2.2.1 Fazit Abschnitt

Mit CSS kann man unterschiedliche Seitenformatvorlagen erstellen, wobei man allerdings merkt, dass dies nicht von Anfang an im Konzept vorgesehen war. Der CSS-Code für komplexe Dokumentformatierungen kann durch die vielen Selektoren auch recht unübersichtlich werden. Und doch ist er noch einfacher zu formulieren und zu verstehen, als XSL-FO-Masterseiten und deren Einpflege.

Etwas, das CSS aber benötigen würde und was XSL-FO im Konzept verankert hat, ist die Seitenfolgenmodellierung. Welche Seitenvorlage nach welcher Seitenvorlage kommt, das kann mit CSS nicht extra bestimmt werden. Die Reihenfolge der auftretenden Musterseiten geschieht nach der Reihenfolge, in der die jeweiligen Elemente im Dokument erscheinen. XSL-FO greift hierbei, anders als CSS, mehr in die Struktur ein und geht über die ausschließliche Seitenlayoutgestaltung hinaus, indem sie unabhängig von der originalen Reihenfolge agiert.

4.2.4 Selektoren

Um besser im Dokumentstrukturbaum oder durch die Seiten navigieren und genau die erwünschte Seite oder das entsprechende Element auswählen zu können, stellt CSS3SELECT neue Pseudoklassen und –elemente zur Verfügung.[81] Dieser Abschnitt erläutert nur die Neuheiten der CSS3 Version, da XSL-FO zum zielgerichteten Selektieren auf XPath zurückgreift.

Bei der HTML5-Erstellung mit XSLT wird auch mit XPath selektiert, aber für die Gestaltung des Dokuments mit CSS werden darüber hinaus oft auch Pseudoklassen und -elemente benötigt.

Pseudoklassen (beginnend mit „**:**“) ermöglichen eine zielgerichtetere Selektion der Elemente. Davon gibt es einige, die für Webanwendungen gedacht sind. Uns interessieren aber hauptsächlich die strukturellen Pseudoklassen, womit sich Elemente direkt anhand ihrer Position ansprechen lassen. Pseudoelemente („**::**“) stehen für nicht vorhandene Elemente.

Z.B. folgende neue Pseudoklassen sind nützlich:[82]

[81] W3C: CSS3SELECT
[82] Ebd.

- Mit `:nth-child()` können die ungeradzahligen (`odd`), geradzahligen (`even`) oder mit einem Zahlenwert übereinstimmenden Kindelemente eines Elements selektiert werden (z.B. `#ivz p:nth-child(4)` – das vierte Kindelement von `section` mit `id="ivz"`).
- Mit `:nth-of-type()` können die ungeradzahligen (`odd`), geradzahligen (`even`) oder mit einem Zahlenwert übereinstimmende Kindelemente eines bestimmten Typs eines Elements selektiert werden (z.B. `#welcome p:nth-of-type(2n)` – jedes zweite `p`-Element in `section` mit `id="welcome"`).
- Mit `:nth-last-child()/-of-type()` gelten dieselben Werte wie bei den Varianten ohne „last“, die Zählung geschieht aber rückwärts und beginnt beim letzten passenden Element (`#ivz :nth-last-child(4)` – das viertletzte Kindelement von `section` mit `id="ivz"`).
- Mit `:[first|last]-child` wird das erste oder letzte Kindelement eines Elements selektiert (analog dazu `:[first|last]-of-type`, z.B. `#ivz tr:last-of-type` – das letzte `tr`-Element in `section` mit `id="ivz"`).
- Mit `:only-child/of-type` werden alleinstehende Kindelemente bzw. einzige Kindelemente eines bestimmten Typs eines Elements selektiert (z.B. `h4:only-of-type` – alle `h4`-Elemente, die unter ihren Geschwisterelementen die einzigen ihres Typs sind).
- Mit `:not()` werden die Elemente, die einem Selektor nicht entsprechen, selektiert (z.B. `#welcome :not(p)` – alle Nachfahrenelemente von `#welcome`, die nicht `p` sind).

Laut „CSS-Conformance“ von AHF sind all diese Selektoren in Version 6.4 implementiert.[83]
In der Praxis gibt es allerdings beim `:not()`-Selektor Schwierigkeiten, denn dieser darf nur einen simplen Selektor als Argument beinhalten, d.h. ein Ausdruck wie `:not(p:first-of-type)` funktioniert nicht. Die anderen Selektoren wirken korrekt.

Für die Seitenselektion stehen momentan nur `:nth-child(even|odd)` und `:nth-of-type(even|odd)` zur Verfügung, also jede gerad- oder ungeradzeilige Seite bzw. Seite eines bestimmten Typs, die anderen Werte werden nicht formatiert.

```
@page ivz:nth-child(odd){
  background-color:aliceblue;
}
```

Code 23: CSS Seitenselektion :nth-child()

4.3 Inhaltstext

Nachdem die Thematik der Seitenformatierung abgehandelt wurde, wird nun der eigentliche Inhaltstext in Form gebracht. Mit Inhaltstext sind alle Boxelemente gemeint, die den Haupttext

[83] AH: „XSL/CSS Extensions“, #css3

beinhalten und auf den Inhaltsbereich der Seiten verteilt werden. Im Folgenden wird die Absatz- und Zeilenformatierung behandelt sowie die Gestaltung von Text und Wort.

4.3.1 Absatz- und Zeilenformatierung – Blöcke, Inline und Inline-Blöcke

Ein Blockelement bildet Absätze, die in der Regel auf einer neuen Zeile beginnen und wonach ein Zeilenumbruch folgt. Inzeilige Elemente werden, wie der Name schon sagt, in einer Zeile verortet und folgen dieser fortlaufend, am Ende des bereitstehenden Zeilenraums brechen sie um.

Wenn man mit solchen Elementen auf einer geometrisch festgelegten Seite arbeitet, gilt es, vom Umbruchkontext abhängige Abstände zu beachten. Beginnt ein neuer Absatz z.B. direkt nach einem Seitenumbruch, dann soll dort kein Abstand gebildet werden, an anderen Stellen jedoch schon.

Welche Möglichkeiten zur Zeilenausrichtung existieren, Schreibmodus, Textorientierung und auch die Umsetzung hängender Zeileneinzüge, d.h. alle Zeilen eines Absatzes bis auf die erste Zeile werden eingerückt, werden betrachtet.

Sollen Abschnitte zusätzlich noch durch horizontale Linien getrennt werden, um z.B. Kapitelwechsel oder inhaltliche Unterschiede kenntlich zu machen, so müssen diese Trennlinien auch umgesetzt werden.

4.3.1.1 XSL-FO

Absatzformatierung mit XSL-FO geschieht nach dem Prinzip, dass der gewünschte Inhalt mit einem Blockelement (***<fo:block>***) umschlossen und mit Attributen angepasst wird.

Ein Inlineobjekt (***<fo:inline>***) unterscheidet sich auch in XSL-FO hauptsächlich von einem Block, indem keine Absätze gebildet werden und die Inhalte inzeilig eingefügt werden. Außer dem margin-Attribut gelten annähernd alle Attribute von ***<fo:block>*** auch für ***<fo:inline>***. Ein Inlineelement wird typischerweise dazu verwendet, einzelne Worte, Wortsequenzen oder Zeichen hervorzuheben (z.B. fett, kursiv oder unterstrichen).[84]

In FO gibt es zwei Arten von Abständen: `margin` und `space`. Soll ein Abstand, z.B. vor einer Überschrift, am Anfang einer Seite oder Spalte nicht dargestellt werden, so muss `space` verwendet werden (**[1]**). `Space` hängt nämlich, anders als `margin`, davon ab, ob vorher (oder nachher) ein Seiten- oder Spaltenumbruch steht oder nicht. `Space` sorgt ebenfalls dafür, dass Abstände zwischen zwei Elementen nicht einfach addiert werden, wenn das vorherige einen

[84] Montero/Krüger, S.94

Abstand danach und das folgende einen Abstand davor aufweist – mit `space` lassen sich diese Abstände vielfältiger und genauer kontrollieren.[85]

```
<fo:block space-before="20mm" padding="5mm 5mm 5mm 5mm"> [1] ...
```

Code 24: XSL-FO space

Zeilenabstände werden mit der Zeilenhöhe bestimmt. Diese wird normalerweise durch die Schriftgröße berechnet. Wird aber nun eine Zeilenhöhe mit `line-height` (z.B. `line-height="1.5em"`) festgelegt, kann somit auch der Zeilenabstand reguliert werden.

Um Absätze kenntlicher zu machen, setzt man in der Typografie gerne Einzüge ein. Mit XSL-FO gibt es Absatzeinrückungsattribute, die alle Zeilen des formatierten Blocks im Start- bzw. Endbereich einrücken (`start-/end-indent`). Zeilen werden durch `text-indent` eingerückt, damit wird die erste Zeile im Startbereich beeinflusst. Mit negativen Werten können Absätze und Zeilen auch in die andere Richtung (nach links) ausgerückt werden. In der Typografie gibt es zwei Arten, einen Absatzeinzug zu gestalten: entweder wird nur die erste Zeile eingerückt (**[2]**) oder es gibt einen sogenannten „hängenden“ Einzug (**[3]**), bei dem alle Zeilen, bis auf die erste, eingerückt werden. Letzteres setzt FO so um, dass der gesamte Block einen Einzug erhält und dann nur die erste Zeile durch einen negativen `text-indent`-Wert wieder ausgerückt wird. Ein ausdrücklicher „hanging-indent“-Befehl o.ä. gibt es im Konzept also nicht.[86]

```
<section>
  <fo:block space-before="5mm" text-align="justify" text-indent="10mm">      [2]
    <xsl:apply-templates/>
  </fo:block>
</section>
...
<section>
  <fo:block text-align="justify" start-indent="10mm" end-indent="10mm" text-
indent="-10mm">                                                            [3]
    <xsl:apply-templates/>
  </fo:block>
</section>
```

Code 25: XSL-FO Einzüge; Montero/Krüger, S.97

Für horizontale Linien innerhalb von Blöcken oder Zeilen in XSL-FO gibt es das Element **`<fo:leader>`** ([4]). Zur genauen Beschreibung dieser Trennlinie können z.B. die Länge, das Zeichenmuster der Linie, Farbe und Dicke festgelegt werden.[87]

[85] Montero/Krüger, S.91
[86] Ebd., S.97
[87] Ebd., S.101f

```
<fo:block>
  <fo:leader leader-pattern="rule" leader-length="70%" rule-style="dotted" rule-
thickness="5pt"/>                    [4]
  …
</fo:block>
```

Code 26: XSL-FO horizontale Linie; Ausschnitt aus Montero/Krüger, S.102

4.3.1.2 CSS [CSS3TEXT, CSS3GCPM]

Das Block- und Inline-Konzept in CSS ist an sich dasselbe wie in XSL-FO, es existiert allerdings auch ein Inline-Block, der einerseits inzeilig positioniert wird, sich aber innerhalb wie ein Block verhält (z.B. Zusammenhalt des Inhalts). Möchte man ein Element als Block kennzeichnen, nutzt man die `display`-Eigenschaft. Damit lässt sich die Anzeigeart des Elements bestimmen (u.a. *`block`*, *`inline`*, *`inline-block`*, *`list`*, *`table`* etc.).

Die Arbeit mit HTML und einem Renderer wie AHF hat hier auch wieder einen positiven Effekt, denn AHF setzt HTML-Elemente schon automatisch als Blöcke, Listen, Tabellen etc. um, so kann man sich oft Zeit sparen, kann aber auch eigene Einstellungen verwenden, falls erwünscht.

Zur Formatierung der Zeilenausrichtung gibt es die neuen `text-align`-Werte *`start`* und *`end`* (eigentlich auch *`match parent`* und *`start end`*, welche aber nicht in AHF implementiert sind),[88] außerdem wurde die `text-align-last`-Eigenschaft eingeführt, um die letzte Zeile/das letzte Element eines Absatzes zu alinieren (**[5]**). Dabei kommen dieselben Werte wie bei `text-align` zum Einsatz. Bei *`start`* richtet sich der inzeilige Inhalt an der Startkante der Zeilenbox aus, bei *`end`* an der Endkante (**[6]**).[89]

```
#welcome p{
   display: block;
   …
   text-align-last:end;     [5]
   /*text-align:end;*/      [6]
}
```

Code 27: CSS Zeilenausrichtung text-align-last; Renderergebnis und Vergleich s. Anhang: Abb. 9

Wenn der Text als Blockschrift (`text-align:`*`justify`*) formatiert wird, kann man mit `text-justify` (Werte: *`auto`*, *`none`*, *`inter-word`*, *`distribute`*) weitere Einstellungen wie Alinierung und Abstände bestimmen. Die Worttrennzeichen wie Leerzeichen oder Tabulator etc. werden mit *`inter-word`* bei der Blocksatzausrichtung beeinflusst (effektive `word-spacing`-Variation), das ist typisch für Deutsch oder Englisch, da diese Trennzeichen zur Wortunterscheidung benutzt

[88] W3C: CSS3TEXT, #text-align-property
[89] Ebd., #text-align-last

werden. Mit `distribute` wird der Abstand zwischen benachbarten Schriftzeichen angepasst (manchmal im Japanischen nützlich; effektive `letter-spacing`-Variation).[90]

In CSS ist der Zeilenabstand genauso wie in FO über das Zeilenhöhen-Attribut (`line-height`, **[7]**) veränderbar. Absatzabstände (von Blöcken oder inzeiligen Blöcken) lassen sich z.B. aber auch durch `margin-top/-bottom` vergrößern oder verkleinern (**[8]**). Folgt bspw. ein Element mit `margin-top` auf eins mit `margin-bottom`, so fallen diese Abstände zusammen zum Wert der größeren Abstands.[91]

```
#welcome p{
   …
   line-height:1.5em;                [7]
}
…
h1{
   display: block;
   margin-top: 26pt;
   margin-bottom: 13pt;              [8]
}
p{
   display:inline-block;
   margin-top:2mm;                   [8]
   margin-bottom:2mm;
}
```

Code 28: CSS margin und line-height

Während in XSL-FO das `space`-Attribut existiert, gibt es in CSS auch für abhängige Abstände nur `margin`. Allerdings bietet AH hier eine Erweiterung an, die `margins` nach Seiten- oder Spaltenumbrüchen nicht darstellt (`-ah-margin-break: discard`, ist automatisch in AH als Defaultwert hinterlegt, **[9]**). Möchte man sich den Abstand trotzdem anzeigen lassen, so setzt man den Wert auto (**[10]**). Außerdem könnte man statt der normalen `margin`-Eigenschaften die jeweilige AHF-Erweiterung einsetzen. So können `space`-ähnliche Abstände mit `-ah-margin-after/-before/-start/-end` entstehen.[92]

```
*{
   …
   -ah-margin-break: discard;        [9]
}
h1{
   …
   -ah-margin-break: auto;           [10]
}
```

Code 29: CSS umbruchabhängiger Abstand

[90] W3C: CSS3TEXT, #text-justify-property
[91] W3Schools: „CSS margins"
[92] AH: „XSL/CSS Extensions", #css3

Einzüge gestalten sich ebenfalls ähnlich zur XSL-FO-Formatierung. Mit `text-indent` lässt sich die erste Zeile eines Blocks (**[11]**) und mit `margin-left` der ganze Block einrücken. Start- und Endbereich sind nicht differenziert. Auch hier muss bei einem hängenden Einzug zuerst der ganze Block mit `margin` eingezogen werden und dann die erste Zeile ausgerückt, durch einen negativen `text-indent`-Wert (**[12]**).

Im CSS-Konzept gibt es allerdings bereits die Idee von Wertzusätzen für `text-indent`, die entweder hängende Einzüge gewährleisten sollen (*`hanging`*) oder auch jede Zeile eines neuen Absatzes einziehen (*`each-line`*) – diese gelten allerdings noch als inoffiziell und experimentell und sind im AHF noch nicht implementiert.[93]

```
#welcome p{
   display: block;
   /* text-indent:5mm; */   [11]
   text-indent: -5mm;       [12]
   margin-left: 5mm;        [12]
   …
}
```

Code 30: CSS Einzüge

Abb. 10: AHF Renderergebnis Einzüge

Hängende Interpunktion, d.h. Satzzeichen, die aus der Zeilenbox herausragen, können mit der Eigenschaft `hanging-punctuation` (Werte: *`none`*, *`first`*, *`last`*, *`force-end`*, *`allow-end`*) deklariert werden (*s. Anhang: Abb. 11*). Mit *`first`* werden Anführungszeichen oder sich öffnende Klammern am Anfang der ersten Zeile eines Blocks ausgerückt, mit *`last`* die schließenden Anführungszeichen oder Klammern der letzten Zeile. Punkte, Trennstriche und Kommas am Ende einer Zeile werden mit *`force-end`* zum „hängen“ gebracht (**[13]**), ebenfalls mit *`allow-end`*, wenn die Zeichen sonst nicht in den Blocksatz passen.[94]

```
#welcome p{
   …
   hanging-punctuation:force-end;  [13]
}
```

Code 31: CSS hanging punctuation; Renderergebnis und Vergleich s. Anhang: Abb. 11

Auch mit CSS ist es möglich, horizontale Trennlinien zu erstellen. Die *`leader()`*-Funktion (CSS3GCPM) lässt sich als Wert zum content-Attribut einsetzen und erstellt dadurch einen String/eine Zeichenkette mit sich wiederholendem Zeichenmuster. Wenn man z.B. nach jedem `article` einen Trennstrich möchte, so wählt man mit dem Pseudoelement-Selektor *`::after`* den

[93] W3C: CSS3TEXT, #text-indent-property
[94] Ebd., #hanging-punctuation-property

gewünschten Bereich aus (**[14]**), setzt die horizontale Linie ein und bestimmt deren Eigenschaften. Der Stil der Linie lässt sich innerhalb der Funktion bestimmen (**[15]**), Farbe, Breite oder anderes kann man durch die Manipulation des Aussehens des selektierten Pseudoelements erreichen (**[16]**).[95]

Leider ist diese Linie immer linksbündig ausgerichtet, was dann unästhetisch aussehen kann, sobald sie hinter Text in dieselbe Zeile gehängt wird. Denn dann schließt sie links nicht bündig mit der Kante des folgenden Elements ab. Somit werden unterschiedlich große Leerräume am Ende der Linie geschaffen.

```
article::after{                [14]
   content:leader(solid);      [15]
   color:gray;                 [16]
}
```

Code 32: CSS horizontale Linie; Renderergebnis s. Anhang: Abb.12

4.3.1.3 Fazit Abschnitt

Ein vom Seiten- oder Spaltenanfang abhängiges Abstandsattribut, wie bei XSL-FO das `space`-Attribut, ist im CSS-Konzept nicht vorhanden. Aber AHF reagierte mit einer implementierten Erweiterung – andere Formatierer müssten dies allerdings auch berücksichtigen, wenn sie CSS unterstützen möchten.

CSS ist noch in der Entwicklung, versucht aber, anwendungsspezifische und nützliche Eigenschaften und Funktionen zu verbessern bzw. zu erweitern. Das sieht man bspw. daran, dass bereits für Einzugsvarianten neue, experimentelle `text-indent`-Werte entwickelt werden.

Die Gestaltung der horizontalen Hilfslinie könnte allerdings noch ausgebaut werden, indem z.B. noch weitere Argumente zur `leader()`-Funktion hinzugefügt werden, u.a. für die Festlegung des Endpunktes der Linie.

4.3.2 Wort- und Zeichenformatierung

Worte und Zeichen eines Dokuments lassen sich vielseitig hervorheben und gestalten. Das Aussehen der Schrift, Textdekorationen wie z.B. Unterstreichungen, Hoch- oder Tiefstellungen und vieles mehr können die Darstellung der Daten verfeinern. Es gilt zu erkunden, wie solche Attribute eingesetzt werden können. Genauso stehen Defaulteinstellungen und weitere Anpassungen des Schriftbilds zur Betrachtung.

[95] W3C: CSS3GCPM, #leaders

4.3.2.1 XSL-FO

Neben der bereits angewandten Art und Weise, Elemente mit Eigenschaften zu bereichern, nämlich indem man sie als Attribute in dem jeweiligen Block- oder oder Inlineelement einbaut, gibt es noch eine weitere Methode, Eigenschaften zu deklarieren, die sich aus der Kombination von FO mit XSLT ergibt. Mit XSL-Attributmengen (**`<xsl:attribute-set>`**, **[1]**) lassen sich mehrere zusammenhängende Attribute global deklarieren und dann an unterschiedlichen Ort einsetzen (**[2]**). Wie Variablen kann man so zentral Attributsets für z.B. Überschriften festlegen und viele Elemente mit diesen verknüpfen. Sollten alle Überschriften plötzlich doch eine andere Farbe oder Schriftgröße erhalten, so kann man das in der passenden Attributmengenvariable ändern, wodurch sich diese Änderung auf alle verknüpften Überschriftenelemente auswirkt.[96]

Auch Standardwerte lassen sich in einer Attributsammlung für das Wurzelelement ausdrücken und werden überschrieben, sobald für eine genauere Elementauswahl andere Werte deklariert werden.

```
<xsl:attribute-set name="Ueberschrift-Titel1">                    [1]
  <xsl:attribute name="font-family">Arial Unicode MS</xsl:attribute>
  <xsl:attribute name="font-size">18pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-after">99pt</xsl:attribute>
  <xsl:attribute name="keep-with-next">always</xsl:attribute>
</xsl:attribute-set>
…
<fo:block xsl:use-attribute-sets="Ueberschrift-Titel1”>           [2]
…
</fo:block>
```

Code 33: XSL-FO Attributsets; Montero/Krüger, S.143

Um den Text nun zu modifizieren, stehen viele bekannte Attribute zur Verfügung. Von der Schriftfamilie und -größe (`font-family/-size`) über Wort- bzw. Zeichenabstände (`word-/letter-spacing`), Textausrichtungen (`reference-orientation`, `writing-mode`) und Textdekorationen wie Unterstrichen o.Ä. (`text-decoration`), Textschatten (`text-shadow`) oder Hoch- und Tiefstellungen (`baseline-shift`) zu u.a. linksbündigen Alinierungen (`text-align`) – alle Attribute aufzuzählen und einzeln zu betrachten, würde den Rahmen der Arbeit sprengen.

4.3.2.2 CSS [CSS3FONT, CSS3TEXT, CSS3TDECO]

Auch CSS bietet eine Vielzahl an Gestaltungsmöglichkeiten in Form von Attributen an, die in Namen, Werten und Funktion meist deckungsgleich zu den XSL-FO-Eigenschaften sind.

Mit CSS3 werden manche aus vorhergehenden CSS-Versionen bereits bekannte Attribute um Werte und Funktionen erweitert, manche Eigenschaften werden zur verbesserten Printgestaltung

[96] Montero/Krüger, S.143f

eingefügt. Das Schriftbild kann mit `font-kerning`, mehreren `font-variant`-Einstellungen, `font-stretch`, `text-decoration` und `text-shadow` sowie `writing-mode` und `text-orientation` gestaltet werden, außerdem kann eine externe Schriftart eingepflegt werden mit ***`@font-face`***.

