Hochschule der Medien Stuttgart
Bachelorarbeit im Studiengang Mediapublishing

Markup-Mix beim Single Source Publishing

Abbildung des Lightweight-Ansatzes von DITA auf den Branchenstandard ParsX

Vorgelegt von Teona Burnadze
an der Hochschule der Medien Stuttgart am 29. September 2021
zur Erlangung des akademischen Grades eines Bachelor of Arts

Erstprüfer: Prof. Dr.-Ing. Marko Hedler
Zweitprüfer: Prof. Dr. Rolf Jäger

Matrikelnummer: 34283
tb121@hdm-stuttgart.de

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit dem Einsatz verschiedener Markupsprachen beim Single Source Publishing. Eine Mischung von Markupsprachen wird bereits vom XML Standard DITA verwendet. Die von DITA abgeleitete Version Lightweight DITA erlaubt einen Mix von drei Auszeichnungssprachen. Es können wechselweise XML, HTML und Markdown eingesetzt werden. In der Arbeit wird analysiert, unter welchen Voraussetzungen LwDITA einen Mix von Markupsprachen erlaubt. Hierbei werden die technischen Merkmale herausgearbeitet und im Anschluss auf andere Publikationsstandards angewendet. Die Validierung der erarbeiteten Merkmale wird auf dem Brachenstandard ParsX überprüft.

Es zeigt sich, dass alle aus dem LwDITA definierte Merkmale erfolgreich auf den ParsX-Standard übertragen wurden und die Kombination verschiedener Auszeichnungssprachen auch beim ParsX-Standard einwandfrei funktioniert.

Abstract

This thesis researches use of different markup languages in single source publishing. A mixed markup is already used by the XML standard DITA. The Lightweight DITA version, derived from DITA, allows a mix of three markup languages. XML, HTML and Markdown can be used alternately. This thesis analyzes the conditions under which LwDITA allows a mix of markup languages. The technical characteristics are worked out and applied to other publication standards. The validation of the characteristics is checked against the industry standard ParsX.

The results show that all the characteristics defined by the LwDITA have been successfully transferred to the ParsX standard and the combination of different marking languages also works perfectly with the ParsX standard.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CSS	Cascading Style Sheets
DCMI	Dublin Core Metadata Initiative
DITA	Darwin Information Typing Architecture
DTD	Document Type Definition
GFM	GitHub Flavored Markdown
HTML	Hypertext Markup Language
IMAP	Das Information Mapping®
LwDITA	Lightweight DITA
SGML	Standard Generalized Markup Language
URI	Uniform Resource Identifier
UTF-8	Unicode Transformation Format (8 Bit)
W3C	World Wide Web Consortium
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XSD	XML-Schemata
XSLT	Extensible Stylesheet Language Transformation

1 Einleitung

Im ersten Kapitel erfolgt eine Einführung in das Thema der vorliegenden Arbeit, und es wird die aktuelle Problemstellung aufgezeigt. Des Weiteren werden die Ziele und die Forschungsfrage definiert. Zuletzt wird ein Überblick über den Aufbau der Arbeit gegeben.

1.1 Problemstellung

Single Source Publishing ermöglicht es, einmal erstellte Inhalte in unterschiedliche mediale Ausprägungen zu publizieren. Diese Möglichkeit, wird durch eine medienneutrale Inhaltslagerung mit XML (Extensible Markup Language) besonders gut unterstützt.[1]

Das editorische Arbeiten mit XML ist aber im Verhältnis zu anderen Auszeichnungssprachen aufgrund der syntaktischen Gegebenheiten nicht besonders intuitiv.[2] Zudem führt die Erweiterbarkeit von XML in vielen Umgebungen zu unnötiger Komplexität, was es Autoren und Informationsanbietenden erschwert, solche Dokumente zu bearbeiten. Die vereinfachte Auszeichnungssprache Markdown vertritt einen anderen Ansatz: Markdown verwendet selbsterklärende Auszeichnungselemente, um Text zu formatieren. Dies bedeutet, dass ein Text auch von technisch nicht versierten Anwendern einfacher bearbeitet werden kann. Markdown eignet sich besonders für Autoren, für die die Arbeit mit XML eine Herausforderung darstellt, und so können die Texte aufgrund der klaren Struktur von Markdown schnell und unkompliziert bearbeitet werden.