Mit *`font-kerning`* (Werte: *`auto`*, *`normal`*, *`none`*) wird die kontextuelle Anpassung des Glyphen-Zwischenabstandes (Glyph ≙ Symbol) geregelt, falls die verwendete Schriftart Kerningdaten, d.h. Anpassungsmuster, umfasst. So können die Abstände zwischen ähnlichen Zeichen verbessert werden, was für Laienaugen aber nur schwer zu erkennen ist, wie man am Beispiel ‚AV' und ‚Ta' sehen kann (s. *Abb. 13*). Der Abstand zwischen den beiden Buchstaben wird jeweils ein wenig verringert, da die Formen der Glyphen es zulassen. Diese Kerningwerte sollten vom Formatierer noch vor den `letter-spacing`-Einstellungen verarbeitet werden.[97] Allerdings gibt AHF eine Fehlermeldung zurück, dass `font-kerning` eine unbekannte Eigenschaft sei.

U.a. zur Erstellung von Ligaturen (Kombinationen von Glyphen zu harmonischeren Formen) und Kapitälchen (Kleinbuchstaben in Form von Großbuchstaben) können `font-variant-ligatures` und `font-variant-caps` bzw. die Abkürzung `font-variant` benutzt werden.[98]

```
#welcome p{
   ...
   font-kerning: normal;
}

h4{
   ...
   font-variant:small-caps;
}
```

Code 34: CSS Kerning und Kapitälchen

Abb. 13: Kerning (l: auto, r: normal)[99]

Abb. 14: AHF Renderergebnis Kapitälchen

Die Eigenschaft `font-stretch` (Werte: *`[extra-]expanded|[extra-]condensed|normal`*) bewirkt, dass die Buchstaben in die Breite gezogen oder zusammengestaucht werden, während die Zeilenhöhe gleichbleibt (**[3]**).[100] Eine Transformation des Textes mit einer gleichmäßigen Erweiterung der Zeichenabstände (Schriftlaufweite[101]) lässt sich mit `text-tranform:`*`full-`*

[97] W3C: CSS3FONTS, #font-kerning-prop
[98] Mehr siehe Götz/Ott, S.52ff
[99] Quelle Abb. 13: https://developer.mozilla.org/en-US/docs/Web/CSS/font-kerning
[100] W3C: CSS3FONTS, #font-stretch-prop
[101] Beinert (2017): „Schriftlaufweite"

`width` erzwingen (**[4]**, auch „sperren“ genannt[102]). Damit können lateinische Schriftzeichen auch mit Zeichen ostasiatischer Sprachen (z.B. Chinesisch oder Japanisch) interagieren.[103]

```
h1{
   …
   font-stretch:extra-expanded; [3]
}
```

Code 35: CSS font-stretch

Abb. 15: AHF font-stretch-Werte (v.o.n.u.) extra-condensed; normal; extra-expanded

```
h3{
   …
   text-transform: full-width;   [4]
}
```

Code 36: CSS text-transform

Abb. 16: AHF text-transform-Werte (o.) normal, (u.) full-width

Text kann ebenfalls Schattierungen und Dekorationslinien erhalten. Textschatten wurden in CSS2 eingeführt, in CSS2.1 verworfen und in CSS3 wieder aufgegriffen. Nun können sie mit `text-shadow` deklariert werden, mit Schatteneffekten wie einem Farbwert und der Ausdehnung des Schattens (**[5]**). Diese Eigenschaft lässt sich auch für die Pseudoelemente `::first-line` und `::first-letter` anwenden und somit z.B. Initialen gestalten (**[6]**). In vorherigen CSS-Versionen war es möglich, einen Text zu unter-, über- oder durchstreichen (`under-, overline, line-trough`). In CSS3 lassen sich mit der `text-decoration`-Eigenschaft nun auch die Farbe (`-color`), Stil (`-style`), horizontale Position (`-line`) und die Durchgängigkeit (`text-decoration-skip`) der Linie gestalten. Mit der Abkürzung `text-decoration` lassen sich Farbe, Stil und Linienposition innerhalb einer Eigenschaft deklarieren, die `color-` und `style`-Werte gleichen den jeweiligen `border`-Werten (**[7]**).[104]

[102] Beinert (2015): „Sperren”
[103] W3C: CSS3TEXT, #text-transform-property
[104] W3C: CSS3TDECO

```
h2:first-letter{      [6]
   font-size:24pt;
   font-family:Zapf-Chancery, cursive;
   font-weight:bold;
   text-shadow:grey 0.5mm;   [5]
}
h2{
   …
   text-decoration:orange solid underline; [7]
}
```

Code 37: CSS Initialen (Textschatten und Textdekoration)

Abb. 17: AHF Initialen

Mit der **`@font-face`** können benutzerdefinierte Schriftarten eingepflegt werden (**[8]**). Die Regel weist den Renderer an, die Schriftart von der angegebenen Host-Adresse herunterzuladen. Die Schriftart kann mit `font-family` benannt sowie mit u.a. `font-style` und `font-weight` gestaltet werden.[105] Während **`@font-face`** sich gut dafür eignet, Schriftarten aufzufrufen, die auf unseren eigenen Servern gespeichert wurden, bieten andere, weiter verbreitete Hosts Webadressen zum Herunterladen von Schriftarten an. Google Fonts bspw. stellt extra für CSS Schriftarten-Importe zur Verfügung.[106] Mit der **`@import`**-Regel, die ganz zu Beginn des Stylesheets stehen muss und mit der man normalerweise andere Stylesheets importiert, kann man diese externen Webfonts einpflegen (**[9]**).[107] Die Schriftarten beider Importarten kann man dann mit `font-family` in einem Deklarationsblock ganz normal aufrufen (**[10]**).

Ohne die **`@font-face`**- oder **`@import`**-Regel können nur die Schriften verwendet werden, die auf dem Benutzercomputer installiert sind.

```
@import url('https://fonts.googleapis.com/css?family=Pacifico');      [9]
@font-face{                                                          [8]
   font-family: MyFont;
   src: url(Fonts/myFont.ttf);
}
…
h1{
   …
   /*font-family: 'MyFont';*/
   font-family: 'Pacifico', cursive;      [10]
…
}
```

Code 38: CSS Schriftarten-Import

[105] W3C (2013): CSS3FONTS, #font-face-rule
[106] Google Fonts: https://fonts.google.com/
[107] W3C (2016): „Cascading and Inheritance Level3", #at-import

Abb. 18: AHF Renderergebnis Schriftarten-Import („Pacifico“)

Möchte man Text nicht nur horizontal von links nach rechts formatieren, verwendet man `writing-mode` und `text-orientation`. Eine vertikale Darstellung von Elementboxen ist mit `writing-mode:vertical-lr` (Schreibfluss von links nach rechts, **[11]**) bzw. `-rl` (Schreibfluss von rechts nach links) möglich, der Defaultwert der Funktion ist `horizontal-tb` (Schreibfluss von oben nach unten, horizontale Ausrichtung). Zusätzlich kann dann die Auslegung des Boxinhalts bestimmt werden, z.B. wenn die Buchstaben eines vertikalen Wortes trotzdem „richtig herum“ angezeigt werden sollen. Dafür benutzt man die Eigenschaft `text-orientation` (Werte: `mixed`, `upright`, `sideways`; **[12]**).[108]

```
.title{
   writing-mode:vertical-lr; [11]
   text-orientation: upright; [12]
}
```

Code 39: CSS Schreibfluss

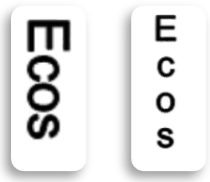


Abb. 19: vertikaler Schreibfluss, text-orientation-Werte (l.) sideways, (r.) upright

Leider besitzt das CSS-Konzept keinerlei Variablenkonzept, wie es XSL-FO mit seinen Attributsets bietet, sodass man keine Attributgruppen bilden kann und jede Änderung lokal passieren muss.

Defaultwerte, d.h. Standardwerte, kann in CSS im Deklarationsblock des Allselektors (Kleene-Start, *) beschrieben und in spezifischeren Elementselektionen überschrieben werden. Dies ist möglich, da die Priorität für den Renderer bei der Umsetzung so bestimmt ist, dass die genauere Selektion Vorrang vor den allgemeineren Selektionen hat.

4.3.2.3 Fazit Abschnitt

Dass man mit dem (momentanen) CSS-Konzept keine Attributvariablen anlegen kann, hat aufwändige Folgen. Änderungen, die eigentlich eine typografische Gruppe betreffen (z.B. Überschriften), müssen lokal bei jeder einzelnen entsprechenden Elementselektion zeitraubend angewendet werden, anstatt in globalen Variablen (besonders ärgerlich in einem großen Stylesheet).

---

[108] Götz/Ott, S.60

Allerdings ist es positiv, dass es für die Textgestaltung in CSS viele Eigenschaften gibt, mit denen man nach Belieben formatieren kann.

4.4 Weitere Inhaltselemente

Neben reinem Text gibt es auch andere Elemente, die formatiert und gestaltet werden können: Listen, Tabellen und Grafiken.

Neben der strukturellen Erstellung dieser Konstrukte können die Inhaltselemente noch weiter gestaltet werden, damit sie sich besser an das Erscheinungsbild und den Wünschen bzw. Bedürfnissen des Nutzers anpassen können.

Position und Verhalten von u.a. Grafiken im Textfluss können modifiziert werden. Zusätzlich lassen sich individuelle Aufzählungszeichen bei Listen, Rahmen- und Kontureneinstellungen für Tabellen und Grafiken und noch weitere Eigenschaften wie Hintergründe anpassen.

4.4.1 Listen

Aufzählungen und Beschreibungen, ob geordnet (mit Ordnungszahl o.ä.) oder ungeordnet mit Listensymbolen, lassen sich am besten mit einer Liste darstellen, z.B. Aufgabenlisten oder eine Auflistung von Stichworten zum Thema. Die Gestaltung der Abstände in den Listen, der Aufzählungszeichen und Listen mit mehreren Ebenen soll betrachtet werden.

4.4.1.1 XSL-FO

Das Listenkonzept in XSL-FO ist aus vier verschiedenen Elementen aufgebaut (*s. Anhang: Abb.20, Code 40*). Die Liste (**`<fo:list-block>`**) besteht aus einem oder mehreren Listeneinträgen (**`<fo:list-item>`**). Diese wiederum haben einen Labelblock (**`<fo:list-item-label>`**) zu Beginn und einen darauffolgenden Block für den Inhaltskörper (**`<fo:list-item-body>`**).[109]

XSL-FO bietet für Elemente einer Liste mehrere Modifikationsmöglichkeiten. Z.B. lassen sich Abstände zwischen Anfang oder Ende des Labelblocks und dem Beginn des Listeninhalts (`provisional-distance-between-starts`, `provisional-label-separation` bei **`<fo:list-block>`**) sowie zwischen Label oder Inhaltselement und dem Seitenrand durch Einrückung (`start-indent/end-indent` bei **`<fo:list-item-label>`** oder **`<fo:list-item-body>`**) bestimmen.[110]

[109] Montero/Krüger, S.106
[110] Ebd., S.109

Möchte man eine Liste mit einer oder mehreren weiteren Ebenen erzeugen (z.B. für Unterkapitel im Inhaltsverzeichnis), muss man je eine weitere Liste in der ersten Liste verschachteln – also in den Listenkörper (***<fo:list-item-body>***) einen neuen ***<fo:list-block>*** einbauen.

Listennummerierungen zu erzeugen oder zu steuern, ist in XSL-FO nicht eigens möglich. Eine automatisch erzeugte Nummerierung muss mit dem XSLT-Element ***<xsl:number>*** in Labelposition geschehen. Damit lassen sich auch mehrere Ebenenstufen nummerieren. Andere Aufzählungszeichen können jeweils im passenden Listeneintrag im Labelblock u.a. durch ein Unicode-Zeichen oder eine Grafik festgelegt werden.[111]

4.4.1.2 CSS [CSS3LIST, SELECT]

Liegt das zu gestaltende Dokument im HTML-Format vor (*s. Anhang: Code 41*), so wird die HTML-Liste automatisch gerendert. Ungeordnete Listen (***<ul>***) werden mit einem Listensymbol vor jedem Listenelement (***<li>***) abgebildet, geordnete Listen (***<ol>***) bekommen durchnummerierte Ordnungszahlen.

Möchte man selbst deklarieren, geschieht das mit `display:list-item` in den Listenelementen. Abstände lassen sich wie bei anderen Blockelementen auch mit `margin` und `text-indent` oder Breiteeinstellungen (z.B. der Labelbox) bestimmen.

Listenlabel lassen sich mit CSS auch recht einfach und schnell einsetzen bzw. bearbeiten. Mit `list-style` (Abkürzung für `list-style-type`, `-position` und `-image`; **[1]**) kann man für das Label unterschiedliche Nummerierungen, Grafiken oder andere Zeichen, die Position dieser (ob ausgerückt oder innerhalb des Listenblocks) und Hintergrundfarbe in geordneten und ungeordneten Listen bestimmen. Auch hier können zusätzliche Attribute der Listenelemente zur weiteren Differenzierung und Bestimmung des Zeichens führen.[112]

```
ul.keywords{
   list-style:square outside;          [1]
}
```

Code 42: CSS Listenlabel list-style

Mit dem Pseudoelement `::marker` **([2])**kann das Label direkt und individuell formatiert werden wie durch `::before`.[113] So könnte man das Label wie normalen Content/Inhalt formatieren oder bspw. eine `counter()`-Funktion für eine automatische Zählung oder eine `symbols()`-Funktion für weitere Aufzählungszeichen angeben.

---

[111] Montero/Krüger, S.109
[112] W3Schools: „CSS Lists"
[113] W3C: CSS3LIST, #marker-pseudo-element

```
ul.keywords li::marker{    [2]
   font-weight: bold;
   width:1cm;
}
```

Code 43: CSS ::marker

Abb. 21: AHF Liste

Ist eine zweite Liste in einem Listenelement integriert und soll die Nummerierung der äußeren Liste fortgeführt werden, so kann man auch mehrere benannte Zähler mit unterschiedlichem Format für jede Liste erstellen und verschachteln.

```
<ol>
...
  <li>
    <ol>                              [9]
      <li>...</li>
      ...
    </ol>
  </li>
</ol>
```

Code 44: HTML Liste zweiter Ebene

Wenn man bspw. zwei geordnete Listen ineinander verschachtelt vorliegen hat, kann man folgende Vorgehensweise mit Zählern nutzen:[114]

```
ol{
   counter-reset: item1;              [10]
}
ol > li{
   display: block;                    [11]
}
ol > li::before {
   content: counter(item1) ". ";
   counter-increment: item1;
}
ol > li > ol{
   counter-reset: item2;              [10]
}
ol > li > ol > li::before{
   content: counter(item1) "."counter(item2) ". ";      [13]
   counter-increment: item2;
}
```

Code 45: CSS Liste zweiter Ebene, „nested counters“

[114] W3C: CSS3LIST, #nested-counters

Für jede geordnete Listenebene wird ein Zähler erstellt, der jeweils jedes Listenelement darin zählt (**[10]**). Auch diese Erstellung der Nummerierung könnte in den XSLT-Prozess ausgelagert werden. Vor jedem dieser Listenelementblöcke (**[11]**) soll nun der aktuelle Wert des Zählers ausgegeben und dann um eins erhöht werden (**[12]**). Bei der Liste zweiter Ebene wird einfach zuerst der Zählerwert der ersten Ebene ausgegeben und danach der aktuelle Wert der momentanen Listenebene angehängt (**[13]**).

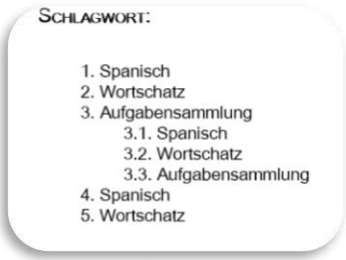


Abb. 1: AHF Liste zweiter Ebene

Mit den Pseudoklassenselektoren `:nth-child()` und `:nth-of-type()` kann übrigens innerhalb der Liste navigiert und beliebige Elemente ausgewählt werden.[115]

4.4.2 Tabellen

Tabellen sind u.a. nützlich für Gegenüberstellungen von Daten. Horizontale und vertikale Linien strukturieren die tabellarischen Informationen und machen sie übersichtlicher und besser vergleichbar.

Mit Tabellen wurden früher Seitenlayouts aufgebaut und die Positionierung damit bestimmt – die Entwickler haben aber dazugelernt und können heutzutage andere Arten geometrischer Strukturen schaffen. Gelegentlich wird aber manches, abseits der Verwendung für Daten zum Vergleich, sehr gut durch tabellarische Strukturen ausgedrückt – nummerierte Überschriften sind ein Paradebeispiel dafür.

Eine nummerierte Überschrift mit einer Tabelle zu formatieren, hat seine Vorteile. So bilden die Zahl und der Überschriftentext eigenständige Blöcke, können eigenständig formatiert werden, bei Umbruch springt der Text nicht unter die Zahl und die Zahl kann am oberen Rand der Tabellenzeile ausgerichtet werden. Ein einfaches Word-Tabellen-Beispiel zur Verdeutlichung:

1.	*Dies ist ein Beispiel. Dies ist ein Beispiel. Dies ist ein Beispiel. Dies ist ein Beispiel.*

Um eine Tabelle noch weiter zu gestalten und zu verbessern (zumindest in den meisten Fällen), ist es möglich, auf einige Ideen zurückzugreifen. Tabellenspalten oder -zeilen kann man zweifarbig gestalten oder gar eine Hintergrundgrafik einfügen und damit die

[115] W3C: CSS3SELECT

Zusammengehörigkeit der Spaltendaten bzw. Zeileninformationen betonen. Kopf- und Fußzeilen gibt es auch in Tabellen, um z.B. Spalten zu kennzeichnen oder Tabellenfußnoten zu erstellen.

4.4.2.1 XSL-FO

XSL-FO besitzt eine eigene Tabellenstruktur, die für typografische Anforderungen eines Satz- und Umbruchsystems geeignet ist, sich von der traditionellen CALS-Tabellenstruktur (Repräsentationsstandard für Tabellen im SGML/XML-Format, u.a. bei TEI eingesetzt) unterscheidet und der HTML-Tabelle ähnelt. Das CALS- und HTML-Tabellenformat könnte gut zu XSL-FO-Tabellen transformiert werden (*s. Anhang: Abb. 22, Code 46*).

Innerhalb der Tabelle (***<fo:table>***) und dem Tabellenkörper (***<fo:table-body>***) befinden sich Reihen (***<fo:table-row>***) mit Zellelementen (***<fo:table-cell>***) und jeweiligem Blockinhalt. Es könnten noch eine Kopf- und Fußzeile (***<fo:table-header/-footer>***) eingefügt werden und zusätzlich allgemeine Informationen über die Spalten angeben (***<fo:table-column>***). Sollte noch eine Tabellenüberschrift eingefügt werden, so kann man das vor ***<fo:table>*** in ***<fo:caption>*** machen. Dann muss aber auch das die gesamte Tabelle umfassende Element ***<fo:table-and-caption>*** eingesetzt werden.[116]

Für die Gestaltung der Tabellen gibt es viele Eigenschaften. Neben den üblichen typografischen Textgestaltungen, Breite- oder Rahmeneinstellungen des Tabellenelements, können z.B. auch benachbarte Zellenrahmen durch das Attribut `border-collapse` zu einer gemeinsamen Linie zusammenfallen (oder je nach Wert separiert werden).[117]

Durch entsprechende XPath-Ausdrücke können detaillierte Elementselektionen stattfinden, sodass man diese nach Belieben gestalten kann, z.B. jeder zweiten Tabellenreihe einen andersfarbigen Hintergrund zuweisen.

Kopf- und Fußzeilen werden, wie bereits erwähnt, mit den entsprechenden Tabellenelementen ***<fo:table-footer>*** und ***<fo:table-header>*** erstellt, welche bei einem Seitenumbruch innerhalb der Tabelle sogar auf der nächsten Seite wiederholt werden, außer man stellt diese Wiederholung durch ein entsprechendes Attribut (`table-omit-[footer|header]-at-break`) aus. Die bestimmte Überschrift (***<fo:table-caption>***) ist nur einmalig und fest.[118]

Eine nummerierte Überschrift lässt sich auch mit XSL-FO in eine Tabellenform bringen. Dabei werden zwei Spalten gebildet. In den Tabellenkörper kommt eine Tabellenreihe und da hinein

---

[116] Montero/Krüger, S.116ff
[117] Ebd., S.117
[118] Ebd., S.117f

zwei Zellen, eine um einiges breiter als die andere. Letztendlich werden dann das Label in Zelle eins und der Text in Zelle zwei eingebettet.[119]

4.4.2.2 CSS [SELECT]

Da CSS mit HTML zusammenarbeitet, können HTML-Tabellen mit kompatiblen Renderern automatisch umgesetzt werden. Möchte man andere Elemente zu einer Tabelle strukturieren, nutzt man `display` mit den jeweiligen Werten.[120] W3C arbeitet gerade an einem CSS3-Tabellenmodul (CSS3TABLE), welches allerdings noch nicht implementierbar ist.[121]

Neben den üblichen Tabellenelementen ***<table>***, ***<tr>***, ***<th>*** und ***<td>*** (*s. Anhang: Abb. 23*) exisitieren im HTML-Tabellenkonzept auch Kopf- und Fußzeilen (***<thead>***, ***<tfoot>***), Spaltenangaben (***<col>***, ***<colgroup>***) und eine Tabellenüberschrift (***<caption>***). Diese lassen sich mit den XSL-FO-Elementen vergleichen, mit dem Unterschied, dass HTML auch Spaltengruppen (***<colgroup>***) beinhaltet, dafür aber kein Element benötigt, das Überschrift und Tabelle umschließt.

Die Tabellen lassen sich mit CSS noch weiter gestalten. Dabei können die meisten Eigenschaften verwendet werden, die für andere (Inline-)Blöcke gelten. Ähnlich wie FO hat CSS u.a. auch das Attribut `border-collapse`, das die Zellrahmen verbinden kann (**[1]**).

```
.control {
   border-collapse: collapse;                    [1]
}
.control, .control th, .control td {
   border: 1px solid black;
}
```

Code 47: CSS Tabelle und Rahmen

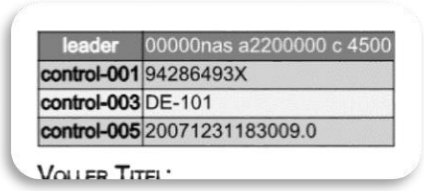
Eine abwechselnde farbliche Hinterlegung der Tabellenreihen lässt sich mit dem Pseudoelementselektor `::nth-child()` realisieren (**[2]**). Damit wählt man z.B. alle geradzahligen oder ungeradzahligen Tabellenreihen aus und bestimmt deren `background-color`. So entsteht eine Art Zebramuster.

```
.control tbody tr:nth-child(even){[2]
   background-color: #f2f2f2;
}
.control tbody tr:nth-child(odd){  [2]
   background-color: lightgrey;
}
```

Code 48: CSS „Zebramuster“ (:nth-child())

---

[119] Montero/Krüger, S.152f
[120] W3C: CSS2.2VIS, #propdef-display
[121] W3C: CSS3TABLE



leader	00000nas a2200000 c 4500
control-001	94286493X
control-003	DE-101
control-005	20071231183009.0

VOLLER TITEL:

Abb. 24: AHF Tabelle „Zebramuster“

Die Kontrolle über Wiederholung der Kopf- und Fußzeilen (***<thead>***, ***<tfoot>***) bei einem Seitenumbruch lässt sich bei CSS nur mit einer AHF-Erweiterung erlangen, `-ah-table-omit-[footer|header]-at-break` (*siehe Anhang: Code 49, Abb. 25/26*), diese funktioniert wie die ähnlich benannte XSL-FO-Eigenschaft. Der Wert `true` verhindert die Wiederholung der Kopf- oder Fußzeile bei Seitenumbruch, während der Wert `false` diese erlaubt.[122]

Eine nummerierte Überschrift in einer Tabelle (*s. Anhang: Code 50*) mit CSS zu gestalten, sieht bspw. folgendermaßen aus:

```
.title{
   margin-top:6mm;
   margin-bottom:4mm;
   page-break-after:avoid;          [1]
   page-break-inside:avoid;         [1]
}
.t_number, .t_name{
   font-size:15pt;
   font-weight:bold;
}
.t_number{
   vertical-align:top;          [2]
   text-indent:-5mm;
}
```

Code 51: CSS Tabelle Titel; Renderergebnis und Vergleich s. Anhang: Abb. 27

Umbrüche innerhalb und nach der Tabelle mit Klasse „`title`“ werden vermieden, damit der gesamte Titel mit dem nachfolgenden Kontext zusammenbleibt (**[1]**). Außerdem wird die Titelzahl mit `vertical-align:top` an den oberen Rand/Start der ersten Zelle verschoben (**[2]**), damit sie auf Höhe der ersten Zeile des nachfolgenden Titelnamens ist, sollte dieser zu lang für eine Zeile sein und deswegen umbrechen (*s. Anhang: Abb. 27*).

4.4.3 Abbildungen

Um Abbildungen für Printausgaben zu gestalten, spielen die Positionierung der Abbildungsbox, der Textumfluss sowie die Rahmengestaltung etc. eine wichtige Rolle.

[122] AH: „XSL/CSS Extensions“, #css3

Zudem ist es wünschenswert, Hintergrundbilder und Wasserzeichen hinter dem Inhalt platzieren zu können.
Inwieweit können also Abbildungen, d.h. „Grafiken und andere bildliche Darstellungen“[123], reibungslos in das Layout eingefügt und positioniert werden?

4.4.3.1 XSL-FO

Mit XSL-FO können bereits im Dokument existierende, aber auch externe Abbildungen eingefügt werden. Mit ***<fo:external-graphic>*** und dem Attribut ***src*** können sowohl blockartige als auch inzeilige Grafikboxen eingebettet werden. Die Größe kann durch `content-height/-width` bestimmt werden. Sollte nur eine der Eigenschaften deklariert werden, wird die andere jeweils automatisch proportional skaliert.[124]

```
<xsl:template match="Grafik">
  <fo:block>
    <fo:external-graphic src="{unparsed-entity-uri(@Datei)}" content-width="12px" />
  </fo:block>
</xsl:template>
```

Code 52: XSL-FO externe Grafik einbetten; angelehnt an Montero/Krüger, S.175

Mit dem Element ***<fo:float>*** und der gleichnamigen `float`-Eigenschaft kann die Abbildung dann je nach Wert auf der Seite, genauer im umgebenden Block, verschoben sowie vom Text umflossen werden. Sie kann an den Start- oder Endrand (von der Schreibrichtung abhängig, *`start/end`*) bzw. an den linken oder rechten Rand (*`left/right`*) positioniert werden, dann fließt der weitere Inhalt jeweils auf der anderen Seite vorbei. Bei einem Doppelseitenlayout kann der Innen- oder der Außenrand einer Seite zur Platzierung ausgewählt werden. Auf einer linken Seite wäre „innen“ dann rechts und „außen“ links, auf der rechten Seite vice versa.[125]

Hintergrundabbildungen können mit dem Attribut `background-image` im gewählten Blockelement alternativ zur Hintergrundfarbe `background-color` eingesetzt werden. Die Position kann mit `background-position-vertical/-horizontal` (kurz: `background-position`) hinterlegt werden. Außerdem kann die Wiederholung des Bildes innerhalb des Blocks bestimmt werden (`background-repeat`).[126]

---

[123] Montero/Krüger, S.112
[124] Ebd., 175f
[125] Ebd., S.122ff
[126] Ebd., S.114

4.4.3.2 CSS [CSSPF, CSS Figures, CSS3BG]

Eingefügt werden können mit CSS keine externen Abbildungen, außer Hintergrundbilder, deshalb müssen die Grafikreferenzen (***<img>***-Element) bereits im HTML-Dokument vorhanden sein.

Diese können dann mit `width` und `height` an die Seite angepasst werden, z.B. mit einem relativen Breitenwert von 50%, der sich an der Breite der Seitenbox orientiert und außerdem die Höhe automatisch mitskaliert. Zentrieren lässt sich die Box durch `margin:auto`.

Eine neue Eigenschaft, die CSS3 mitbringt, ist `border-radius`. Damit lassen sich abgerundete Rahmen erstellen, die je nach angegebenem Längenmaß skaliert werden (**[1]**).[127] Allerdings greift diese Einstellung nur, wenn das Objekt nicht mit `float` verschoben oder als `display:block` angezeigt wurde. Ansonsten wird der Inhalt in normaler Form über dem runden Rahmen angezeigt (*s. Anhang: Abb. 28/29*). Das Bild lässt sich dann aber auch nicht mehr mit `margin` formatieren, wenn die Box kein Blockelement ist.

```
img{
   width:50%;
   border:grey 1pt solid;
   border-radius: 1cm;  [1]
}
```

Code 53: CSS Grafik abgerundeter Rahmen; Renderergebnis s. Anhang: Abb. 28

```
img{
   display:block;
   width:50%;
   margin: auto;
   border:grey 1pt solid;
   border-radius: 1cm;
   float:top;
}
```

Code 54: CSS Grafik Positionierung; Renderergebnis s. Anhang: Abb. 29

In CSS3 werden neue `float`-Werte vorgestellt, die die Positionierung und den Textumfluss der Grafiken (u.a.) bestimmen soll.[128] Zur Positionierung gibt es die Werte `top` und `bottom` (*s. Anhang: Abb. 30/31*). Damit lässt sich die Box entweder an den Anfang der Seitenbox oder eben an das Ende verschieben, sodass die obere bzw. untere Randkante der Box mit der jeweiligen Randkante der Seitenbox übereinstimmt/abschließt.
Damit auch auf Doppelseiten um die Grafik-Box passend Text fließen kann, wurden außerdem die Werte `inside` und `outside` erweitert (**[2]**). Diese entsprechen den bereits vorhandenen Werten `left` und `right`, sind allerdings darauf abgestimmt, ob sich das Element auf einer rechten

[127] W3Schools: „CSS3 Rounded Corners"
[128] W3C: CSS3FLOATS

oder linken Seite befindet und verschieben die Box entsprechend an den äußeren oder inneren Rand der Seite. Mit `clear` kann bestimmt werden, dass die nachfolgenden Elemente der verschobenen Box diese nicht oder nur auf einer Seite umfließen sollen.

Laut W3C beinhaltet die CSS3-Version auch die Eigenschaft `float-defer`, mit der man die Box zwischen Spalten oder Seiten verschieben kann, und außerdem `float-offset`, was das Element in die gegensätzliche Richtung in die es verschoben wurde zurückschiebt.[129] Allerdings kennt AHF diese nicht (Fehlermeldung: "*Unknown property: 'float-reference'. Unknown property: 'float-defer'.*").

```
img{
   display:block;
   width:50%;
   margin: 3mm;
   border: grey 1pt solid;
   float:outside;              [2]
}
```

Code 55: CSS Textumfluss; Renderergebnis s. Anhang: Abb. 32

Es lassen sich auch mehrfach externe Hintergrundbilder für ein Element oder eine Seite einfügen mit `background-image`.[130] Ein Wasserzeichen kann so z.B. entstehen (*s. Anhang: Code 56, Abb. 33*), indem man im Deklarationsblock einer **`@page`**-Regel den Hintergrund ändert, die Größe des Bildes mit `background-size` auf *`100%`* stellt, also es über die gesamte Seite zieht, und das ganze zentriert (`background-position:center`). Oder man verwendet ein Muster, um einen Abschnitt zu hinterlegen (*s. Anhang: Code 57, Abb. 34*). Dieses Muster kann auch kleiner dargestellt werden und sich mehrfach wiederholen (`background-repeat`, kann auch Wert *`no-repeat`* bekommen). Mit der Kurzform background können alle diese Werte hintereinander aufgeführt werden.
Die mehrfache Hinterlegung von Bildern innerhalb eines Elements funktioniert allerdings nicht so, wie bei W3C beschrieben. Theoretisch sollte man, mit einem Komma abgetrennt, eine weitere Grafik einfügen können, ebenso mit Kommatrennung die Werte der anderen Attribute erweitern. Aber AHF beschwert sich über falsche Eigenschaftswerte.[131]

[129] W3C: CSS3FLOATS
[130] W3C: CSS3BG
[131] Ebd., #the-background-image

```
@page first:first{
   background-image: url(../Muster-hellgrau-blume.jpg), url(../Muster-hellgrau-
ecke.jpg);
   background-size: 50%%, 40%;
   background-position:top center, bottom center;
}
```

Code 58: CSS mehrfache Hintergrundbilder (funktioniert nicht)

Vorsicht bei der Verwendung der `*:empty`-Selektion, mit der man alle leeren Elemente des Dokuments formatieren kann. Meistens ist ein `img`-Element leer, weshalb die Selektion und deren Deklaration also auf Grafiken Auswirkungen hat. Z.B. wird ein Bild nicht angezeigt, sollte man durch `display:none` die Anzeige leerer Elemente verhindert haben.

4.4.4 Zwischenfazit

Die Liste und die Tabelle selbst muss schon in der richtigen Reihenfolge und Form vorhanden sein, um sie mit CSS zu formatieren – generiert werden können z.B. nur die Listenzeichen. Dafür gibt es für die Zeichen einige Methoden zur Gestaltung.

Ein eindeutiger Vorteil von HTML: Formatierer wissen bereits, wie sie eine HTML-Liste oder -Tabelle umzusetzen haben. Wenn man dazu bedenkt, dass XSL-FO ebenfalls einen „Vorprozess“ durchführt, um die erwünschte Struktur der Liste zu erhalten, kann man sagen, dass die CSS-Listen schneller und und einfacher umzusetzen sind.

Allerdings wird der CSS-Code schnell unübersichtlich, sobald eine Liste mehrere Ebenen enthält. Das führt dann zu einer Vielzahl an Elementselektionen und -deklarationen.

Auch die Grafiken sollten am besten schon an der richtigen Stelle im Dokument stehen, da die Positionierung mit CSS eher mangelhaft ist. Die Verschiebung funktioniert nur nach oben, unten rechts oder links – die Verschiebung zwischen Zeilen, Seiten oder Spalten ist nicht möglich. Auch die Gestaltung der Abbildung mit runden Rahmen funktioniert nur ohne weitere Modifikationen und der Textumfluss ist nicht detailliert steuerbar. Vieles davon sind Implementierungsfehler vom Formatierer, aber auch das Konzept zur Bilderintegration und -gestaltung müsste noch erweitert werden.

Leider lassen sich externe Grafiken mit CSS auch nicht so gut einbauen, wie es in XSL-FO möglich ist. Hintegrundgrafiken dürfen aus externer Quelle stammen, auf andere Abbildungen muss jedoch im Dokument referiert werden.

Eine Eigenschaft zur Transparenz der eingefügten Grafiken, vor allem für Hintergrundbilder, wäre bspw. etwas, das im Konzept noch erweitert werden könnte.

4.5 Automatisch generierte Elemente

Bei der Erstellung eines PDF gibt es einige Strukturen und Inhalte im Enddokument, die so nicht in den Ausgangsdaten stehen. Fußnoten, Marginalien wie Kolumnentitel und Seitenzahlen oder Verweisstrukturen wie Inhaltsverzeichnisse oder Sachregister müssen erst erzeugt werden. Auch Zusatzinformationen wie Schnittmarken, Auflösungen oder Farbeinstellungen müssen generiert werden.

4.5.1 Fußnoten

Fußnoten sind ebenfalls häufiger Bestandteil zu formatierender Printmedien. Im Dokument erscheint i.d.R. anstelle einer zusätzlichen Information oder Quellenangabe eine hochgestellte Ziffer, welche auf einen Block am Ende der Seite weist, der diese Information unter gleichzahliger Fußnotenziffer aufführt. Dieser Fußnotenblock wird gerne auch mit vorangehendem Weißraum/Abstand oder einer Linie vom Haupttext getrennt.

4.5.1.1 XSL- FO

An Stelle des Elements im Dokument, das zur Fußnote formatiert werden soll, wird ein Block mit einem ***<fo:footnote>*** gesetzt (**[1]**). Darin definiert man ein Inlineelement für das Verweiszeichen, das im Text stehen bleibt (**[2]**), und einen Fußnotenkörper (***<fo:footnote-body>,*** **[3]**). In letzterem kann dann, am besten durch eine Liste, die formatierte Information dargestellt werden (**[4]**) – das zugehörige Fußnotenzeichen als Label und der Informationstext im Listenkörper. Eine automatische Nummerierung erlangt man wieder durch das XSLT-Element ***<xsl:number>***, wenn nicht eine im Dokument festgelegte oder selbstbestimmte Ziffer oder ein Zeichen eingesetzt wird.[132]

```
<fo:block>
  Text
  <fo:footnote>                [1]
    <fo:inline font-size="15pt" baseline-shift="super">1</fo:inline>  [2]
    <fo:footnote-body>         [3]
      <fo:list-block ...>      [4]
      ...
      </fo:listblock>
    </fo:footnote-body>
  </fo:footnote>
</fo:block>
```

Code 59: XSL-FO Fußnote; Montero/Krüger, S.110

[132] Montero/Krüger, S.110ff

Eine Fußnotenlinie zur Trennung der Fußnote vom Haupttext lässt sich mit einem bereits kennengelernten ***<fo:leader>***-Element erstellen (**[5]**). Dieses muss in der Seitenfolge (***<fo:page-sequence>***) als statisches Element festgelegt werden.[133]

```
<fo:page-sequence master-reference="PageMaster.Inhalt">
...
  <fo:static-content flow-name="xsl-footnote-separator">
    <fo:block text-align-last="justify" space-before="5mm" space-after="3mm">
  [5] <fo:leader leader-length="30%" rule-thickness="1pt" leader-pattern="rule"/>
    </fo:block>
  </fo:static-content>
...
```

Code 60: XSL-FO Fußnote Trennlinie; Montero/Krüger, S.162f

4.5.1.2 *CSS [CSS3GCPM]*

In CSS muss ein Element, das zur Fußnote werden soll, mit `float:footnote` umgewandelt werden. Dadurch passieren mehrere Schritte gleichzeitig:[134]

1. Ein Verweiszeichen wird im Haupttext hinterlassen (`::footnote-call`).
2. Das Element samt Inhalt wandert („floats“) in den für Fußnoten vorgesehenen Bereich am Ende des Seiteninhaltsbereichs (***@footnote***).
3. Vor dem Element in der Fußnotenbox wird ein entsprechendes Fußnotenzeichen erstellt (`::footnote-marker`).
4. Der hinterlegte Fußnotenzähler wird um eins erhöht (`counter(footnote, decimal)`).

Jeder dieser Bestandteile kann durch den Aufruf des ursprünglichen Elementnamens, den Pseudoelementen, bzw. über die ***@footnote***-Regel genauer formatiert werden. Für den Elementinhalt kann z.B. Satzart, Schriftgröße, Einzüge etc. deklariert werden. Für das Verweis- und Fußnotenzeichen können dieselben Eigenschaften wie für Listenzeichen benutzt werden. Das CSS-Konzept erstellt mit der Fußnotengenerierung automatisch eine `counter()`-Funktion, die eine automatische (dezimale) Nummerierung der Fußnoten gewährleistet, sollte man keine anderen Zeichen bestimmt haben.

Mit ***@footnote*** kann theoretisch die Darstellung (`footnote-display`, **[6]**) des Fußnotenbereichs verändert werden, was in der Praxis allerdings keine Auswirkungen zeigt (Fehlermeldung AHF: „*Unknown property*“). Außerdem lassen sich wie für jede andere Seitenbereichsbox `margin`, `border`, `background` etc. deklarieren. Indem man einen oberen Rahmen mit `border-top` festlegt,

[133] Montero/Krüger, S.162f
[134] W3C: CSS3GCPM, #footnotes

wird eine Trennlinie über der Fußnotenbox sichtbar (**[7]**). Diese kann mit den üblichen Rahmeneinstellungen gestaltet werden.

```
@page{
  ...
  @footnote{
    footnote-display:block;  [6]
    margin-top: 0.5em;
    border-top: thin solid black;  [7]
    padding-top: 0.5em;
  }
}
...
.footnote{
   float:footnote;
   font-size:10pt;
}
::footnote-marker {
  content:'[' counter(footnote) ']';
  list-style-position: outside;
}
::footnote-call {
  content: ' [' counter(footnote) ']';
  vertical-align: baseline;
  font-size: 100%;
  line-height: inherit;
}
```

Code 61: CSS Fußnote

Abb. 35: AHF Fußnote

Die Positionierung geschieht automatisch, d.h. durch den Formatierer wird der untere Rand des Fußnotenbereichs bündig mit Ende des Seiteninhaltsbereichs ausgerichtet, es lässt sich auch bisher keine andere Platzierung erreichen. Das bedeutet, dass Fußnoten für Spalten und gar mehrfache Fußnotenbereiche nicht umsetzbar sind.

Es gibt allerdings eine Möglichkeit, Randbemerkungen („side notes“) zu gestalten, indem man das ausgewählte Element mit `float:sidenote` in den vorgesehenen Bereich verschiebt. Dieser wird als Teil der `@page`-Deklaration mit `@sidenote` formatiert, darin wird er an den rechten Rand (aber noch innerhalb der Inhaltsbox) geschoben (`float:right`). Es wird außerdem dafür gesorgt, dass das Element nicht von Text umflossen wird und die Breite der Box wird eingestellt. Analog zur Fußnotengenerierung, aber nicht automatisch erstellt, gibt es auch für Randbemerkungen Verweiszeichen usw., nur eben mit dem Begriff `sidenote` anstatt `footnote` *(s. Anhang: Code 62, Abb. 36)*. Ohne die Deklaration der Sidenotezeichen wäre weder eine Rahmendarstellung

noch eine detailliertere Gestaltung der Box gelungen (*s. Anhang: Abb. 37*). Mit den Sidenotezeichen lässt sich eine umrahmte Box erstellen und nach Belieben gestalten.[135]

4.5.1.3 *Fazit Abschnitt*

Mit dem CSS-Konzept ist die Fußnotengenerierung einfach, da das meiste automatisch geschieht. Auch der vorhandene Fußnotenzähler ist praktisch, da er sich der Situation anpasst, sollte eine Fußnote hinzugefügt oder gelöscht werden.

Die Idee einer Sidenote ist gut, allerdings ist die Umsetzung nicht sehr gelungen. Ohne Deklaration der Vorzeichen lässt sich die Box nur wenig gestalten. Und da sie sich innerhalb der Inhaltsbox befinden, sieht das Ergebnis bei Randbemerkungen, auf die eine Überschrift oder ein langer Text folgt, unästhetisch aus, da sich die Elemente fast überlappen (*s. Anhang: Abb. 38)*.

Dass richtige Fußnoten bisher nur an einer Stelle, nämlich am Seitenende, positioniert werden können, ist schade. Denn bei einer professionellen Formatierung können auch bessere Randbemerkungen, Spaltenfußnoten oder alle dieser Varianten gemeinsam auftreten. Allerdings sind die CSS-Entwickler dabei, diese Schwäche zu beheben.[136]

4.5.2 Marginalien

Der Randbereich einer Seite kann auch gestaltet werden. Dort werden Kolumnentitel, Paginierungen oder andere Randinformationen erstellt oder hin verschoben.

4.5.2.1 Allgemein

Generierte Marginalien haben ihre Positionierung und grobe Gestaltung gemeinsam. Daher wird hier noch einmal auf die Einstellungen der Randbereiche im FO- und im CSS-Konzept eingegangen. Wie kann man auf diese zugreifen und sie dann am besten gestalten?

4.5.2.1.1 XSL-FO

Wie bereits im Abschnitt *4.1 Grundstruktur* beschrieben wurde, besteht eine Seite in XSL-FO aus Regionen (*s. Anhang: Abb. 3*). Die Hauptregion ***<fo:region-body>*** ist für den eigentlichen Inhalt reserviert, zusätzlich gibt es aber noch vier weitere Regionen im Randbereich, in die Marginalien eingefügt werden sollen. Der Kopf- und Fußbereich (***<fo:region-before>***, **[1]**/ ***<fo:region-after>***, **[2]**) sowie eine linke und rechte Region (***<fo:region-start>***, ***<fo:region-end>***).