Technisch gesehen bietet Markdown einige gleiche Funktionen und Vorteile wie XML, nämlich die Trennung zwischen Inhaltsstruktur und der visuellen Darstellung, sowie die Möglichkeit der Konvertierung in mehrere Ausgabeformate. Aufgrund der Einfachheit verfügt Markdown allerdings nicht über genügend Möglichkeiten, um allen Anforderungen des Single Source Publishing zu gerecht zu werden. Beispielsweise enthält Markdown keine Optionen zur Verwaltung von längeren Dokumenten, zur Einbindung von mehreren Dateien und Widerverwendung von Inhalten.[3] Das Fehlen einer allgemein anerkannten Standardisierung erschwert auch die Validierung und Prüfung von Dokumenten.

Es zeigt sich also, dass keine der vorgestellten Formate sich universell für alle editorischen Bedürfnisse gleichermaßen eignet. Eine Möglichkeit könnte es daher sein, für Publikationsprojekte mehrere Markupsprachen in einem Mix miteinander zu kombinieren. Diese Kombination wird bereits in der Darwin Information Typing Architecture (DITA) verwendet. Aus DITA wurde im

[1] vgl. Ott 2014, S.45
[2] vgl. Kornelsen 2004, S.31
[3] vgl. Seen 2016

Jahr 2017 die vereinfachte Version Lightweight DITA (LwDITA) abgeleitet, die neben einem kleineren Element- und Attributsatz für XML einen Mix an Markupsprachen erlaubt. XML, HTML5 und Markdown können hierbei wechselweise eingesetzt werden.[4]

1.2 Ziele und Forschungsfragen

In dieser Arbeit wird der Standard LwDITA daraufhin analysiert, unter welchen Voraussetzungen die Mischung von Markupsprachen auf andere Publikationsstandards übertragen werden kann. Die Forschungsfragen dazu lauten also:

- Inwieweit kann der LwDITA-Ansatz auf andere Publikationsstandards aus der Verlagsbranche übertragen werden?
- Welche Voraussetzungen und Bedingungen müssen vorliegen, damit der LwDITA-Ansatz bei anderen XML-Standards einsetzbar ist?

Zur Beantwortung der Forschungsfrage werden zunächst die technischen Merkmale, die beim LwDITA-Ansatz den MarkupMix ermöglichen, analysiert und herausgearbeitet. Diese Merkmale werden dann generalisiert und priorisiert. Im Anschluss daran wird die Übertragbarkeit dieser Merkmale auf andere XML-Publikationsstandards geprüft. Dies erfolgt durch eine beispielhafte Anpassung des Branchenstandards ParsX.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in zwei Teile: Zuerst wird mittels ausgewählter Literatur die erforderliche Wissens- sowie Verständnisbasis geschaffen. Darauf aufbauend findet eine empirische Untersuchung in Form einer prototypischen Umsetzung statt.

Im Detail gestaltet sich der Aufbau der Arbeit wie folgt: Zu Beginn wird Grundlegendes zum Single Source Publishing betrachtet. Insbesondere wird hier auf den XML-Standard DITA eingegangen. In den folgenden Kapiteln werden die für den Markupmix verwendeten Technologien – XML, HTML und Markdown – genauer beschrieben. Es wird außerdem ein Blick auf den XML-Standard Lightweight DITA und auf ParsX geworfen. Für die Beantwortung der Forschungsfrage arbeitet das darauffolgende Kapitel alle Merkmale heraus, die das Single Source Publishing mit mehreren Markupsprachen ermöglicht. Im praktischen Teil werden die technischen Merkmale auf den Branchenstandard ParsX angewendet; abschließend wird ein Fazit gezogen.

[4] vgl. Evia 2019

2 Single Source Publishing

Das Single Source Publishing ist eine Content-Marketing-Strategie und ermöglicht die Erstellung von Informationen in unterschiedlichen Formaten für unterschiedliche Ausgabekanäle aus einer medienneutralen Quelle.[5]

In diesem Kapitel wird auf wichtige Aspekte des Single Source Publishing eingegangen und aufgezeigt, welche Strukturierungsmöglichkeiten es für die Inhalte gibt.