Zuerst müssen in einer Seitenvorlage (***<fo:simple-page-master>***) die erwünschten Regionen spezifiziert werden, die nicht benötigten Randregionen müssen nicht angegeben werden. Die Regionen bekommen einen Namen zugewiesen, damit man den Bereichen später die Inhalte

---

[135] AH: „Float Extension", #FootnoteCSS
[136] W3C: CSS3GCPM, #future

zuordnen kann (`region-name/flow-name`). Mit `display-align` wird festgelegt, wo die Region abgebildet wird (**[3]**).[137]

```
  <fo:simple-page-master master-name="PageMaster.Inhalt-rechts" margin="15mm 25mm
30mm 13mm" page-height="297mm" page-width="210mm">
          ...
        <fo:region-before region-name="Inhalt-rechts-davor" extent="20mm"    [1]
         border-after-style="solid" border-after-width="1pt" display-align="before"/>  [3]
        <fo:region-after region-name="Inhalt-rechts-danach" extent="10mm"    [2]
         border-before-style="none" display-align="after" />    [3]
        <fo:region-body margin="25mm 0mm 15mm 0mm"/>
  </fo:simple-page-master>
...
```

Code 63: XSL-FO Marginalien Seitenvorlage; Ausschnitt aus Montero/Krüger, S.191

Als nächstes wird in der Seitenfolge (**`<fo:page-sequence>`**) der Inhalt mit den Bereichen verknüpft. Randregionen sind statische Bereiche (**`<fo:static-content>`, [4]**), im Gegensatz zum über die Seiten fließenden Inhaltsbereich (**`<fo:flow>`, [5]**). In diesem statischen Element wird dann ein Block gesetzt, der mit dem gewünschten Inhalt befüllt wird (**[6]**). Das können Seitenzahlen sein, Texte, lebende Kolumnentitel, Grafiken etc. Das alles kann jeweils für die linke und rechte Seite analog gestaltet werden.[138]

```
...
  <fo:page-sequence master-reference="Inhalt-Seiten">
     ...
      <fo:static-content flow-name="Inhalt-rechts-davor">  [4]
        <fo:block font-family="Arial" font-size="8pt" text-align="right">  [6]
          ...
        </fo:block>
      </fo:static-content>
      ...
      <fo:static-content flow-name="Inhalt-rechts-danach">
        <fo:block font-family="Arial" font-size="8pt" text-align="center">
          ...
        </fo:block>
      </fo:static-content>
      ...
      <fo:flow flow-name="xsl-region-body">  [5]
        <fo:block font-family="Arial" font-size="10pt">
          <xsl:apply-templates/>
        </fo:block>
      </fo:flow>
  </fo:page-sequence>
...
```

Code 64: XSL-FO Marginalien Seitenfolge; Ausschnitt aus Montero/Krüger, S.192

[137] Montero/Krüger, S.192f
[138] Ebd., S.193f

Zur besseren Gliederung können Kopf- oder Fußbereiche Tabellen zur Hilfe nehmen. Anstatt einen Block in das statische Element einzusetzen, wird dort eine einzeilige Tabelle deklariert. So kann z.B. ganz rechts im Kopfbereich eine Grafik geladen werden (Zelle eins), mittig in einer zweiten Zelle ein Kolumnentitel stehen und ganz links in der letzten Zelle die Seitenzahl. Die einzelnen Zellen können nach eigenem Ermessen weiter gestaltet und strukturiert werden.[139]

4.5.2.1.2 *CSS [CSS3PAGE, CSS3GCPM]*

In CSS ist alles eine Box – diese Information kann bei CSS immer wieder ins Gedächtnis gerufen werden. So sind neben der Inhalts-Box im Randkontext („margin-context“) 16 Margin-Boxen auf einer Seite des CSS-Konzepts zu finden (*s. Anhang: Abb. 5*).[140]

Dort wo XSL-FO zur Strukturierung einer Kopf- oder Fußzeile auf eine Tabelle zurückgreift, besitzt CSS bereits eine stärker unterteilte Struktur. Diese Randboxen berechnen sich normalerweise automatisch, abhängig von `margin`- und `size`-Eigenschaften, können aber einzeln angesprochen und modifiziert werden.

Innerhalb einer ***@page***-Regel (ob benannte oder allgemeine Seite) können die individuellen Boxen mit einem entsprechenden Bereichsselektor angesprochen werden:[141]

- Kopfzeile mit ***@top-left-corner***, ***@top-left***, ***@top-center*** (**[6]**), ***@top-right***, ***@top-right-corner***
- Fußzeile mit ***@bottom-left-corner***, ***@bottom-left***, ***@bottom-center***, ***@bottom-right***, ***@bottom-right-corner***
- Linker Randbereich mit ***@left-top***, ***@left-middle***, ***@left-bottom***
- Rechter Rand mit ***@right-top***, ***@right-middle***, ***@right-bottom***

```
@page {
    @top-center{            [6]
    content: "Hier kann ein Inhalt eingefügt werden"; [7]
    text-align:center;
  }
}
```

Code 65: CSS Marginbox

Darin können allgemeine Boxenstruktureigenschaften formuliert werden (`margin`, `border` etc.) oder mit der `content`-Eigenschaft Inhalte eingefügt und diese ebenfalls formatiert werden (**[7]**).

---

[139] Montero/Krüger, S.161
[140] W3C: CSS3PAGE, #margin-boxes
[141] Ebd., #page-selector-and-context

4.5.2.2 Lebende Kolumnentitel

Kolumnentitel sind in der Typografie Seiten- und/oder Kapitelzahlen und –überschriften im Randbereich, mit denen man innerhalb eines Dokuments einen besseren Überblick über die Seiten gewinnt und somit schneller zum gesuchten Inhalt kommen kann.

„Lebende“ Kolumnentitel sind solche Elemente, die sich an den auf der Seite befindlichen Inhalt anpassen und sich ändern, während man unter „toten“ Kolumnentiteln die Seitennummerierung (Pagina) versteht, oder aber auch ein anderes Element, das sich auf den nachfolgenden Seiten nicht ändert (z.B. Buchtitel). Kapitelüberschriften als lebendige Kolumnentitel bspw. können im Randbereich zeigen, in welchem Kapitel man sich befindet.[142]

Dabei können mehrere Kapitelüberschriften auf einer Seite vorhanden sein, weshalb man eine Regel festlegen muss, welches dieser Elemente letztendlich als Kolumnentitel verwendet werden soll.

4.5.2.2.1 XSL-FO

Zur Generierung von lebenden Kolumnentiteln stellt XSL-FO ein Marker-Konzept bereit (*s. Anhang: Code 66*). Grob umrissen funktioniert es so, dass man das Element markiert (mit ***<fo:marker>***), welches dann im Kopfbereich in der Seitenfolge aufgerufen werden soll (mit ***<fo:retrieve-marker>***). Ähnlich zu einem Variablenkonzept kann man mehrere Marker erstellen, unterschiedlich benannt, und diese an verschiedenen Stellen aufrufen. Man kann auch pro Region mehrere Elementkopien einfügen. Der Inhalt des Elements wird damit kopiert, d.h. das Originalelement wird trotzdem im Hauptbereich angezeigt.[143]

Weitere Attribute wären zum einen `retrieve-position`, dessen Wert angibt, ob der auf der Seite erstauftretende Marker bzw. deren Abschnitt auf der Seite enden (*`first-including-carryover`*) oder der letztauftretende Marker (*`last-ending-within-page`*) kopiert wird. Auf welchen Seiten der Seitenfolge dieser Marker kopiert werden soll, schreibt zum anderen die Eigenschaft retrieve-boundary vor. Der Wert `page-sequence` steht für die Verwendung des Markers, bis für den folgenden Abschnitt ein anderer Marker bestimmt wurde.[144]

Ebenso lässt sich der Kolumnentitel in die linke oder rechte Seitenregion übertragen. Dort muss allerdings evtl. der Text- und Schreibfluss angepasst werden mit `text-orientation` und `writing-mode`, damit der Titel lesbar bleibt und sich in den Seitenrand einpassen kann.

---

[142] Beinert (2017): „Kolumnentitel“
[143] Montero/Krüger, S.158f
[144] Ebd., S.159

4.5.2.2.2 CSS [CSS Books, CSS3GCPM]

Es gibt in CSS zwei Möglichkeiten, lebende Kolumnentitel zu erstellen. Entweder kopiert man einen Inhalt in die Randboxen oder man verschiebt das ganze Element samt Formatierung.

Die erste Methode („named strings“), das Kopieren, ist ähnlich zur XSL-FO Vorgehensweise. Man „markiert“ ein Element und ruft dieses in der entsprechenden Randbox auf. Auch das bekannte „Kopieren und Einfügen“-Prinzip („*copy & paste*“, z.B. bei Textbearbeitungsprogrammen) lässt sich damit vergleichen. Mit der Eigenschaft `string-set` wird eine ausgewählte Zeichenkette eines Elements als benannte Variable gespeichert (**[1]**). Dabei kann als zweites Argument noch bestimmt werden, was genau der kopierte Inhalt sein soll. Mit `content()` wird der Elementinhalt kopiert, mit `attr()` der Name eines ausgewählten Elementattributs, und mit `counter/s()` ein das Element betreffender Zählerwert.

Mit der `string()`-Funktion als Wert der `content`-Eigenschaft wird die genannte Variable aufgerufen und eingefügt. Mit einem modifizierenden, weiteren Argument kann dazu auch bestimmt werden, ob die erste Variable (`first`) bzw. die erste Variable, mit deren Element die Seite beginnt (`start`), übertragen werden soll oder ob der String der letzten Variable der Seite verwendet werden soll (`last`). Der vom W3C definierte Wert `first-except` wird im AHF nicht unterstützt. Es können außerdem auch mehrere Strings hintereinander aufgerufen werden (**[2]**).

Im Beispiel wird jeweils für die rechte und linke Seite eine unterschiedliche Reihenfolge der aufgerufenen Kolumnentitel bestimmt. Je Artikel der Auswahl wird der Titel aus der Überschriftentabelle kopiert. Der Titel (bestehend aus Nummer und Name) aus der Tabelle im HTML-Dokument soll als Kolumnentitel dargestellt werden (*s. Anhang: Code 67*).

```
@page auswahl:left{
    @top-center{
    content: string(colnumber, first) '\2002' string(coltitle, first);        [2]
    text-align:left;
  }
}
@page auswahl:right{
    @top-center{
    content: string(colnumber, first) '\2002' string(coltitle, first);        [2]
    text-align:right;
  }
}
...
.t_number{
  string-set:colnumber content();  [1]
  ...
}
.t_name{
  string-set:coltitle content();                [1]
}
```

Code 68: CSS Kolumnentitel string()-Variante, Renderergebnis s. Anhang: Abb. 39

Die zweite Variante („running elements“) wird ähnlich deklariert, hat aber einen ganz anderen Effekt. Mit der `position`-Eigenschaft und der `running()`-Funktion als Wert wird die Position eines Elements zu einem bewegbaren und mit Benennung aufrufbaren Objekt übertragen (**[3]**), das dann in eine Randbox mit der `content`-Eigenschaft und einer `element()`-Funktion verschoben wird (**[4]**). Das Prinzip ist mit dem „Ausschneiden und Einfügen“-Prinzip („*cut & paste*“) zu vergleichen, denn genauso wird das Element aus der originalen Position entfernt und in der neuen Position eingesetzt. Dabei werden sämtliche Struktur- und Stileigenschaften, die für dieses Element deklariert wurden, mit übertragen. Außerdem kann in einer `content`-Eigenschaft nur eine `element()`-Funktion deklariert werden, nicht wie bei der ersten Methode mehrere hintereinander. Eine Kombination dieser beiden Methoden in einer Randbox ist also auch nicht möglich.

Man verwendet diese Art der Kolumnentitelgenerierung nur, wenn ein passendes, zusätzliches Element im Text vorhanden ist, das dort nicht mehr gebraucht wird.

```
@page auswahl:left{
    @top-center{
        content: element(col);      [4]
        text-align:left;
  }
}
…
.title{
  position: running(col);           [3]
  margin-top:6mm;
  margin-bottom:4mm;
  page-break-after:avoid;
  page-break-inside:avoid;
}
```

Code 69: CSS Kolumnentitel element()-Variante; Renderergebnis s. Anhang: Abb. 40

Den Kolumnentitel in den linken oder rechten Seitenrand zu überführen, funktioniert auch. Dort muss, genauso wie in XSL-FO, der Text- und Schreibfluss angepasst werden mit `text-orientation` (**[5]**) und `writing-mode` (**[6]**), damit der Titel lesbar bleibt und sich in den Seitenrand einpassen kann. Mit writing-mode wird der Schreibfluss entweder horizontal ausgelegt, oder vertikal, d.h. von oben nach unten (*s. Anhang: Abb. 42*). Mit `text-orientation` können die Zeichen des vertikalen Elements gedreht werden (*s. Anhang: Abb. 41).*

```
@page register:left{
  ...
  @left-top{
    content:'Stichwortregister';
    text-orientation: upright;        [5]
    writing-mode: vertical-lr;        [6]
    text-transform:uppercase;
  }
}
@page register:right{
  ...
  @right-top{
    content: 'Stichwortregister';
    writing-mode: vertical-rl;
    text-transform:uppercase;
  }
}
```

Code 70: CSS Kolumnentitel rechter Seitenrand und Schreibfluss

4.5.2.3 Pagina (tote Kolumnentitel)

Mit Pagina ist die Nummerierung der Seiten im Seitenrand gemeint. Diese können u.a. entweder römische oder arabische Zahlen haben. Außerdem kann ein Dokument eine einzige, durchgängige Zählung haben oder eine geteilte Nummerierung mit „Zählerneustart“ ab einem bestimmten Abschnitt, falls z.B. nachträglich ein externer Abschnitt eingefügt werden oder das Inhaltsverzeichnis eine andere Nummerierung als der nachfolgende Hauptteil besitzen soll.

4.5.2.3.1 XSL-FO

Nachdem eine Seitenvorlage mit entsprechenden Regionen erstellt wurde (in diesem Beispiel ***<fo:region-before>***), kann in der Seitenfolge (***<fo:page-sequence>***) die Seitennummerierung mit FO festgelegt.

Das Attribut `initial-page-number` legt eine Start-Seitenzahl für die Seitenfolge fest (**[1]**). Beginnt danach eine andere Seitenfolge mit neuer Start-Seitenzahl, so wird die Seitenzählung geteilt, ansonsten läuft sie fort (Werte: *`auto`*|*`auto-odd`*|*`auto-even`*|*`<number>`*|*`inherit`*).[145]

Mit ***<fo:page-number>*** kann man dann im ***<fo:static-content>*** der Kopfregion die aktuelle Seitenzahl ausgeben lassen (**[2]**). Das Element kann mit einer Vielzahl an Attributen modifiziert werden, z.B. mit `font-family, font-style, vertical-align, text-transform` u.v.m.[146]

[145] Montero/Krüger, S.160/272
[146] Ebd. S.227

```
  <fo:simple-page-master master-name="PageMaster.Inhalt-rechts" margin="15mm
25mm 30mm 13mm" page-height="297mm" page-width="210mm">
    ...
        <fo:region-before region-name="Inhalt-rechts-davor" extent="20mm"
         border-after-style="solid" border-after-width="1pt" display-align="before"/>
        ...
  </fo:simple-page-master>
...
<fo:page-sequence master-reference="Inhalt-Seiten" initial-page-number="1"> [1]
      <fo:static-content flow-name="Inhalt-rechts-davor">
          <fo:block text-align="right">
              <fo:page-number/>      [2]
          </fo:block>
        ...
</fo:page-sequence>
```

Code 71: XSL-FO Paginierung; angelehnt an Montero/Krüger, S.160

4.5.2.3.2 CSS [CSS3PAGE]

Wenn in CSS irgendetwas gezählt wird, so passiert das auf Basis der `counter()`-Funktion und **`@page`**-Regel. Beim Rendern wird automatisch eine Seitenzählervariable (`counter(page)`) erstellt und je neu erzeugter Seite um den Wert eins erhöht, außer über CSS erfolgen andere Anweisungen.[147]

Möchte man die Seitenzahl in einer Randbox ausgeben, so verwendet man als Wert der `content`-Eigenschaft `counter(page)`. Damit ruft man den aktuellen Wert des Seitenzählers auf (*s. Anhang: Code 72*).

Soll nun ab einer neuen Seitenvorlage eine neue Seitenzählung beginnen, so wird es ein wenig komplizierter. Man wählt die erste Seite des gewünschten Seitentyps aus und setzt einen neuen Zähler (`counter-reset`), als Wert der Eigenschaft erstellt man einen Zählernamen (**[3]**).
Dann gibt man an, dass jede neue Seite des gewählten Seitentyps diesen Zähler um eins erhöhen soll (**[4]**).
Um die aktuelle Seitenzahl auszugeben, ruft man im `content` der entsprechenden Randbox der entsprechenden Seitenvorlage den erstellten Zähler auf (**[5]**). Das erste Argument der `counter()`-Funktion ist der Name des gewünschten Zählers und durch das zweite Argument lässt sich die Darstellung der Seitenzahl bestimmen (z.B. `upper-roman` für große römische Zahlen).

[147] W3C: CSS3PAGE, #page-based-counters

```
...
@page ivz:first{
   counter-reset: ivzpage;              [3]
}
@page ivz{
   counter-increment: ivzpage;          [4]
}
@page ivz:left{
   @top-left{
      content: counter(ivzpage, upper-roman);          [5]
      font-family:"Times New Roman";
   }
}
...
```

Code 73: CSS geteilte Paginierung; Renderergebnis s. Anhang: Abb. 43

Soll die Seitenzählung aus irgendeinem Grund nicht mit der Seitenzahl eins beginnen, kann man den Startwert beim Zählerneustart als zweiten Wert von `counter-reset` festlegen. Achtung: in diesem Beispiel wird der Wert 29 gesetzt (**[6]**), die Seitenzählung beginnt aber mit der Zahl 30, da direkt auf der ersten Seite der Zähler um eins erhöht wird. Dies hätte sich z.B. theoretisch mit dem `:not()`-Selektor vermeiden lassen, indem man anstatt `@page auswahl` die Formulierung `@page auswahl:not(:first)` verwendet. Aber leider ist eine solche Formulierung im CSS-Konzept für `:not()` nicht vorgesehen.

```
@page auswahl:first{
   counter-reset: mainpage 29;      [6]
}
@page auswahl{
    counter-increment: mainpage;
}
```

Code 74: CSS Paginierung Startzahl; Renderergebnis s. Anhang: Abb. 44

4.5.2.4 Zwischenfazit

Während XSL-FO Tabellen zur Strukturierung von Randbereichen nutzen muss, können in CSS einzelne Randboxen ausgewählt werden, deren Größe angepasst werden können und die man mit unterschiedlichem Inhalt füllen kann.

Positiv ist, dass es zwei Varianten gibt, Kolumnentitel zu erstellen. Denn so kann man entweder im Dokument angezeigte Elemente kopieren oder zusätzliche Elemente für alternative Kolumnentitel einfügen, die dann mitsamt der deklarierten Formatierung verschoben werden.

Bei der Auswahl, welches der auftretenden Elemente einer Seite als Kolumnentitel fungieren soll, haben die CSS-Entwickler auch einige gute Wertvorgaben eingesetzt – jedoch ist die Komplexität und Erweiterungsmöglichkeit bei XSL-FO wiederum größer (z.B. CSS‘ simples Funktionsargument „`first`“ entspricht XSL-FOs „`retrieve-position='first-including-carryover`“).

Seitenzählungen sind möglich, auch individuelle Seitenzählungen, aber ohne ein Konzept, wie die Seitenfolgen von XSL-FO, wird es kompliziert und erfordert sehr viel Schreibarbeit, wenn man für bestimmte Abschnitte unterschiedliche Zählungen haben möchte. Je komplexer die Musterseiten und deren Verschachtelung, desto mehr Code wird für die Umsetzung der erwünschten Seitenzählung benötigt.

4.5.3 Verweise

Verweise sind Verbindungen von einem Verweisursprung zum Verweisziel, dokumentintern (z.B. zu Kapiteln) oder auch -extern (z.B. zu Webadressen). Im nächsten Abschnitt wird erklärt, wie das im Allgemeinen funktioniert, d.h. wie man Verweise er- und darstellen kann. Danach werden zwei Anwendungsbeispiele aufgezeigt, ein Inhaltsverzeichnis und ein Register, in denen Verweise mit anderen typografischen Konzepten umgesetzt werden.

4.5.3.1 Allgemein

Wie funktioniert im Allgemeinen so eine Verweiserstellung und -darstellung mit XSL-FO und CSS? Was ist nötig, um eine referenzierte Seitenzahl oder den Text des Verweisziels an Stelle des Verweises in einem Printdokument anzeigen zu lassen?

4.5.3.1.1 XSL-FO

XSL-FO unterscheidet bei Verweisen zwischen Querverweisen und Hyperlinks. Für den Druck werden Querverweise verwendet, da Hyperlinks für interaktive Reader bestimmt sind (z.B. um vom Link-Ursprung durch Klicken zum Link-Ziel zu kommen). Querverweise können an Stelle des Verweisursprungs die Bezeichnung des Verweisziels erscheinen lassen.[148]

Das Verweisziel muss ein Attribut `id` erhalten, das das Element eindeutig identifiziert. Dort, wo die Verlinkung auftauchen soll, wird ein Element ***`<fo:basic-link>`*** gesetzt, und somit der Verweisursprung markiert. Dieses Element kann viele Attribute haben, hier werden allerdings nur die drei für diese Arbeit wichtigsten behandelt. Ist das Verweisziel dokumentintern, benutzt man das Attribut `internal-destination` (`external-destination` dann für dokumentexterne Webadressen oder andere Dokumente). Das Attribut erhält als Wert einen XPath-Ausdruck, der auf das Ziel verweist. Zusätzlich kann mit `show-destination` gesteuert werden, wie der Verweis angezeigt werden soll. Der Wert *`replace`* lässt das Verweisziel im Falle des Drucks auf der Druckseite anzeigen, während der Wert *`new`* für Printprodukte keine Bedeutung hat, da er das Verweisziel in einem neuen Fenster o.ä. öffnet.[149]

---

[148] Montero/Krüger, S.124ff
[149] Ebd., S.125

Die Auflösung des Verweises, also die Generierung der ID und das Auflösen des XPath-Ausdrucks, geschieht im ersten Schritt des XSL-FO-Prozesses, d.h. durch den XSLT-Prozess.

Im Folgenden wird ein beispielhafter Ausschnitt aus jeweils dem XML-Ausgangsdokument (*s. Code 75*), dem XSL-Stylesheet (*s. Code 76*) und dem resultierenden XML-FO-Dokument (*s. Code 77*) gezeigt, um die Vorgehensweise besser verständlich zu machen.
Das **`<titel>`**-Verweisziel (**[1]**) im XML-Dokument enthält ein `id`-Attribut mit Wert *`links`*, welcher auf den Verweisursprung **`<ref>`** mit gleichwertigem Attribut `idref` verweist (**[2]**):

```
<titel id="links">Querverweise/Hyperlinks</titel>                [1]
        ... text ... <ref idref="links"/> ... text ...           [2]
```

Code 75: XML-Dokument Verweis; Montero/Krüger, S.125

Nun wird im XSLT-Stylesheet bestimmt, dass für jedes **`<titel>`**-Element (**[3]**) und für den Verweisursprung des internen Verweises (**[4]**) ein `id`-Wert generiert wird.

```
<xsl:template match="titel">
  <fo:block id="{generate-id()}" ...>            [3]
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
 ...
<xsl:template match="ref">
  <xsl:for-each select="//titel[contains(@id,current()/@idref)]">
    <fo:inline text-decoration="underline">
      <fo:basic-link internal-destination="{generate-id()}" show-destination="replace">
                                  [4]
        <xsl:value-of select="."/>
      </fo:basic-link>
    </fo:inline>
  </xsl:for-each>
</xsl:template>
```

Code 76: XSL-FO Stylesheet Verweis; Montero/Krüger, S.126

Wenn dieses Stylesheet nun den transformierenden XSLT-Prozessor durchläuft, entsteht ein resultierendes XML-FO-Dokument, in dem die `id`-Werte für die jeweiligen Attribute des **`<titel>`**- und **`<fo:basic-link>`**-Elements eingesetzt wurden (**[5]** und **[6]**). Damit kann der Formartierer später den Hyperlink zwischen den Seiten des Verweisziels und Verweisursprungs herstellen. Innerhalb des **`<fo:basic-link>`**-Elements wurde der Inhalt des verwiesenen Titels eingefügt (**[7]**). Hätte man im Stylesheet anstatt **`<xsl:value-of>`** den Ausdruck **`<fo:page-number-citation>`** verwendet, würde die Seitenzahl des referenzierten Titels eingetragen werden.[150]

[150] Montero/Krüger, S.126

```
...
<fo:block id="IDAHGBZB" ...>Querverweise / Hyperlinks</fo:block> [5]
...
<fo:inline text-decoration="underline">
  <fo:basic-link internal-destination="IDAHGBZB" show-destination="replace"> [6]
    Querverweise / Hyperlinks                [7]
  </fo:basic-link>
</fo:inline>
...
```

Code 77: XML-FO-Dokument; Montero/Krüger, S.127

4.5.3.1.2 CSS [CSS3GCPM]

Mit CSS können die Element-IDs, die für einen Querverweis notwendig sind, nicht generiert werden, dafür muss der Vorprozess sorgen. Das HTML-Element für Links und Verweise ist das **`<a>`**, einem **`<fo:basis-link>`** entsprechend. Das Linkelement **`<a>`** hat ein Attribut `href`, das u.a. mit Raute(#) und Verweisziel-ID aufgebaut ist, falls es ein interner Verweis ist, oder einer URL, wenn es ein externer Link ist. Die ID kann von XSLT entweder generiert werden (ähnlich wie im XSL-FO Stylesheet) oder einen einzigartigen Wert annehmen, der schon in den Daten vorhanden ist (z.B. ISBN eines Buchs).

Im HTML-Dokument (*s. Anhang: Code 78*) steht nach dem XSLT-Vorprozess als Verweisursprung ein Linkelement (**`<a>`**) mit Attribut href und einem ID-Wert, der mit der ID des Verweisziels (Attribut `id`) übereinstimmt. Möchte man den Titeltext im Verweisursprung anzeigen, so kann man das entweder im XSLT-Vorprozess durchführen (z.B. im selben Schritt wie die ID-Verknüpfung des einzigartigen, bereits vorhandenen Datenwerts) oder mit `content`-Eigenschaft und als Wert einer *`target-text()`*-Funktion[151] arbeiten. Das erste Argument der Funktion ist die URL der auszuwählenden Zeichenkette. Das optionale, zweite Argument, das den `string-set`-Argumentwerten gleicht, gibt an, ob man den direkten Inhalt möchte (*`content`*, Defaultwert), den ersten Buchstaben (*`first-letter`*) oder die Zeichenkette vor bzw. nach dem Element (*`before, after`*).

```
a {
  content: target-text(attr(href url));
}
```

Code 79: CSS Verweis

An Stelle des Links wird dann durch *`target-text()`* der textliche Inhalt des Elements eingefügt, auf das in `href` verwiesen wird. Das funktioniert allerdings nicht, sollte der gewünschte Text nicht direkt in dem Verweiszielelement (oder dem durch die Argumente erreichbaren Rahmen) stehen.

[151] W3C: CSS3GCPM, #target-text

Es kann auch u.a. die Seitenzahl des referenzierten Elements im Dokument angezeigt werden. Dafür verwendet man die `target-counter[s]()`-Funktion.[152] Durch diese können numerische Querverweise generiert werden, die der `counter()`- bzw. `counters()`-Funktionsweise entsprechen. Mit `target-counter()` wird allerdings der Wert einer Zählervariable im Bezug auf die Position eines Elements im Dokument abgefragt. Das erste Argument ist die URL des Verweisziels, das zweite Argument gibt an, welchen der Zähler (`counter()`) dieses Elements man möchte (z.B. ‚`page`', den Seitenzähler). Durch das dritte Argument kann der Zählerstilname angegeben werden (u.a. ‚`decimal`', ‚`upper`'- ‚`lower-roman`' etc.). Die `target-counter()` können auch kombiniert als content-Wert auftreten. Der folgende Code (*80*) hätte bspw. diesen Output:[153]

(siehe Kap. 3.2 "Querverweise", S. 85)

```
content: '(siehe Kap. '
        target-counter(attr(href url), section, '.', decimal)
        ' " '
        target-text(attr(href url), content)
        ' " , S. '
        target-counter(attr(href url), page)
        ')'
```

Code 80: CSS Bsp. Verweiskombination; Götz/Ott, S.85

4.5.3.2 Inhaltsverzeichnis

Ohne Verweise wäre ein Inhaltsverzeichnis weniger nützlich, aber mit ihnen findet man einzelne Abschnitte schnell, da Seitenzahlen zur Verfügung stehen. D.h. es wird ein Inhaltsverzeichnis formatiert, das nummerierte Einträge hat und auf die Seitenzahlen des Inhalts referenziert.[154]

Da dies nicht nur durch Verweise entsteht, wird hier die Verwendung von Verweisen in Kombination mit Listen oder Tabellen und mit typischen Führungslinien untersucht.

4.5.3.2.1 XSL-FO

Ein Inhaltsverzeichnis (IVZ) kann in XSL-FO aus dem Inhalt generiert werden, inklusive automatische Nummerierung der Einträge und die Seitenreferenzierung. Im Beispiel des XSL-FO Buchs von Montero und Krüger wird zur Erstellung des Inhaltsverzeichnisses das Tabellenkonzept benutzt, um das Inhaltsverzeichnis zu strukturieren (*s. Anhang: Code 81*).[155]

[152] W3C: CSS3GCPM, #target-counter
[153] Götz/Ott, S.85
[154] Köhler, S.160ff
[155] Montero/Krüger, S.184ff

Zuerst wird also eine ID für jeden Abschnitt generiert, auf den ***<fo:page-numer-citation>*** dann später referenziert. Dann wird an der gewünschten Stelle das IVZ erzeugt. Sollte kein leeres Element für das IVZ vorhanden sein, so muss die Generierung in einer Seitenfolge (<***fo:page-sequence>***) und innerhalb von <***fo:flow>*** geschehen, ansonsten im passenden Elementaufruf („template"). Zunächst wird eine Überschrift für das IVZ erstellt und danach für jeden Abschnitt eine Tabellenzeile mit zwei Spalten erstellt. Die erste Spalte ist für die Nummerierung der IVZ-Einträge und die zweite für die Abschnittsüberschrift vorgesehen. So sichert die Tabellenstruktur, dass die Nummerierungen unabhängig von den Überschriften gleichmäßig angezeigt werden und auch die Überschriften können bei einem Zeilenumbruch einheitlich links-aligniert dargestellt werden.

Wie bereits erwähnt, kann FO keine automatische Zählung generieren, deshalb wird wieder das Element <***xsl:number>*** verwendet, um die IVZ-Überschriften zu nummerieren. In der zweiten Spalte wird zum einen der momentane Titelinhalt des Abschnitts eingefügt, zum anderen ein Inlineelement mit <***fo:leader>***, das die Übersichtlichkeit unterstützt . Außerdem wird zum Schluss mit <***fo:page-number-citation>*** die Zahl der Seite eingebaut, in der der referenzierte Abschnitt mit der entsprechenden ID beginnt. Theoretisch könnte man die Seitenzahl auch in eine dritte Spalte einfügen, damit diese auch unabhängig gestaltbar bleibt.

Durch den Formatierer wird die Referenzbeziehung des Verweises durch das Einfügen einer Seitenzahl aufgelöst. Durch den um die Tabelle eingefügten Block und das Attribut `text-align-last="justify"` wird die horizontale Verteilung der letzten Spalten ausgeglichen (dank Vererbung vom Block an seine Unterstrukur, die Tabelle), sodass die Seitenzahl an den rechten Satzspiegelrand gesetzt wird.[156]

4.5.3.2.2 CSS [CSS3GCPM, CSS3CST]

In CSS bietet sich die Tabellenform ebenfalls zur Inhaltsverzeichnis-Formatierung an. Im Vorprozess wurde mit XSLT ebenso für jeden Abschnitt bzw. jedes Objekt (<***article>***, **[1]**) im HTML-Dokument ein vorhandener, einzigartiger Wert als ID festgelegt und eine Tabelle erstellt mit einer generierten Nummerierung in der ersten Spalte bzw. dem „table-header" (**[2]**). Die zweite Spalte enthält die jeweilige Abschnittsüberschrift und die dritte ein leeres Linkelement, das für die Seitenzahlenerstellung nützlich ist, mit einem entsprechenden Referenzattribut und seinem ID-Wert (**[3]**).

[156] Montero/Krüger, S.186

```
<section id="ivz">
      <h2>Inhaltsverzeichnis</h2>
      <table>
        <tbody>
          <tr>
            <th>1.</th>        [1]
            <td class="ivz-text">Ecos</td>
            <td class="ivz-link">
                  <a href="#1031896554"/>
                </td>          [2]
          </tr>
...
</section>
...
  <article id="1031896554">          [3]
        ...
  </article>
```

Code 82: HTML Inhaltsverzeichnis

Nun kann die Nummerierung vertikal am oberen Zellenrand ausgerichtet werden, damit sie auch bei umbrechenden Überschriften in der nächsten Zelle vor der „ersten Zeile“ steht (**[4]**). Hinter die Zelle des Überschriftentexts kann mit der content-Eigenschaft und der `leader()`-Funktion eine Hilfslinie eingefügt werden (**[5]**).

Anstelle des leeren Linkelements wird, ebenfalls mit der content-Eigenschaft, die Seitenzahl des Verweisziels gezeigt. Diese wird mit der *`target-counter()`*-Funktion und den Argumenten ‚*`attr(href url)`*‘ für die Zieladresse, und ‚*`page`*‘ für den aktuellen Wert des Zielseitenzählers, ausgewählt(**[6]**). Der Inhalt der letzten Zelle wird dann noch am unteren Zeilenrand ausgerichtet, damit die Seitenzahl auch bei umbrechenden Überschriften in der vorherigen Zelle vor der „letzten Zeile“ steht (**[7]**). Zusätzlich könnte man das Schriftbild noch nach Belieben mit den üblichen Textattributen gestalten.

```
#ivz table > tbody> tr th {
   vertical-align:top;          [4]
   font-weight-bold;
}
#ivz table > tbody> tr> td.ivz-text::after {
   content:leader(dotted)' ';                    [5]
}
#ivz table > tbody> tr> td.ivz-text{
   text-align:justify;
}
#ivz table > tbody> tr> td.ivz-link > a {
   content:target-counter(attr(href url), page);          [6]
}
#ivz table > tbody> tr> td.ivz-link{
   text-align:right;
   vertical-align:bottom;                [7]
   width:5mm;
}
```

Code 83: CSS Inhaltsverzeichnis

Wie bereits bemängelt, endet die horizontale Trenn- bzw. Hilfslinie nicht immer bündig mit der Randkante des nächsten Elements, so auch hier. Auch mit unterschiedlichen Breiteneinstellungen der Elemente und Veränderungen der Linie selbst ist keine Verbesserung zu bemerken.

4.5.3.3 Register

Stichwortverzeichnisse oder andere Register sind (meist nach Alphabet) sortierte Namens- oder Sachverzeichnisse, die direkt hinter den Begriffen auf die entsprechenden Seiten des Zielverweises referenzieren. Bei der Generierung eines Registers kommt meistens eine Kombination von Mehrspaltenlayout, Textmodifikation und Verweisen zur Anwendung, um ein erwünschtes Layout zu erreichen.

4.5.3.3.1 XSL-FO

XSL-FO Sachregister gestalten sich ein wenig anders, als die allgemeinen FO-Verweise. Für ein Sachregister werden zuerst mindestens zwei Seitenvorlagen erstellt, eine für den Hauptinhalt und eine für das Register (*s. Anhang: Code 84*). Der ***<fo:region-body>*** der Registerseitenvorlage wird mehrspaltig formatiert und kann weiter modifiziert werden, durch die Attribute `column-count,-gap` und die AHF-Erweiterungen für die Trennlinie.

Auch hier gilt, wenn kein leeres Element im Dokument für ein Sachregister existiert, so muss der Inhalt in eine Seitenfolge, genauer in ein ***<fo:flow>***–Element übertragen werden. Innerhalb der Seitenfolge der Hauptinhaltsseiten werden die erwünschten Stichworte oder Elemente mit einem `index-key`-Attribut ausgestattet, sodass jedes Auftreten markiert und mit gleichartigen Indexschlüsseln gruppiert wird.

In der Register-Seitenfolge wird innerhalb eines ***<fo:block>***–Elements für jeden gewünschten Registereintrag das Stichwort selektiert und eingetragen (entweder mithilfe von XSLT oder per Hand). Mit ***<fo:index-page-citation-list>*** und dem darin befindlichen ***<fo:index-key-reference>***-Element lassen sich die Seitenzahlen aller Vorkommen der festgelegten Indexschlüssel einfügen. Mit ***<fo:index-key-reference>*** werden die jeweiligen `index-key`-Werte der Stichworte von der Hauptseitenfolge aufgegriffen. Block für Block entstehen so die Registereinträge mit Seitenverweisen.[157]

Mit der Kombination des `start-indent` und negativem `text-indent` kann ein hängender Einzug bestimmt werden, damit sich das Stichwort von den Seitenzahlen abhebt, auch wenn diese über mehrere Zeilen gehen. Oder man generiert sich eine „unsichtbare“ Hilslinie (`<fo:leader`

[157] Montero/Krüger, S.189f

`leader-pattern="space">`), die nur aus Leerzeichen anstatt Punkten o.ä. besteht, zwischen Stichwort und Seitenzahlen.

4.5.3.3.2 CSS [CSS3GCPM, CSS3COL]

Das, was XSL-FO mit Hilfe von XSLT-Transformationen erreicht, muss auch hier durch den Vorprozess erledigt werden (*s. Anhang: Code 85*). Für jedes Stichwort wird ein Absatzelement in einen Registerabschnitt (**`<section>`**) eingefügt und Duplikate werden aussortiert. Die jeweiligen Seitenreferenzen werden in Form von ID-Attributwerten in leeren Linkelementen in diesem Absatz hinter dem Stichwort abgespeichert.

Nun können mit CSS auch hier wieder die Linkelemente durch die Seitenzahl des Verweisziels ersetzt werden (**[1]**), gefolgt von einem Komma – außer es ist das letzte Linkelement, dann wird das Komma weggelassen (**[2]**). Auch die Mehrspaltigkeit für den Registerabschnitt kann deklariert werden (**[3]**). Die Überschrift soll am besten alle Spalten überspannen und nicht nur am Anfang einer einzigen Spalte stehen. Damit die oft langen Absätze nicht unansehnlich über die Seiten und Spalten umbrochen werden, kann man mit `break-inside:avoid` den Block beisammenhalten (**[4]**). Auch nach der Überschrift kann eine umbruchverhindernde Eigenschaft angewandt werden, sodass die Überschrift nicht fälschlicherweise ohne den dazugehörigen Inhalt auf einer Seite steht (**[5]**).

Wie auch mit XSL-FO sollte mit CSS ein hängender Einzug gestaltet werden, damit man die Stichworte leichter überschauen kann (**[6]**). Wenn man zwischen Stichwort und erstem Linkelement noch einmal einen Leerraum einsetzen möchte, dann wählt man das Pseudoelement des Platzes vor dem ersten Linkelement aus (**[7]**) und fügt einfach mit `content` einen Unicode für den passenden Leerraum ein. Auch hier steht es jedem frei, den Text noch weiter zu gestalten.

```
#register p > a{
   content: 'S.'target-counter(attr(href url),page)', ';          [1]
   display:inline;
   hyphens:auto;
   text-decoration:none;
}
#register p>a:first-child::before{                    [7]
   content: '\2002';
}
#register p > a:last-of-type{                         [2]
   content: 'S.'target-counter(attr(href url),page);
   display:inline;
   hyphens:auto;
}
```

```
#register{
   column-count:2;                    [3]
   column-gap: 4mm;
   column-rule-style:solid;
   column-rule-width:1pt;
```

```
   column-fill:balance;
}
#register h2{
   column-span:all;
   break-after:avoid;                [5]
}
#register p{
   display:block;
   break-inside:avoid;               [4]
   margin-left:2mm;                  [6]
   text-indent:-2mm;                 [6]
}
```

Code 86: CSS Register

4.5.3.4 Zwischenfazit

Generierte Inhalte sind bei CSS selten, deshalb muss das meiste, das eine Verweiserstellung betrifft, mit XSLT vorprozessiert werden. Und auch wenn XSL-FO im Prinzip ebenso auf XSLT zurückgreift, ist es doch praktischer, Verweiskonstruktionen mit FO zu erstellen. Denn man muss nur ein Stylesheet anpassen und sieht direkt die Zusammenarbeit zwischen inhaltlicher Transformation und Layouterstellung. XSL-FO hat außerdem den Vorteil, dass es XPath benutzen kann, z.B. um gezielt Textwerte oder andere das Verweisziel umgebende Werte einzutragen.

Mit CSS Verweise zu gestalten, beschränkt sich auf die Auflösung der Verlinkung zu Inhaltstext oder Seitenzahl bzw. den Wert eines anderen Zählers, falls vorhanden. In einem digitalen PDF kann man dann durch Klicken auch von Verweisursprung zu Verweisziel springen (Hyperlink).

Der CSS-Code erscheint hier zwar einfach zu verstehen, allerdings kommt man auch hier nicht um viel Schreibaufwand herum, wenn man jeden Sonderfall abdecken möchte.

4.5.4 Exkurs: Lesezeichen

Wenn es um die Erstellung von PDF geht, sind Lesezeichen ein wichtiges Thema, auch wenn sie für das gedruckte Layout selbst nicht relevant sind. Damit kann man das PDF-Dokument gut überblicken und darin navigieren. Dabei hat man die Möglichkeit, verschiedene Hierarchieebenen zu deklarieren, um auch Unterkapitel erreichen/darstellen zu können. Z.B. PDF-basierte E-Books arbeiten viel mit den Lesezeicheneinstellungen, um dem Leser die Möglichkeit zu geben, direkt zur gewünschten Seite zu springen und die aktuelle Seite abzuspeichern.

4.5.4.1 XSL-FO

Mit XSL-FO werden Lesezeichen in einem Lesezeichenbaum (**`<fo:bookmark-tree>`**) nach dem **`<fo:layout-master-set>`** erstellt (*s. Anhang: Code 87*). Darin werden **`<fo:bookmark>`**-Elemente deklariert, also Lesezeichen, die auf externe Adressen (`external-destination`), z.B. eine Webadresse, oder auf interne IDs (`internal-destination`) verweisen. Eine solche ID ist der Verweisursprung, der auf ein Verweisziel-Element (z.B. **`<fo:block>`**) mit entsprechendem

`id`-Wert im fließbaren Inhaltsbereich (***<fo:flow>***) der Seitenfolge (***<fo:page-sequence>***) referenziert. In den Lesezeichen-Elementen können dann Titel (***<fo:bookmark-title>***) bestimmt werden, die die Lesezeichenbenennungen repräsentieren. Lesezeichen zweiter Ebene können in ein weiteres ***<fo:bookmark>***–Elemente verschachtelt werden.[158]

4.5.4.2 CSS [CSS3GCPM]

Das CSS3 - Konzept stellt Eigenschaften zur Lesezeichenerstellung bereit (*s. Anhang: Code 88*). Man kann für jeden Abschnitt mit `bookmark-level`, `-label` und `-state` ein Lesezeichen erstellen. Dabei wird die Hierarchieebene mit `bookmark-level`, die Lesezeichenüberschrift mit `bookmark-label` und der Anzeigemodus mit `bookmark-state` (Werte: *`open/closed`*) deklariert. Die Lesezeichenüberschrift lässt sich entweder direkt mit einem String-Wert, mit *`content`* (Inhalt des gewählten Elements) oder *`attr()`* bestimmen.[159]

Die Lesezeichenumsetzung mit CSS funktioniert gut (*s. Anhang: Abb. 45/46*). Ein Defizit ist dabei aber, dass die Lesezeichen nicht automatisch nummeriert werden können.

4.5.5 Vorbereitung Druck

Mit CSS lassen sich für den Druckprozess Anschnitt, Schnitt- und Passermarken, die Verwendung des CMYK-Farbraums und Auflösungswerte für Abbildungen erstellen. Dieser Abschnitt ist nur eine theoretische Betrachtung, denn zum korrekten Bewerten der Funktionalität wäre die Durchführung des Druckprozesses nötig.

Der Anschnittbereich umrahmt die Seite und ist nützlich für z.B. Bilder, die zu nah am Rand positioniert sind und deswegen vom Drucker nicht genau umgesetzt werden können.

4.5.5.1 XSL-FO

Im XSL-FO-Konzept gibt es nicht direkt Eigenschaften für Anschnittbereiche und Schnitt- bzw. Passermarken. AHF bietet dafür Erweiterungen – `axf:bleed` für den Bereich und `axf:printer-marks` (Werte: *`crop`*, *`cross`* und optionale Quellangabe des *`cross`*-Symbols) für die Marken. Außerdem können die Marken noch detaillierter beschrieben werden mit `axf:printer-marks-*`[160], `axf:crop-*`[161] und `axf:bleed-*`[162]. Diese Erweiterungen müssen dann in der Seitenformatvorlage deklariert werden (*s. Anhang: Code 89*). Für die Auflösung der Abbildungen

[158] Montero/Herkert, S.63-71
[159] Götz/Ott, S.91ff
[160] Montero/Herkert, S.3218ff
[161] Ebd., S.2808ff
[162] Ebd., S.2675ff

gibt es auch nur AHF-Erweiterungen. So können Grafikauflösungen mit `afx:image-resolution` bestimmt werden, bzw. Hintergrundbilder mit `afx:background-image-resolution`.[163]

Farbprofile lassen sich innerhalb ***<fo:declarations>*** mit ***<fo:color-profile>*** definieren und referenzieren, mit dem Attribut `src` wird auf extern gespeicherte Farbprofildaten verwiesen. Das gebräuchlichste RGB-Farbprofil ist auch ohne Deklaration verwendbar. Für einen anspruchsvolleren CMYK-Farbraum kann man allerdings ein solches Farbprofil erstellen (*s. Anhang: Code 90*). [164]

4.5.5.2 CSS [CSS3PAGE, CSS4COLOR, CSS3IMAGE]

CSS besitzt seit der dritten Version Eigenschaften für die Generierung von Anschnitt und entsprechenden Marken (*s. Anhang: Code 91*). Innerhalb der ***@page***-Regel kann mit `bleed` und einem Längenmaß ein Anschnittbereich definiert werden. Dieser wird bei *`auto`*-Wert mit 6pt umgesetzt, wenn Schnittmarken angelegt wurden. Der Bereich ist außerhalb der Seitenbox („page box“) und wird im Fertigungsprozess auch abgeschnitten. Der `marks`-Eigenschaft können die Werte *`crop`* (für Schnittmarken) und *`cross`* (für Passermarken) zugewiesen werden, oder *`none`*, falls keine dieser Marken gerendert werden sollen. In den Ecken befinden sich dann die Schnittmarken, die bestimmen, wo der Seitenrand geschnitten werden soll. Passermarken sind meist kreuzförmig, befinden sich außerhalb der Seitenkanten und sorgen für eine Seitenausrichtung während des Druckprozesses (*s. Anhang: Abb. 47*).[165]

Mit dem neuen Farbmodul CSS4COLOR kann der CMYK-Farbraum angelegt werden (*s. Anhang: Code 92*). Dieser ist in der Druckproduktion beliebt und lässt sich mit `color:`*`device-cmyk()`* bestimmen. Die Argumentreihenfolge ist die Angabe des Cyan-, Magenta-, Gelb- und Schwarzwerts als Nummer (0-1) oder Prozentwert (0-100%). Zusätzlich kann ein Farbkanal als fünftes Argument notiert werden.[166] CSS3 unterstützt übrigens neben RGB- und Hexadezimalfarben auch RGBA, HSL, HSLA und Deckkraftangaben.[167]

Die Auflösung der Grafiken ist mit der Eigenschaft `image-resolution` kontrollierbar. Die Auflösung der Quellressource kann mit dem Wert *`from-image`* zum Rendern verwendet werden. Durch einen *`dpi`*-Wert lässt sich aber auch manuell eine Auflösung angeben, wenn die Grafik keine Auflösung hinterlegt hat (*s. Anhang: Code 93*).[168]

---

[163] Ebd., S.3040 (axf:image-resolution)/ S.2664 (axf:background-image-resolution)
[164] Montero/Krüger, S.75f
[165] Götz/Ott, S.91
[166] Ebd., S.93
[167] W3Schools: „CSS3 Colors”
[168] W3C: CSS3IMAGE, #the-image-resolution

5 Lösungsmöglichkeiten und Ausblicke

In den letzten Jahren wurde das Thema CSS3 u.a. in der Verlagsbranche immer stärker diskutiert und es hat sich auch viel bei der Entwicklung der Sprache getan[169] – aber reicht der momentane Stand von CSS3 aus, um professionelle Printlayouts zu erstellen?