2.1 Grundlagen

Die Bereitstellung von Informationen wird immer komplexer. Der jeweilige Inhalt muss in vielen verschiedenen Formaten und für unterschiedliche Zielgruppen verfügbar sein. Wenn die Informationen redundant gehalten werden, besteht das Risiko, dass inkonsistente Informationen veröffentlicht werden.[6] „Eine Kernidee zur Lösung wird heute als ‚Single Sourcing' bezeichnet."[7] Sein zentraler Aspekt ist die möglichst flexible und weitreichende Verwendung der erstellten Inhalte,[8] was eine Reduzierung von Arbeitsaufwand und Produktionskosten garantiert.

Eine wichtige Basis für das Single Source Publishing ist die Trennung von Inhalt, Struktur und Layout.[9] Die Inhalte sollen in einem Datenformat vorliegen, in dem die Struktur der Texte und weitere Informationen dargestellt werden können.[10] Die Speicherung der strukturierten Inhalte wird meistens mit XML realisiert (siehe Kap. 4.2). Die Grammatik für die XML-Sprache kann mithilfe von DTDs definiert werden (siehe Kap. 4.3). So können aus dem zentral gespeicherten Inhalt durch den Austausch der Formatangaben unterschiedliche Ausgaben generiert werden. Dies erfordert aber die Auswahl einer Technologie, welche die Datentransformation gewährleistet. Um XML-Daten zu transformieren, gibt es einen eigenen Standard: die Extensible Stylesheet Language Transformation (XSLT) (siehe Kap. 4.5). XSLT hat sich aus der frühen Standard Extensible Stylesheet Language (XSL) entwickelt, welche eine Sprachdefinition für die XML-Datenpräsentation und Datentransformationen angibt.

Mithilfe von XSLT lässt sich die Struktur eines XML-Dokuments beliebig umordnen, und die Inhalte können in Web, eBook und Apps sowie in Print-Kanälen publiziert werden.[11]

[5] vgl. Wiemer 2020
[6] vgl. ebd.
[7] Closs 2011, S.28
[8] vgl. ebd., S.13
[9] vgl. Ott 2014, S.28
[10] vgl. ebd.
[11] vgl. Ott 2014, S.85

Das Publizieren aus einer Quelle bedeutet eine nur einmalige Bearbeitung eines Dokumentes, welches als Quelldokument dann an einem Ort gespeichert und wiederverwendet werden kann. Wie diese Inhalte dann organisiert oder gruppiert werden, dafür gibt es verschiedene Strukturierungsmöglichkeiten. Eine davon, die für das Single Source Publishing zum Einsatz kommt, ist die Topic-orientierte Strukturierung,[12] bei der die Inhalte in wiederverwendbare Module (Topics) aufgeteilt werden.

2.2 Strukturieren von Inhalten

Um ein Single-Source-Konzept in die Praxis umzusetzen, ist eine sinnvolle Informationsstruktur notwendig. Zum Einsatz kommen folgende Strukturierungsmethoden:

2.2.1 Topic-orientierte Strukturierung

Wie oben erwähnt, bietet die topic-orientierte Strukturierung eine Möglichkeit, die Inhalte auf Bausteine, sogenannte „Topics“, aufzuteilen. „Ein Topic ist ein in sich abgeschlossener, möglichst kontextunabhängiger Inhaltsblock, der einer vorgegebenen Systematik genügt und eine Kernaussage enthält.“[13]

Im Gegensatz zu Buchinhalten, bei denen die Informationen eine lineare Struktur haben, geht man bei der topic-orientierten Strukturierung davon aus, dass ein Benutzer mit dem Lesen des Inhalts bei jedem Thema beginnen kann. Ein Topic macht nur Sinn, wenn es einen Bezug zu anderen Topics hat. Daher sollten zwischen den Topics bestimmte Beziehungen bestehen, die in Form von Verweisen (Links) vorliegen.[14] Eine Sammlung von Topics, die inhaltlich zusammengehören, wird als Informationsgebilde bezeichnet, die jeweils in einer Map (eine Art Inhaltsverzeichnis) hierarchisch angeordnet werden.[15] Die Topics werden in Klassen eingeteilt, welche dazu dienen, die Inhalte anhand ihrer Struktur und ihres Verhaltens zu kategorisieren. Diese Technik bietet einen Ansatz für die Modellierung und Standardisierung von topic-basierten Informationsgebilden.[16] Die topic-orientierte Strukturierung macht den Inhalt übersichtlicher und bietet spezifische Funktionen zur Wiederverwendung von Inhalten. Das gleiche Thema kann so in einer Vielzahl von Dokumenten und Projekten verwendet werden.[17]