Welche Vorteile und welche Schranken hat CSS3 denn letztendlich zu bieten? Bestehen schon Möglichkeiten, wie man den Arbeitsprozess mit CSS3 verbessern oder erweitern könnte? Wie sieht die Zukunft von CSS aus? Auf diese Fragen soll im Folgenden eingegangen werden.

5.1 Vorteile und Schranken von CSS

Die Verwendung von CSS in diesem Bereich ist recht stark von dem gewünschten Produkt abhängig. Einerseits lassen sich einfache Buchstrukturen, Magazinartikel, interne Protokolle etc. ordentlich umsetzen, sodass eine professionelle Nutzung möglich ist. Auch der Vorteil der Zusammenarbeit von HTML und CSS, die gemeinsame Datenbasis für mehrere Output-Dokumente, sowie die Trennung von Inhalt, Struktur und Layout spricht für die CSS-Nutzung für die Printformatierung.

Andererseits wird immer ein Vorprozess von Nöten sein, um Daten aus z.B. einer Datenbank zu einem professionell-printfähigen Ausgabedokument zu machen. Kaum etwas lässt sich mit CSS generieren – Verzeichnisse, Register und Elementvertauschungen sind auf die externe Generierung angewiesen und auch die Seitenabfolge wird allein von der Inhaltsreihenfolge des Dokumentes bestimmt. Man könnte argumentieren, dass auch XSL-FO auf die Hilfe von XSLT zurückgreift, allerdings ist dieser Schritt vollautomatisiert und es bedarf nur der Anpassung eines Stylesheets, das die Zusammenarbeit von Strukturierung und Layoutierung deutlich macht. Mit XSLT, HTML und CSS muss bei jeder Änderung der strukturellen Ebene ein neues HTML-Dokument transformiert werden, um darauf dann weiter mit CSS arbeiten zu können.

Ein positiver Aspekt von CSS ist, dass es wirklich einfach zu erlernen ist – somit auch um einiges einfacher zu lesen und zu schreiben als das komplizierte XSL-FO, das eine längere Einarbeitungsphase benötigt. Dadurch, dass in CSS alles als Box gewertet wird, ist die Gestaltung der unterschiedlichen Elemente und der Seite selbst jeweils recht ähnlich, viele Eigenschaften lassen sich auch vielseitig anwenden. Aber durch das einfache Konzept von CSS wird es mit erhöhter Komplexität des Ausgangsdokuments oder der Umsetzungswünsche immer schwieriger, ein zufriedenstellendes Print-Dokument zu erstellen. Schnell wird der Code unübersichtlich, da

[169] U.a. W3C: CSS3PAGE, #changes

jeder kleinste Sonderfall eine neue Deklaration benötigt und nicht einfach mit einer if-Abfrage o.ä. abgehandelt werden kann.

Zum Thema Unübersichtlichkeit lässt sich auch ein wichtiger Mangel kritisieren: die Abwesenheit von Variablen. Wenn man bspw. ein Buch gestalten möchte, in diesem ein Farbschema in mehreren Elementen vertreten ist und dann das Farbschema verändert wird, dann muss man in jedem einzelnen betroffenen Element den Farbcode ändern, anstatt den Wert einer einzigen deklarierten Farbvariable umschreiben zu können. Dazu kommen noch Schriftarten und -größen u.v.m., was den Prozess der Buchlayoutierung sehr stark verlangsamt, nervenaufreibend macht und sich somit schneller Fehler einschleichen können.

XSL-FO liegt auch eindeutig vorne, was die Navigation durch das zu verarbeitende XML- bzw. HTML-Dokument betrifft. Das ist XPath zu verdanken, da diese Technologie viele Selektionsmöglichkeiten bietet, um auch wirklich genau das Element zu erreichen, das formatiert werden soll. CSS hat immerhin bereits Pseudoklassenselektoren eingefügt, die das Navigieren durch ein Dokument besser möglich machen und für viele Fälle ausreichend sind.

5.2 Bestehende Erweiterungsmöglichkeiten

Der Pagina-Satzvertrieb entwickelte Ideen, wie man mit Erweiterungen CSS-Layoutierung verbessern könnte, und verwendet diese bereits in Projekten. So ist das Problem der fehlenden Variablen mit einem less[170]- oder sass[171]-Vorprozess lösbar. Diese Erweiterungssprachen für CSS sorgen für neue Features, mit denen man Variablen deklarieren sowie weitere Funktionen verwenden kann. Ebenso erhält man durch diese weitere Bereicherungen für CSS-Stylesheets. In einem Vorprozess werden hier bspw. die im sass-/less-Stylesheet deklarierten Variablen aufgelöst und die Werte in die jeweiligen CSS-Deklarationen eingefügt.[172]

Um noch besser auf die Umbruchsteuerung eingehen zu können, sind PIs (*process instructions*) nützlich. PIs sind Steuerungsanweisungen für XML und SGML, die als Element angelegt werden und für spezielle Anwendungen spezielle Instruktionen speichert (z.B. Ausnahmen bei Umbruchregelungen), die diese dann lesen und verarbeiten können. [173]

5.3 Ausblick

Ein wichtiger Punkt bei der Weiterentwicklung wäre, dass kostenlose Renderer oder am besten Browser CSS3 für die Printanwendung mehr unterstützen. Dadurch würden mehr User die

[170] Offizielle Webseite zu *less*: http://lesscss.org/
[171] Offizielle Webseite zu *sass*: https://sass-lang.com/
[172] Vgl. Götz/Fischer, S.27
[173] Vgl. ebd., S.40

Eigenschaften und Funktionen testen und es würden Szenarien entstehen, auf die man eingehen könnte.

Die CSS-Module werden stetig weiterentwickelt und neue Konzepte ausprobiert.[174] So wird z.B. an einem Regionenkonzept gearbeitet, das dem des XSL-FO ähnelt. In diesem Konzept sollen auch sogenannte „Named Flows", also Seitenfolgen, zur Verfügung stehen, damit nicht nur die Seitenformatierung selbst, sondern auch die Kombination der Formatvorlagen modifiziert werden können.[175] Erweiterte Positionierungs[176]- und Alinierungskonzepte[177] sind ebenfalls als momentaner Arbeitsentwurf zur Entwicklung freigegeben. Relationale Selektoren u.v.m. sind auch bereits im Gespräch, damit man z.B. bei Elementselektionen Besitz- bzw. Existenzabfragen durchführen kann mit der Pseudoklasse *:has()* oder einer Selektoren-Disjunktion, d.h. entweder der eine oder der andere Selektor soll zutreffen, mit :*matches()*.[178]

6 Schlussfolgerung/Bilanz

CSS3 als Alternative zu XSL-FO? Jein.

Eine grundlegende Seitenstruktur, Marginalien und die wichtigsten Umbrüche lassen sich einfach einstellen, simple Seitenvorlagen sind auch gut umsetzbar. Doch werden die Seitenvorlagen komplexer oder sind stark verschachtelt, entsteht viel unübersichtlicher Code und es ist nicht gewährleistet, dass man mit den vorhandenen Selektoren auch immer genau die Seiten auswählen kann, die modifiziert werden sollen. Das fehlende Konzept zur Erstellung von Seitenfolgen ist ein Defizit, das CSS aufholen sollte (und laut W3C-Webseite auch in Arbeit ist).

Text und Absätze wiederum sind mit vielen funktionierenden Eigenschaften und Funktionen gestaltbar. Einige kleine Verbesserungen sind ratsam, aber nichts Gravierendes. Ein wichtiges Manko stellt allerdings die fehlende Variablennutzung dar. Deswegen müssen für eine professionelle Verwendung von CSS3 in der Printlayoutformatierung weitere Technologien herangezogen werden.

Tabellen und Listen umzusetzen ist bei der Verwendung einer HTML-Basis einfach und praktisch, solange auch diese Strukturen nicht zu komplex werden, da auch hier die genaue Selektion dann nicht mehr gewährleistet ist und sich dann ebenso eine Menge an Code ansammelt, um jedes Szenario bzw. jeden Einzelfall abzudecken.

---

[174] Mehr dazu siehe W3C: „CSS - aktuelle Arbeit"
[175] Mehr dazu siehe W3C: CSSREGIONS, #regions
[176] Mehr dazu siehe W3C: CSS3POSITION
[177] Mehr dazu siehe W3C: CSS3ALIGN
[178] Mehr dazu siehe W3C: CSS4SELECT, #relational, #matches

Grafiken bzw. Abbildungen sollten bereits alle ins Dokument eingefügt worden sein und man sollte diese nicht sehr verändern wollen, wenn man mit CSS3 Printlayout umsetzt. Die Handlungsmöglichkeiten sind zum einen u.a. durch die zu geringen Positionierungs- sowie Umflusseinstellungen und zum anderen durch fehlerhafte bzw. fehlende Implementierungen beschränkt.

Fußnoten lassen sich für durchschnittliche, einfache Anforderungen gut umsetzen, aber für weiteres ist der Wirkungsrahmen der Attribute zu gering. Randbemerkungen lassen sich zwar erstellen, aber diese sehen nicht professionell genug aus, um sie vollends nutzen zu können. Infoboxen oder andere ähnliche Marginalien sind nicht umsetzbar.

Wiederum gut umsetzen lassen sich lebende Kolumnentitel und Seitenzahlen, die mit entsprechenden Funktionen und Argumenten erstellt und angepasst werden können. Aber auch hier hat XSL-FO mehr nützliche Einstellungen zur exakteren Bestimmung des Verhaltens, welcher Kolumnentitel ausgewählt werden soll oder wo sich die Paginierung ändert (durch die Zusammenarbeit mit Seitenfolgen).

Generiert werden kann zwar kaum etwas automatisch mit CSS3, aber mit einem XSLT-Vorprozess (der indirekt ebenfalls bei XSL-FO vorhanden ist) können auch Strukturen wie Inhaltsverzeichnisse oder Sachregister ordentlich umgesetzt werden – wenn auch nicht genauso effizient wie mit XSL-FO. Zusätzlich lassen sich mit CSS Anschnittsbereiche und Marker erstellen sowie funktionierende Lesezeichen, die eine bedeutende Rolle bei der Verwendung des PDF-Outputs für E-Books spielen.

Gemeinhin reicht CSS3 noch lange nicht an die Mächtigkeit von XSL-FO heran, aber für einfache Layoutierungen, die z.B. schnell gehen müssen oder von einem Verlagsfremden kontrolliert und gewartet werden sollen, ist es eine gute Alternative. Sobald Browser die volle CSS3-Funktionalität unterstützen, die Szenarienstatistik erweitert wird und daran orientiert die Technologie weiterentwickelt wird, wird CSS sicher häufig zur Printlayoutformatierung von Dokumenten verwendet („digital first“-Verwendung). Momentan fallen außerdem hohe Kosten an, wenn man mit CSS3 arbeiten möchte, da für diese Zwecke bisher keine (guten) kostenlosen Formatierer (wie z.B. FOP) auf dem Markt zu finden sind.

Schlussendlich lässt sich sagen, dass CSS3 für die automatische Printlayoutformatierung noch verbesserungsbedürftig, aber auch ausbaufähig und zukunftstauglich ist.

7 Quellen

7.1 Printwerke

Götz, Christin, Ott, Tobias (Hrsg.) (2015): „Print CSS. Das CSS paged media Modul, Grundlagen und Referenz", pagina, Tübingen

Montero Pineda, Manuel, Krüger, Manfred (2004): *„XSL-FO in der Praxis"*, dpunkt, Heidelberg
- Online-Version: *https://www.data2type.de/xml-xslt-xslfo/xsl-fo/* (Stand: 22.03.17)

Montero Pineda, Manuel, Herkert, Steffen (2016): *„XSL-FO - Die Referenz"*, dpunkt, Heidelberg

Cole, Timothy W., Han, Myung-Ja K. (2013): „XML for Catalogers and Metadata Librarians", Libraries Unlimited, St.Barbara/Kalifornien (USA), Aufl. 1

Ahmed, Kal, Rivers-Moore, Daniel et al. (2001): „Professional XML Meta Data", Wrox Press, Birmingham/England

Welsh, Anna, Batley, Sue (2012): „Practical Cataloguing – AACR, RDA and MARC 21", Facet Publishing, London

Miller, Steven J. (2011): „Metadata for Digital Collections: A How-To-Do-It Manual", Neal-Schuman Publishers, New York/USA

Köhler, Ralf (2002): „Typo & Design - Grundlagen der Typografie", MITP-Verlag, Frechen

Meyer, Erica A. (2011): „CSS - kurz und gut", O'Reilly Verlag, Heidelberg, Aufl. 4

Wolf, Jürgen (2015): „HTML5 und CSS3 – Das umfassende Handbuch", Rheinwerk Verlag, Bonn

Stührenberg, Maik (2012): „Auszeichnungssprachen für linguistische Korpora - Theoretische Grundlagen, De-facto-Standards, Normen", unv. Diss., Universität Bielefeld, Fakultät für Linguistik und Literaturwissenschaft
- Online-Version: *https://pub.uni-bielefeld.de/publication/2492772* (Stand: 22.03.17)

7.2 Internetquellen

Hinweis:

1. Manchen Quellen wurden nach ihrer Ursprungsplattform gruppiert, mit Index ausgewiesen („AH", „W3C", *„W3Schools"*) sowie gemeinsame Verfasser und/oder Jahresangabe herausgezogen, für eine bessere Übersicht und um sie besser im Fußnotenbereich kennzeichnen zu können.
2. CSS-Module von W3C-Seiten werden mit der vorangestellten Abkürzung referenziert (z.B. **CSS3PAGE**). Ein Fußnotenverweis sieht dann z.B. so aus: „[1]*W3C: CSS3PAGE*".
3. In den Fußnoten auftretende weitere Verweise werden unter *„Fußnoten-Links"* mit entsprechender Fußnotenzahl aufgelistet.

Index **AH**, Antenna House (o.J.):

„CSS Conformance",
URL: *https://www.antennahouse.com/product/ahf64/ahf-css6.html*
(Stand: 22.03.17)

„Description of Hyphenologist",
URL: *https://www.antennahouse.com/hyphenation/descript.htm*
(Stand: 22.03.17)

„Float Extension",
URL: *https://www.antennahouse.com/product/ahf64/ahf-float.html*
(Stand: 22.03.17)
- #FootnoteCSS

„XSL/CSS Extensions",
URL: *https://www.antennahouse.com/product/ahf64/ahf-ext.html*
(Stand: 22.03.17)
- #axf.font-face
- #axf.font-variant
- #css3

Index **W3C**, World Wide Web Consortium:

„2. Cascading Style Sheets (CSS) — The Official Definition" in „CSS Snapshot 2017, W3C Working Group Note" (2017),
URL: *https://www.w3.org/TR/css3-roadmap/#css* (Stand: 22.03.17)

„6 Formatting Objects" in „Extensible Stylesheet Language (XSL) Version 1.1, W3C Recommendation" (2006),
URL: *https://www.w3.org/TR/xsl/#fo-section* (Stand: 22.03.17)

„6.10 Formatting Objects for Indexing" in „Extensible Stylesheet Language (XSL) Version 1.1, W3C Recommendation" (2006),
URL: *http://www.w3.org/TR/xsl/#d0e13293* (Stand: 22.03.17)

„6.11 Formatting Objects for Bookmarks" in „Extensible Stylesheet Language (XSL) Version 1.1, W3C Recommendation" (2006),
URL: *https://www.w3.org/TR/xsl/#d0e14206* (Stand: 22.03.17)

„CSS - aktuelle Arbeit" (2017),
URL: *http://www.w3.org/Style/CSS/current-work* (Stand: 22.03.17)

CSS2.2VIS/ „9 Visual formatting model" in „Cascading Style Sheets Level 2 Revision 2 (CSS2.2) Specification" (2016),
URL: *http://www.w3.org/TR/CSS22/visuren.html* (Stand: 22.03.17)
- #propdef-display

CSSREGIONS/ „CSS Regions Module Level 1" (2014),
URL: *https://www.w3.org/TR/css-regions-1/* (Stand: 22.03.17)
- #regions

CSS3ALIGN/ „CSS Box Alignment Module Level 3“ (2017),
URL: *http://www.w3.org/TR/2017/WD-css-align-3-20170215/*
(Stand: 22.03.17)

CSS4SELECT/ „Selectors Level 4“ (2017),
URL: *https://drafts.csswg.org/selectors-4/*
- #relational
- #matches

CSS3BG/ „Backgrounds and Borders Module Level 3” (2014),
URL: *http://www.w3.org/TR/css3-background/* (Stand: 22.03.17)
- #the-background-image

CSS3BREAK/ „CSS Fragmentation Module Level 3” (2017),
URL: *http://www.w3.org/TR/css-break-3/* (Stand: 22.03.17)

CSS3CASC/ „Cascading and Inheritance Level3” (2016),
URL: *http://www.w3.org/TR/css3-cascade/* (Stand: 22.03.17)
- #at-import

CSS3COL/ „CSS Multi-column Layout Module” (2011),
URL: *http://www.w3.org/TR/css3-multicol/* (Stand: 22.03.17)
- #column-fill

CSS3FLOATS/ „CSS Page Floats” (2015),
URL: *https://www.w3.org/TR/2015/WD-css-page-floats-3-20150915/* (Stand: 22.03.17)

CSS3FONTS/ „CSS Font Module Level 3” (2013),
URL: *http://www.w3.org/TR/css-fonts-3/* (Stand: 22.03.17)
- #font-kerning-prop
- #font-stretch-prop

CSS3GCPM/ „CSS Generated Content for Paged Media Module” (2014),
URL: *https://www.w3.org/TR/css-gcpm-3/* (Stand: 22.03.17)
- #future
- #target-text
- #target-counter
- #footnotes
- #leaders

CSS3IMAGES/ „CSS Image Values and Replaced Content Module Level 3” (2012),
URL: *https://www.w3.org/TR/2012/CR-css3-images-20120417/*
(Stand: 22.03.17)
- #image-values

CSS3LIST/ „CSS Lists and Counters Module Level 3” (2014),
URL: *http://www.w3.org/TR/css-lists-3/* (Stand: 22.03.17)
- #marker-pseudo-element
- #nested-counters

CSS3PAGE/ „CSS Paged Media Module Level 3" (2013),
URL: *https://www.w3.org/TR/css3-page/* (Stand: 22.03.17)
- #page-selector-and-context
- #using-named-pages
- #blank-pseudo
- #margin-boxes

CSS3POSITION/ „CSS Positioned Layout Module Level 3" (2016),
URL: http://www.w3.org/TR/css-position-3/ (Stand: 22.03.17)

CSS3SELECT/ „Selectors Level 3" (2011),
URL: *http://www.w3.org/TR/css3-selectors/* (Stand: 22.03.17)

CSS3SPEECH/ „CSS Speech Module" (2012),
URL: *https://www.w3.org/TR/css3-speech/* (Stand: 22.03.17)

CSS3TABLE/ „CSS Table Module Level 3" (2017),
URL: http://www.w3.org/TR/css-tables-3/ (Stand: 22.03.17)

CSS3TDECO/ „CSS Text Decoration Module Level 3" (2013),
URL: *http://www.w3.org/TR/css-text-decor-3/* (Stand: 22.03.17)

CSS3TEXT/ „CSS Text Module Level 3" (2013),
URL: *http://www.w3.org/TR/css-text-3* (Stand: 22.03.17)
- #text-align-property
- #text-align-last
- #word-break-property
- #line-break-property
- #text-justify-property
- #text-indent-property
- #hanging-punctuation-property
- #text-transform-property
- #page-based-counters
- #page-model
- #page-size

Index **W3Schools**, o.V., o.J.:

„HTML5 Introduction" in „W3Schools, HTML5 Tutorial",
URL: *http://www.w3schools.com/html/html5_intro.asp* (Stand: 22.03.17)

„XSLT Introduction" in: „W3Schools, XSLT Tutorial",
URL: *http://www.w3schools.com/xml/xsl_intro.asp* (Stand: 22.03.17)

„CSS Introduction" in „W3Schools, CSS Tutorial",
URL: *http://www.w3schools.com/css/css_intro.asp* (Stand: 22.03.17)

„CSS3 Introduction" in „W3Schools, CSS Tutorial",
URL: *http://www.w3schools.com/css/css3_intro.asp* (Stand: 22.03.17)

„CSS Lists" in „W3Schools, CSS Tutorial",
URL: *http://www.w3schools.com/css/css_list.asp* (Stand: 22.03.17)

„CSS3 Rounded Corners" in „W3Schools, CSS Tutorial",
URL: *https://www.w3schools.com/css/css3_borders.asp* (Stand: 22.03.17)

„CSS margins" in „W3Schools, CSS Tutorial",
URL: *https://www.w3schools.com/Css/css_margin.asp* (Stand: 22.03.17)

„CSS3 Colors" in „W3Schools, CSS Tutorial",
URL: *https://www.w3schools.com/css/css3_colors.asp* (Stand: 22.03.17)