[12] vgl. Closs 2011, S.49
[13] vgl. ebd., S.56
[14] Ebd., S.72
[15] Ebd., S.83
[16] vgl. Closs 2011, S.125
[17] vgl. Parker 2021

2.2.2 Information Mapping® (IMAP)

Das Information Mapping® (IMAP) wurde von Robert E. Horn zur Mitte der 1960er-Jahre entwickelt und sollte eine lesergerechte Darstellung der Informationen gewährleisten.[18] Beim Information Mapping werden die Informationseinheiten in kleinere, in sich geschlossene Informationsblöcke (chunks) gegliedert. Die Informationen sollten in kompakter und verständlicher Form zur Verfügung stehen, und relevante Informationen sollten schnell auffindbar sein.[19] Die Zusammenstellung von mehreren Blöcken ergibt eine Map. Da es eine definierte Gruppe dieser Informationsblöcke und Maps gibt, ist es für den Schreiber einfacher zu wissen, welche Blöcke er zu einem Thema schreiben soll. Es erlaubt ihm auch, Aufgaben leichter zu delegieren und das Geschriebene genauer zu bearbeiten.[20]

2.2.3 Modularisierung

Die Modularisierung ist die Aufteilung der Inhalte in einzelne Bausteine (Module). Der Inhalt ist in Module gegliedert, die je nach Zielgruppe oder Produkt individuell zusammengestellt werden können (Baukastenprinzip).[21] Das Information-Mapping-Verfahren bietet eine gute Basis für die Modularisierung. Durch die Zusammenstellung von Maps und Blöcken entsteht ein modular aufgebautes Dokument.[22]

Bei der Entwicklung des modularen Inhaltsansatzes sollte auf die Modulgröße geachtet werden. Wenn das Modul kleiner ist, ist der Aufwand der Modularisierung entsprechend höher. Bei einem kleineren Modul ist die Zusammenstellung der Inhalte jedoch flexibler. Die Wiederverwendung des modularen Inhalts ist ein weiterer wichtiger Aspekt. Die ordnungsgemäße Wiederverwendung kann während des gesamten Veröffentlichungszyklus erhebliche Zeit und Mühe sparen. Darüber hinaus trägt dies zur Konsistenz und Qualität der Publikation bei. Die Modularisierung bringt mehrere Vorteile, aber nicht alle Dokumente müssen modularisiert werden. Einige Dokumente sind einzigartig, sehr stabil oder so statisch, dass es wenig Nutzen bringt, sie in mehrere Teile zu zerlegen.[23]

[18] vgl. Horn 1974, S.5
[19] vgl. Finke 2009, S.4
[20] vgl. Horn 1974, S.5
[21] vgl. Closs 2011, S.27
[22] vgl. Engels o.J.
[23] vgl. Roberts 2015

3 DITA-Standard

Die topic-orientierte Strukturierung ist eine effiziente Lösung für das Single Source Publishing. Die *Darwin Information Typing Architecture* (DITA) ist wiederum ein offener XML-Standard mit Ausrichtung auf die topic-orientierte Strukturierung. DITA eignet sich für das Verfassen, Produzieren und Bereitstellen von technischen Informationen.[24]

Im Fokus des dritten Kapitels steht der DITA-Standard mit seinen wichtigen Komponenten „Topic" und „Map". Es wird auf die Struktur beider Komponenten eingegangen und gezeigt, über welche Elemente und Attribute sie verfügen und wie sie funktionieren.

3.1 Grundlagen

Der Name „Darwin" steht für Vererbung und Anpassbarkeit. Der Terminus „Information Typing" bezieht sich auf die Typisierung und Einteilung von Inhalten in verschiedene Kategorien, und das Wort „Architecture" besagt, dass DITA nicht nur eine Datenstruktur hat, sondern auch die Regeln beinhaltet, wie die Architektur angepasst und erweitert werden kann.[25]

Der DITA-Standard besteht aus den *Document Type Definitions* (DTD) und den XML-Schema (XSD). Beide Komponenten definieren das DITA-Markup und legen die Struktur von Topic- und Map-Dokumenten fest.[26]