Beinert, Wolfgang (2017): „Kolumnentitel" in „Typolexikon.de, Lexikon der europäischen Typographie",
URL: *http://www.typolexikon.de/kolumnentitel* (Stand: 22.03.17)

Beinert, Wolfgang (2017): „Schriftlaufweite" in „Typolexikon.de, Lexikon der europäischen Typographie",
URL: *http://www.typolexikon.de/schriftlaufweite* (Stand: 22.03.17)

Beinert, Wolfgang (2015): „Sperren" in „Typolexikon.de, Lexikon der europäischen Typographie",
URL: *http://www.typolexikon.de/sperren* (Stand: 22.03.17)

CLARIN (o.J.): "Component Metadata",
URL: *https://www.clarin.eu/content/component-metadata* (Stand: 22.03.17)

DCMI (Dublin Core Metadata Initiative) (2012): „Dublin Core Metadata Element Set, Version 1.1",
URL: *http://www.dublincore.org/documents/dces/* (Stand: 22.03.17)

Götz, Christin, Fischer, Tobias (2016): „Print CSS in der Praxis", Vortragsfolien, XUGS (XML User Group), Stuttgart,
URL: *https://www.pagina.gmbh/slides/2016-12-07_PrintCSS_XUGS_Goetz-Fischer.pdf* (Stand: 22.03.17)

Heller, Stephan (2011): „CSS3 im Praxistest: Multi-column Layout",
URL: *http://webkrauts.de/artikel/2011/css3-im-praxistest-multi-column-layout* (Stand: 22.03.17)

IMDI (ISLE Metadata Initiative) (2009): „Metadata Elements for Session Descriptions",
URL: *https://tla.mpi.nl/wp-content/uploads/2012/06/IMDI_MetaData_3.0.4.pdf* (Stand: 22.03.17)

IMDI (ISLE Metadata Initiative) (2009): „Metadata Elements for Catalogue Descriptions",
URL: *https://tla.mpi.nl/wp-content/uploads/2012/06/IMDI_Catalogue_3.0.0.pdf* (Stand: 22.03.17)

Kleinfeld, Sanders (2013): „HTML5 is the Future of Book Authorship", in: „O'Reilly Radar",
URL: *http://radar.oreilly.com/2013/09/html5-is-the-future-of-book-authorship.html* (Stand: 22.03.17)

Lie, Håkon W. (1994): „Cascading HTML style sheets -- a proposal",
URL: https://www.w3.org/People/howcome/p/cascade.html (Stand: 22.03.17)

McKesson, Nellie (2012): „Building Books with CSS3",
URL: *http://alistapart.com/article/building-books-with-css3* (Stand: 22.03.17)

Montero Pineda, Manuel (2004): „Schnellstart" in: „Einführung in XSLT 1.0",
URL: *https://www.data2type.de/xml-xslt-xslfo/xslt/xslt-einfuehrung/schnellstart/*
(Stand: 22.03.17)

O.V. (2004): „Vorteile von XML" in: „Microsoft API- und Referenzkatalog",
URL: *https://msdn.microsoft.com/de-de/library/cc431269.aspx* (Stand: 22.03.17)

O.V. (o.J.): „Ohne sie geht nichts: Medienneutrale Daten" in: „PAGINA - Das Kompendium",
URL: *https://www.pagina.gmbh/xml-hintergruende/pagina-das-kompendium/themenkomplex-i-cross-media/ohne-sie-geht-nichts-medienneutrale-daten/* (Stand: 22.03.17)

O.V. (o.J.): „Was ist Adobe PDF?",
URL: *https://acrobat.adobe.com/de/de/why-adobe/about-adobe-pdf.html*
(Stand: 22.03.17)

TEI (Text Encoding Initiative) (2015): „TEI Guidelines",
URL: *http://www.tei-c.org/Guidelines/* (Stand: 22.03.17)

TEI (Text Encoding Initiative) (2016): „TEI: Text Encoding Initiative",
URL: *http://www.tei-c.org/index.xml* (Stand: 22.03.17)

TEI (Text Encoding Initiative) (2016): „P5: Guidelines for Electronic Text Encoding and Interchange",
URL: *http://www.tei-c.org/release/doc/tei-p5-doc/en/html/* (Stand: 22.03.17)
- "10 Manuscript Description", *MS.html*
- "2 The TEI Header", *HD.html*

TUG (Tex User Group) (o.J.): „TeX Hyphenation Patterns",
URL: *http://tug.org/tex-hyphen/#introduction* (Stand: 22.03.17)

Fußnoten-Links

[14] DNB Metadatenshop, URL: *https://portal.dnb.de/metadataShop.htm* (Stand: 22.03.17)

[15] Offizielle Webseite der DCMI, URL: *http://dublincore.org* (Stand: 22.03.17)

[32] Offizielle Webseite von ISOcat, URL: *http://www.isocat.org/* (Stand: 22.03.17)

[52] Apache FOP, URL: *https://xmlgraphics.apache.org/fop/* (Stand: 22.03.17)

[54] AHF kostenlose Testversion,
URL: *https://www.antennahouse.com/product/ahf60/download.htm* (Stand: 22.03.17)

[105] Google Fonts, URL: *https://fonts.google.com/* (Stand: 22.03.17)

[169] Offizielle Webseite zu less: *http://lesscss.org/* (Stand: 22.03.17)

[170] Offizielle Webseite zu sass: *https://sass-lang.com/* (Stand: 22.03.17)

7.3 Abbildungsquellen

Abb. 1: Montero/Krüger, S.9

Abb. 2: Montero/Krüger, S.68

Abb. 3: Montero/Krüger, S.70

Abb.4: W3C: CSS3PAGE, #page-modell

Abb.13: https://developer.mozilla.org/en-US/docs/Web/CSS/font-kerning

Abb. 20: Montero/Krüger, S.106

Abb. 22: Montero/Krüger, S.116

Abb. 23: http://wiki.selfhtml.org/images/0/09/Tabelle-1.svg

8 Anhang

8.1 Codes

Code 1: MARC-XML-Ausschnitt; (dnb_data.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://www.loc.gov/MARC21/slim">
 <record type="Bibliographic">
  <leader>00000nas a2200000 c 4500</leader>
  <controlfield tag="001">1031896554</controlfield>
  <controlfield tag="003">DE-101</controlfield>
        ...
  <datafield tag="015" ind1=" " ind2=" ">
   <subfield code="a">13,A16</subfield>
   <subfield code="2">dnb</subfield>
  </datafield>
        ...
  <datafield tag="245" ind1="0" ind2="0">
   <subfield code="a">Ecos</subfield>
   <subfield code="b">die Welt auf Spanisch</subfield>
   <subfield code="n">[...]</subfield>
   <subfield code="p">Plus : der Sprachtrainer für Spanisch</subfield>
  </datafield>
        ...
 </record>
        ...
</collection>
```

Code 2: Grundaufbau XSLT-Stylesheet für HTML-Output; (angelehnt an Montero/Krüger, S.21)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" mlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="iso-8859-1"/>
  <xsl:template match="/">
        <html>
          <body>
            <p><xsl:apply-templates/></p>
          </body>
        </html>
  </xsl:template>
  <xsl:template match="title">
      <h1><xsl:apply-templates/></h1>
  </xsl:template>
...
</xsl:stylesheet>
```

Code 17: XSL-FO Seitenvorlagen Doppelseiten; (Ausschnitt aus Montero/Krüger, S.191)

```
<fo:layout-master-set>
  <fo:simple-page-master master-name="PageMaster.Inhalt-rechts"
   margin="15mm 25mm 30mm 13mm" page-height="297mm" page-width="210mm">
    ...
    <fo:region-body margin="25mm 0mm 15mm 0mm"/>
  </fo:simple-page-master>
  <fo:simple-page-master master-name="PageMaster.Inhalt-links"
   margin="15mm 13mm 30mm 25mm" page-height="297mm" page-width="210mm">
    ...
    <fo:region-body margin="5mm 0mm 7mm 0mm"/>
  </fo:simple-page-master>
  <fo:page-sequence-master master-name="Inhalt-Seiten">
    <fo:repeatable-page-master-alternatives>
      <fo:conditional-page-master-reference
        master-reference="PageMaster.Inhalt-rechts" odd-or-even="odd"/>
      <fo:conditional-page-master-reference
        master-reference="PageMaster.Inhalt-links" odd-or-even="even"/>
    </fo:repeatable-page-master-alternatives>
  </fo:page-sequence-master>
</fo:layout-master-set>
<fo:page-sequence master-reference="Inhalt-Seiten" >
    ...
  <fo:flow flow-name="xsl-region-body">
    <fo:block font-family="Arial" font-size="10pt">
      <xsl:apply-templates/>
    </fo:block>
  </fo:flow>
</fo:page-sequence>
```

Code 40: XSL-FO Listenaufbau; (Montero/Krüger, S.106f)

```
<fo:block>
  <fo:list-block>
    <fo:list-item>
      <fo:list-item-label>
        <fo:block>
          1.
        </fo:block>
      </fo:list-item-label>
      <fo:list-item-body>
        <fo:block>
          Listentext
        </fo:block>
      </fo:list-item-body>
    </fo:list-item>
        ...
  </fo:list-block>
</fo:block>
```

Code 41: HTML Listenaufbau

```
<h1>Ungeordnete Liste</h1>
<ul>
        <li>…</li>
        <li>…</li>
</ul>
<h1>Geordnete Liste</h1>
<ol>
        <li>…</li>
        <li>…</li>
</ol>
```

Code 46: XSL-FO Tabellenaufbau; (Ausschnitt aus Montero/Krüger, S.120)

```
<fo:block>
<fo:table width="110mm" border-style="ridge" border-
 width="5pt">
    <fo:table-body>
        <fo:table-row>
            <fo:table-cell width="40mm" border-style="solid" border-width="1pt">
                <fo:block> 1. Zelle</fo:block>
            </fo:table-cell>
            …
        </fo:table-row>
        <fo:table-row>
            <fo:table-cell border-style="solid" border-width="1pt">
                <fo:block> 1. Zelle</fo:block>
            </fo:table-cell>
            …
        </fo:table-row>
    </fo:table-body>
</fo:table>
</fo:block>
```

Code 49: CSS Tabelle Header und Footer; Renderergebnis siehe Anhang: Abb. 25/26

```
.control {
   border-collapse: collapse;
   -ah-table-omit-header-at-break:true;
   -ah-table-omit-footer-at-break:true;
}
.control thead  tr{
   background-color: grey;
   color: white;
}
.control tfoot tr{
   background-color: darkgrey;
   color: white;
}
```

Code 50: HTML Tabelle Titel; (CSS_dnb-BA.html)

```
<article id="1031896554">
        <table class="title">
          <tbody>
            <tr>
              <th class="t_number">I. </th>
              <td class="t_name">Ecos</td>
            </tr>
          </tbody>
        </table>
...
```

Code 56: CSS Hintergrundbild Seite (Wasserzeichen)

```
@page first:first{
   background-image: url(../Wasserzeichen_Kopie.gif);
   background-size: 100%;
   background-position:center;
}
```

Code 57: CSS Hintergrundbild Abschnitt

```
#first{
   ...
   background-image: url("../Muster-grau.jpg");
   background-size: 30%;
   background-repeat: auto;
}
```

Code 62: CSS Sidenote; Renderergebnis s. Anhang: Abb. 36

```
@page{
  size:A4 portrait;
  @sidenote{
     float: right;
     clear: both;
     width: 20%;
     text-align:justify;
  }
}
...
aside {
   float: sidenote;
   display:block;
   font-family:courier, monospace;
   font-size: 80%;
   padding:1mm;
   border:grey dotted 1pt;
   border-radius: 5mm;
}...
::sidenote-call {
  content: ' [' counter(sidenote) ']';
  vertical-align: baseline;
```

```
  font-size: 100%;
  line-height: inherit;
  break-before: avoid-line;
}
::sidenote-marker {
   content:'[' counter(sidenote) ']';
   list-style-position: outside;
}
```

Code 66: XSL-FO Kolumnentitel; (Montero/Krüger S.158)

```
<fo:page-sequence master-reference="PageMaster.Inhalt">
   <fo:static-content flow-name="xsl-region-before">
      <fo:block text-align="center" font-size="10pt" font-weight="bold">
        <fo:retrieve-marker retrieve-class-name="Ueberschrift-Ebene1"
        retrieve-position="first-including-carryover"
        retrieve-boundary="page-sequence"/>
      </fo:block>
      ...
   </fo:static-content>
</fo:page-sequence>
...
<xsl:template match="Abschnitt/Titel">
  <fo:block>
    <fo:marker marker-class-name="Ueberschrift-Ebene1">
      <xsl:number level="multiple" count="Abschnitt" format="1. "/>
      <xsl:value-of select="."/>
    </fo:marker>
  </fo:block>
</xsl:template>
...
```

Code 67: HTML Titelelemente; (CSS_dnb-BA.html)

```
<section id="auswahl">
       ...
       <article id="1031896554">
         <table class="title">
           <tbody>
             <tr>
               <th class="t_number">I. </th>
               <td class="t_name">Ecos</td>
             </tr>
           </tbody>
         </table>
...
```

Code 72: CSS Paginierung

```
@page:left{
   padding-left: 1cm;
   @top-left{
      content: counter(page);
   }
}
@page:right{
   padding-right: 1cm;
   @top-right{
      content: counter(page);
   }
}
```

Code 78: HTML Verweis

```
… <a href="#1031896554">Ecos</a> …
<article id="1031896554">…</article> …
```

Code 81: XSL-FO Inhaltsverzeichnis; (angelehnt an Montero/Krüger, S.184f)

```
<xsl:template match="Abschnitt">
  <fo:block id="{generate-id()}">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="Inhalt">
  <fo:block font-size="18pt" font-weight="bold" text-align="center" space-after="6mm">
    Inhaltsverzeichnis
  </fo:block>
  <xsl:for-each select="Abschnitt">
    <fo:block text-align-last="justify">
      <fo:table width="160mm">
        <fo:table-column column-number="1" column-width="10mm"/>
        <fo:table-column column-number="2" column-width="150mm"/>
        <fo:table-body>
          <fo:table-row>
            <fo:table-cell column-number="1">
              <fo:block>
                <xsl:number level="single" count="Abschnitt" format="1."/>
              </fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="2">
              <fo:block>
                <xsl:value-of select="Titel"/>
                <fo:inline><fo:leader leader-pattern="dots"/></fo:inline>
                <fo:page-number-citation ref-id="{generate-id()}"/>
              </fo:block>
            </fo:table-cell>
          </fo:table-row>
        </fo:table-body>
      </fo:table>
    </fo:block>
  </xsl:for-each>
</xsl:template>
…
```

Code 84: XSL-FO Register; (Montero/Herkert S.183ff)

```
<fo:root font-size="15pt" line-height="1.3">
    <fo:layout-master-set>
      <fo:simple-page-master master-name="site" page-height="210mm"
        page-width="148mm">
        <fo:region-body margin="20mm"/>
      </fo:simple-page-master>
      <fo:simple-page-master master-name="index" page-height="210mm"
        page-width="148mm">
        <fo:region-body margin="20mm" column-count="2"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="site">
      <fo:flow flow-name="xsl-region-body">
        <fo:block text-align="justify" hyphenate="true">
          <fo:block space-after="1em" id="AlsGregorSamsa">Als Gregor<fo:wrapper
           index-key="Gregor"/> Samsa<fo:wrapper index-key="Samsa"/> eines
           Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu
           einem ungeheueren Ungeziefer verwandelt. …
          </fo:block>
          …
          <fo:block space-after="1em">…
            Es war eine Kreatur des Chef<fo:wrapper index-key="Chef"/>s, ohne
            Rückgrat und Verstand. Wie nun, wenn er sich krank meldete? Das wäre
            aber äußerst peinlich und verdächtig, denn Gregor<fo:wrapper index-
            key="Gregor"/> war während seines fünfjährigen Dienstes noch nicht einmal
            krank gewesen. …
          </fo:block>
        </fo:block>
      </fo:flow>
    </fo:page-sequence>
    <fo:page-sequence master-reference="index">
      <fo:flow flow-name="xsl-region-body">
        <fo:block text-align="justify">
          <fo:block>Index
          </fo:block>
          <fo:block text-align-last="justify">Gregor <fo:leader leader-pattern="space"/>
            <fo:index-page-citation-list>
              <fo:index-key-reference ref-index-key="Gregor"/>
            </fo:index-page-citation-list>
          </fo:block>
          …
        </fo:block>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
```

Code 85: HTML Register; (CSS_dnb-BA.html)

```
<section id="register">
     <h2>REGISTER</h2>
     <p>Angewandte Linguistik
       <a href="#104216973X"></a><a href="#1024976548"></a>
       <a href="#1003859003"></a><a href="#1014028655"></a>
     </p>
…
```

Code 87: XSL-FO Lesezeichen; (Montero/Herkert, S.68)

```
<fo:root font-family="Arial" font-size="10pt" line-height="1.2em">
    <fo:layout-master-set>
      <fo:simple-page-master master-name="Inhalt-Seiten"
       margin="0mm 0mm 0mm 0mm" page-height="297mm"
       page-width="210mm">
        <fo:region-body margin="26mm 55mm 41mm 55mm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:bookmark-tree>
      <fo:bookmark internal-destination="id1">
        <fo:bookmark-title font-weight="bold">
            Übersicht zu XSL-FO
        </fo:bookmark-title>
        <fo:bookmark internal-destination="id2">
          <fo:bookmark-title font-style="italic">
              Einführung in die Sprache für
              Seitengestaltung und Umbruch
            </fo:bookmark-title>
        </fo:bookmark>
        <fo:bookmark external-
            destination="http://www.antennahouse.com/">
          <fo:bookmark-title>
              http://www.data2type.de.com/
            </fo:bookmark-title>
        </fo:bookmark>
      </fo:bookmark>
    </fo:bookmark-tree>
    <fo:page-sequence master-reference="Inhalt-Seiten">
      <fo:flow flow-name="xsl-region-body">
        <fo:block id="id1">
            Übersicht zu XSL-FO
          </fo:block>
        <fo:block id="id2">
            Einführung in die Sprache für Seitengestaltung
            und Umbruch
        </fo:block>
      </fo:flow>
    </fo:page-sequence>
</fo:root>
```

Code 88: CSS Lesezeichen; Renderergebnis s. Anhang: Abb.45/46

```
#welcome{
  bookmark-level:1;
  bookmark-label: 'Willkommen';
  bookmark-state: open;
}
...
.title{
  bookmark-level:2;
  bookmark-label:content;
  bookmark-state:closed;
}
```

Code 89: XSL-FO Anschnitt und Marker; (Ausschnitt aus Montero/Herkert, S.3218)

```
<fo:layout-master-set>
        <fo:simple-page-master … master-name="PageMaster1"
        axf:printer-marks="crop cross url(img/colorbar.svg)"
        axf:printer-marks-line-width="0.25pt"
        axf:crop-offset="20mm 14mm 20mm 14mm" axf:bleed="4mm">
        …
        </fo:simple-page-master>
</fo:layout-master-set>
```

Code 90: XSL-FO Farbprofil CMYK; (Montero/Krüger, S.75f)

```
<fo:declarations>
   <fo:color-profile src="..." color-profile-name="CMYK"/>
</fo:declarations>
```

Code 91: CSS Anschnitt und Marker, Renderergebnis s. Anhang: Abb. 47

```
@page{
  bleed: 6pt;
  marks:crop cross;
  …
}
```

Code 92: CSS Farbraum CMYK und Bildauflösung

```
img{
  color:device-cmyk(0%, 81%, 81%, 30%);
  image-resolution: from-image 300dpi;
}
```

8.2 Abbildungen

Abb. 1: XSL-FO Prozesse (Quelle: Montero/Krüger, S.9)

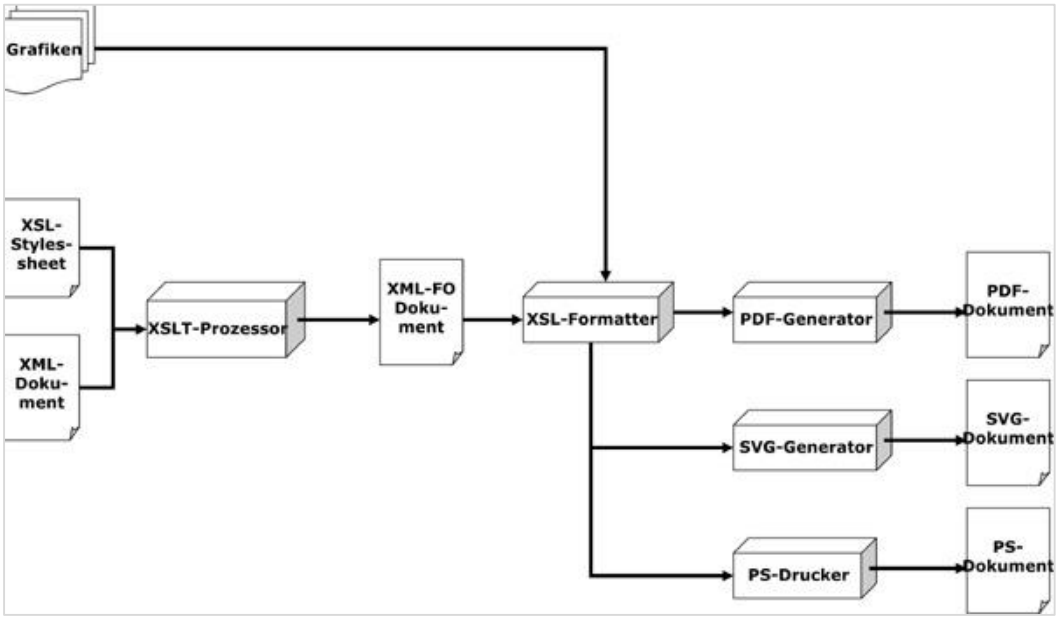


Abb. 2/3: XSL-FO Seitenkonzept Aufbau und Regionen (Quelle: Montero/Krüger, S.68/S.70)

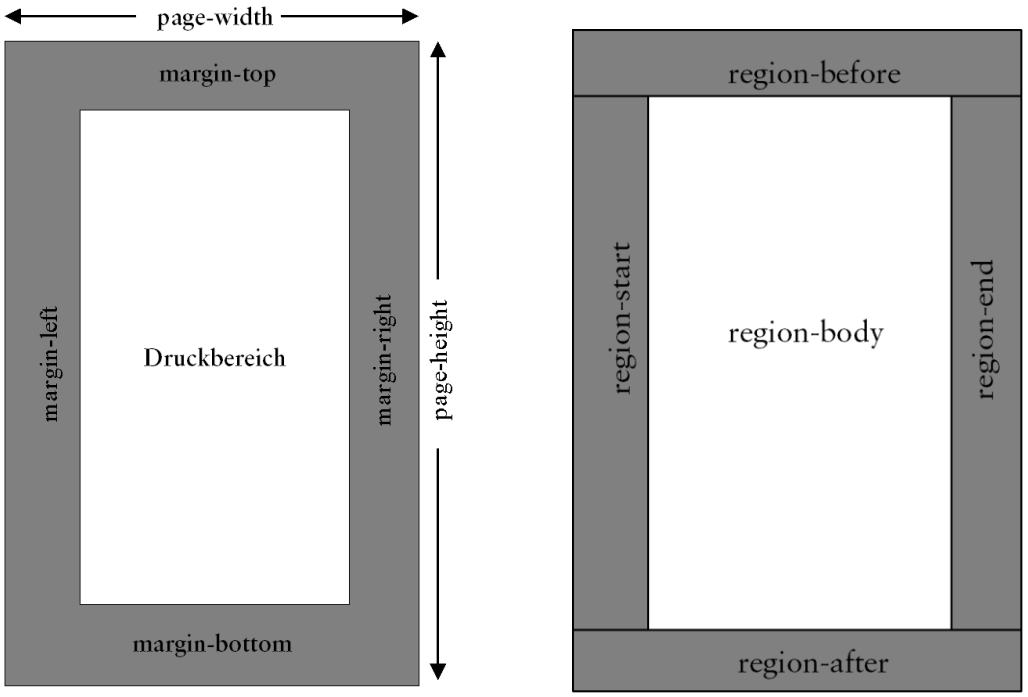


Abb. 4/5: CSS Seitenkonzept Aufbau (Quelle links: W3C: CSS3PAGE, #page-modell)

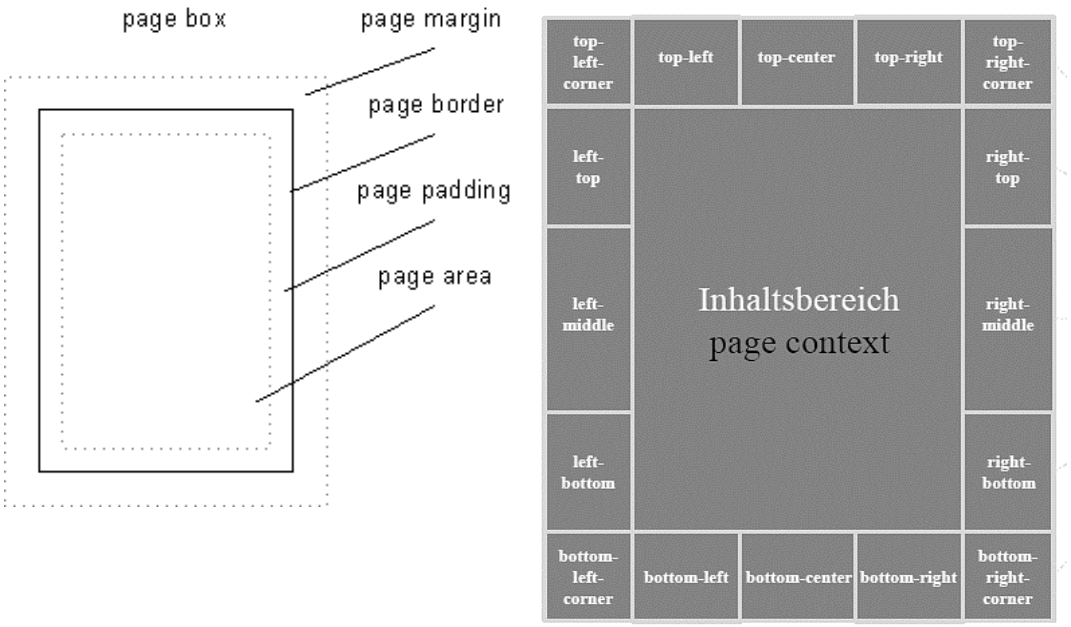
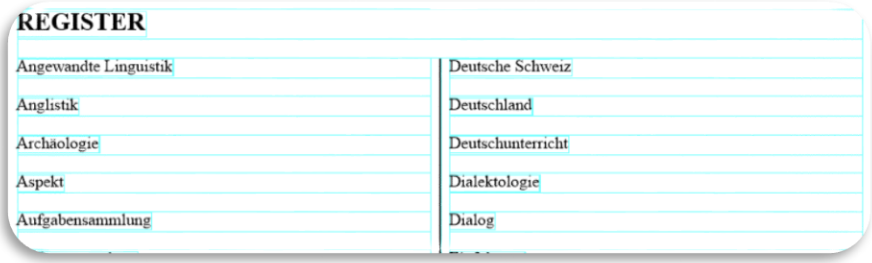


Abb. 6: AHF Renderergebnis Mehrspaltensatz



REGISTER

Angewandte Linguistik	Deutsche Schweiz
Anglistik	Deutschland
Archäologie	Deutschunterricht
Aspekt	Dialektologie
Aufgabensammlung	Dialog

Abb. 7: AHF Spaltenausgleich, column-fill:balance

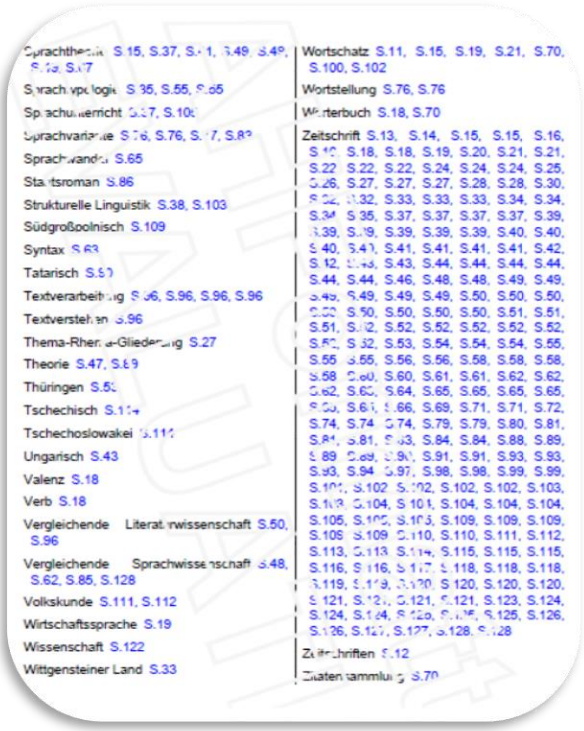


Abb. 8: AHF kein Spaltenausgleich, column-fill:auto

Abb. 9: AHF Renderergebnis (v.o.n.u.) text-align:start; text-align:end; text-align-last:end

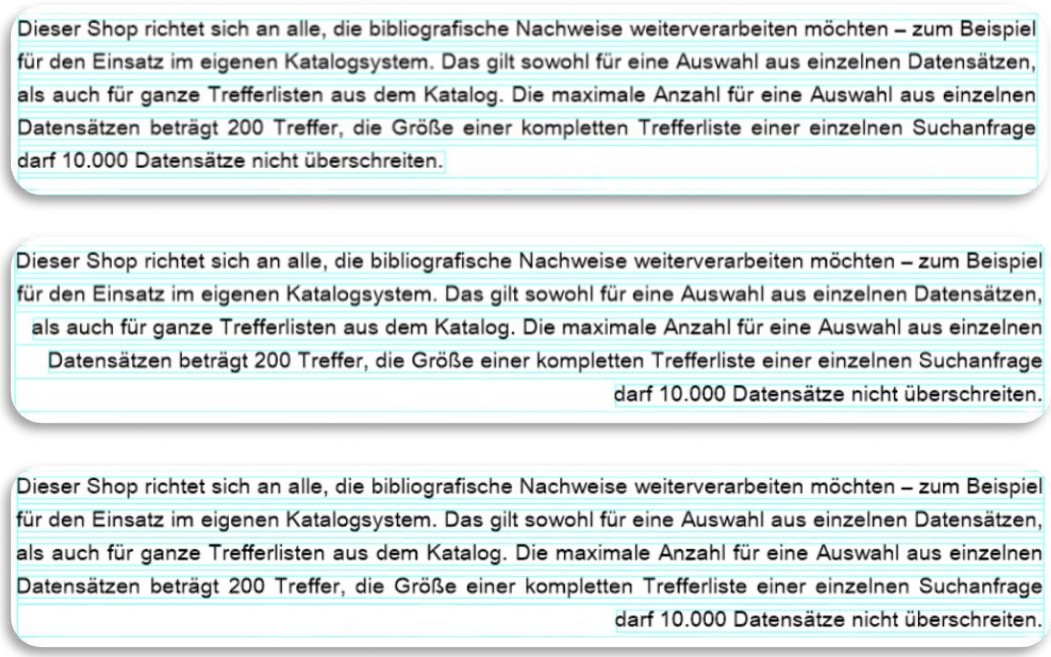


Abb. 11: AHF Renderergebnis hanging punctuation (l: none, r: force-end)

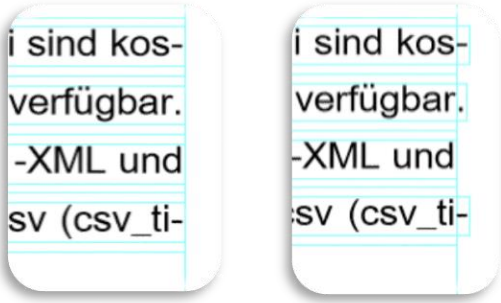


Abb. 12: AHF Renderergebnis horizontale Linie

Abb. 20: XSL-FO Listenaufbau (Quelle: Montero/Krüger, S.106)

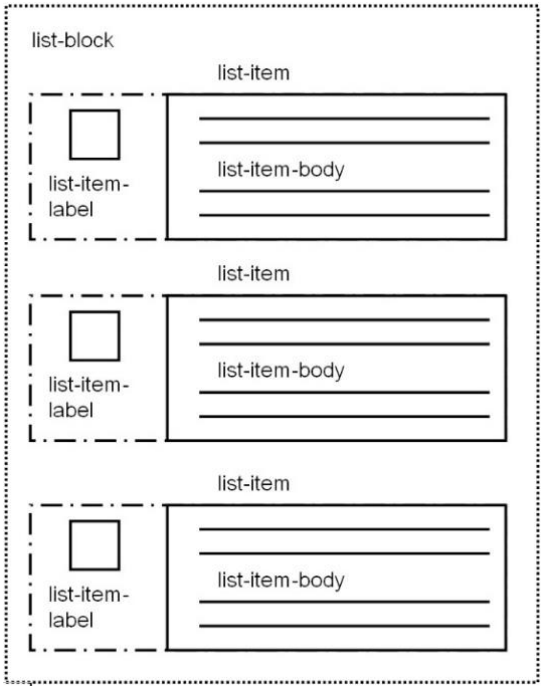


Abb. 22: XSL-FO Tabellenkonzept (Quelle: Montero/Krüger, S.116)

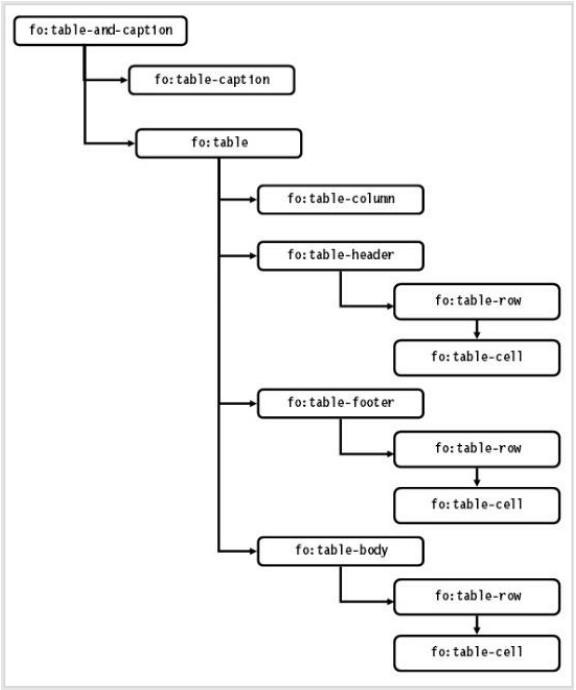


Abb. 23: HTML Tabellenkonzept (Quelle: http://wiki.selfhtml.org/images/0/09/Tabelle-1.svg)

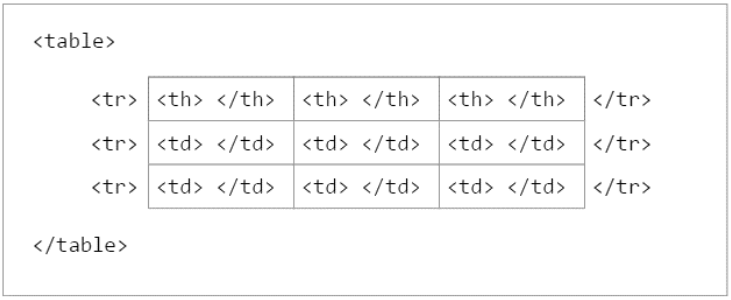


Abb.25/26: AHF Tabelle Header und Footer, (l.) ohne Wdh., (r.) mit Wdh.

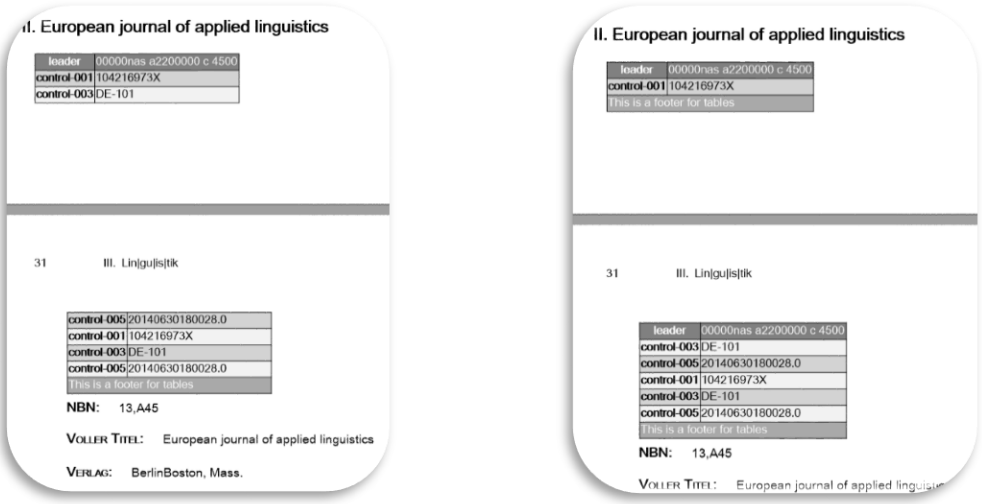


Abb. 27: AHF Tabelle Titel, (o.) ohne vertical-align, (u.) mit vertical-align:top

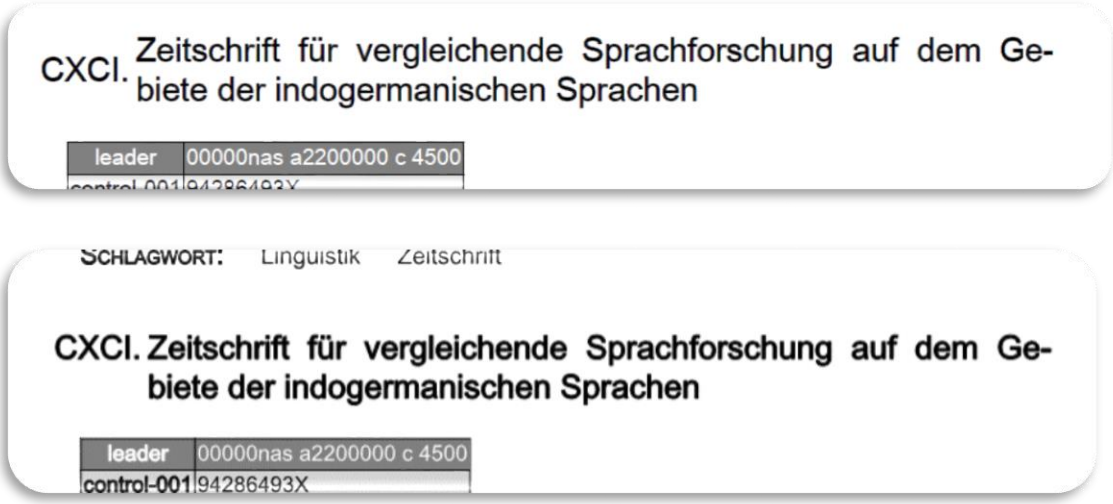


Abb. 28: o: rounded border, u: wegen float oder display:block;

Abb. 29: AHF Fehler abgerundeter Rahmen

Abb. 30/31: AHF (l.) float:bottom, (r.) float:top

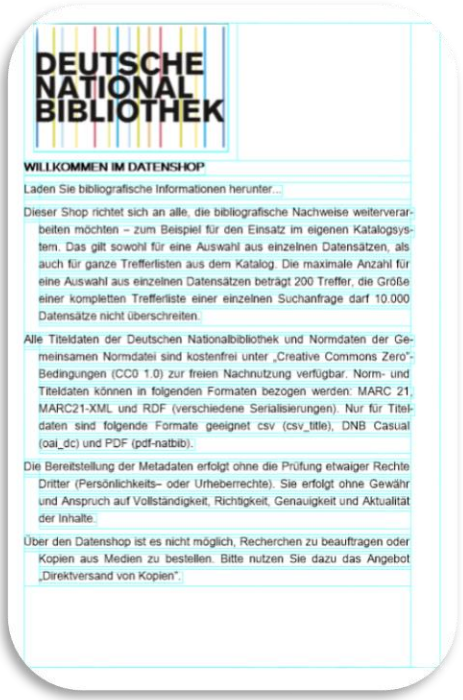


Abb. 32: AHF Textumfluss

Abb. 33: AHF Hintergrundbild Seite (Wasserzeichen)

Abb. 34: AHF Hintergrundbild Abschnitt

Abb. 36: AHF Sidenote

Abb. 37: AHF Sidenote Rahmen ohne Zeichen

Abb. 38: AHF Sidenote trifft Überschrift

Abb. 39: AHF Kolumnentitel string()-Variante

Abb. 40: AHF Kolumnentitel element()-Variante

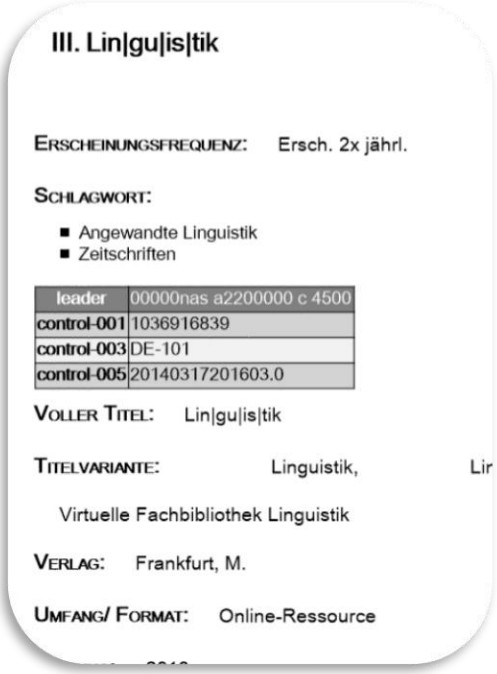


Abb. 41/42: AHF writing-mode:vertical-lr, (l.) text-orientation:upright, (r.) ohne text-orientation

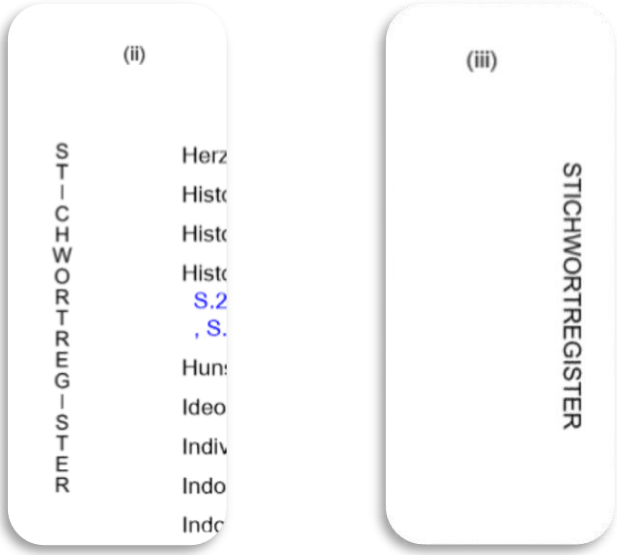


Abb. 43: AHF geteilte Paginierung, römischer Zähler

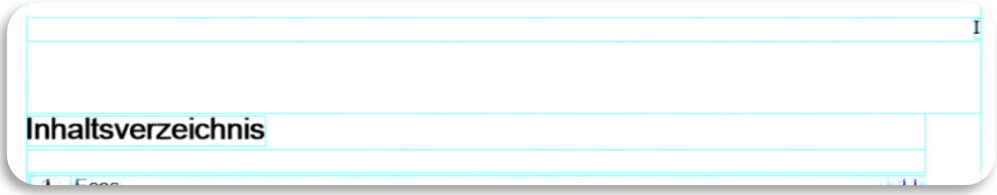


Abb. 44: AHF Paginierung Startzahl

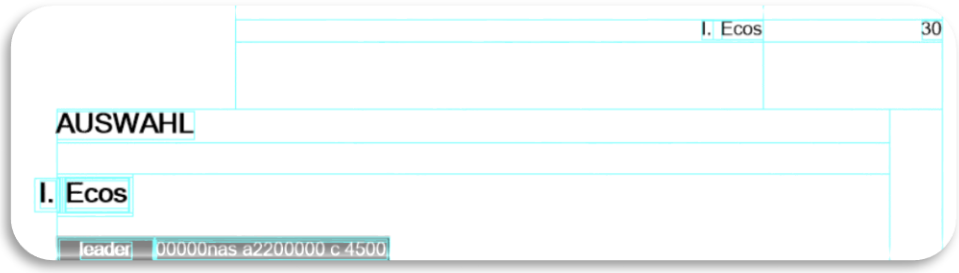


Abb. 45/46: Ansicht Adobe Acrobat Reader, …-level:2, (l.) …-state:closed, (r.) …-state:open

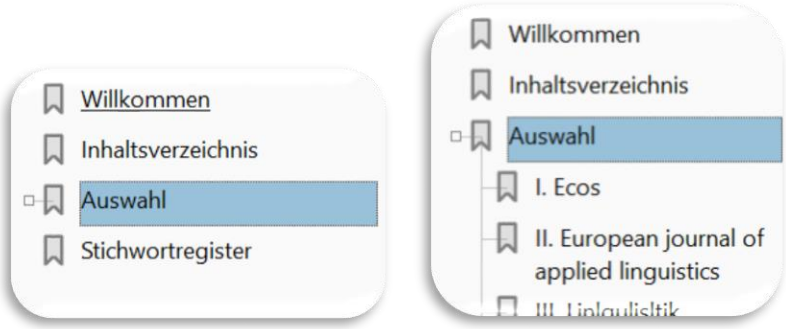


Abb. 47: AHF Anschnitt und Marker