Die höchste Standardstruktur in DITA ist das Topic. Informationen werden aus mehreren Topics aufgebaut und in einer DITA-Map, einer Art Inhaltsverzeichnis, hierarchisch angeordnet.[27] Ein Hauptziel von DITA ist es, Inhalte wiederzuverwenden. Entweder wird ein Topic in jedem topic-ähnlichen Kontext wiederverwendet oder die Inhalte werden mit einem `@conref`-Attribut versehen, das auf jedes andere gleichwertige Element in demselben oder einem anderen Topic verweisen kann.[28]

Das DITA-Modell stellt Metadaten und Attribute, die beim Filtern und Verknüpfen von Inhalten verwendet werden können, zur Verfügung. Das DITA-Modell für Metadaten unterstützt die Standardkategorien der *Dublin Core Metadata Initiative* (DCMI). Darüber hinaus ermöglichen die DITA-Metadaten die Anwendung verschiedener Content-Management-Ansätze auf die Inhalte.[29] Die Kernelemente in DITA-DTD lehnen sich an HTML und XHTML an und verwenden bekannte Elementnamen, wie `p`, `ol`, `ul` und `dl`, in einer HTML-ähnlichen Topic-Struktur. DITA-

[24] vgl. Eberlein o.J.
[25] vgl. Closs 2015, S.6
[26] vgl. O.A.S.I.S 2010, S.6
[27] vgl. Closs 2015, S.7
[28] vgl. Day 2005, S.3
[29] vgl. ebd.

Topics können auch ähnlich wie HTML geschrieben werden, um direkt im Browser gerendert zu werden.[30]

3.2 DITA-Topic

Ein Topic ist ein zentraler Inhaltsbaustein im DITA-Standard. Die Inhalte werden als unabhängige Informationsblöcke aufgeteilt und in einer eigenen Datei gespeichert. Jedes Topic steht für sich allein. Eine Verbindung zwischen diesen einzelnen Topics wird durch eine Referenzierung ermöglicht.[31]

Alle Topics haben die gleiche Grundstruktur, unabhängig vom Thementyp:

- Titel und Informationen zum Inhalt
- Block-Elemente
- Meta-Daten
- Grafiken und Multimedia-Objekte
- Indexeinträge[32]

Die erste Zeile in einem DITA-Topic besteht aus einer XML-Deklaration. Als Nächstes wird eine DOCTYPE-Deklaration angegeben, in der die topic.dtd-Datei referenziert wird. Die topic.dtd enthält weiter eine Referenz zur topic.mod-Moduldatei, in der die Elemente für das Topic definiert sind.[33]

Das erste Element in einem Topic, in dem sich alle weiteren Elemente befinden, ist <topic>. Das Wurzelelement <topic> ist in der DOCTYPE-Deklaration definiert und muss ein @id-Attribut für die Identifizierung enthalten.[34]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "../dtd/topic.dtd">
<topic id="fluidconcept">
  <title> Windshield </title>
  <body>
    <p>Windshield washer fluid...</p>
  </body>
</topic>
```

[30] vgl. Day 2005, S.4
[31] vgl. Hentrich 2008, S.156
[32] vgl. ebd.
[33] vgl. ebd.
[34] vgl. ebd.

Der Titel wird in das Element <title> eingegeben und speziell für die Navigation oder Suche verwendet. Wenn der Titel zu lang ist, werden dafür zwei Elemente eingesetzt: <navtitle>, in dem die Navigationstitel definiert sind, und <searchtitle>, das einen Suchtitel angibt. Beide Elemente werden im <titlealts>-Element definiert.[35]

Zwischen dem <title> und dem <body> Element kann das Element <abstract> mit dem Kindelement <shortdesc> eingefügt werden. Sie dienen als Kurzbeschreibung, die den Zweck beziehungsweise das Thema des Topics repräsentieren und sind auch als Linkvorschau und zum Suchen gedacht.[36] Ist der erste Abschnitt nicht für die Vorschau gedacht, dann wird der Inhalt im <abstract>-Element platziert. In das Element <shortdesc> werden hingegen die Inhalte aufgenommen, die als eine Vorschau angezeigt werden sollen.[37]

Innerhalb eines Topics werden zahlreiche Blockelemente eingesetzt. Viele davon werden auch bei einem HTML-Dokument verwendet (siehe Kap. 5), wie zum Beispiel <p> für einen Absatz, <ol> für die geordnete Liste, <ul> für ungeordnete Liste oder <table> für eine Tabelle usw.[38] Ein weiteres häufig verwendetes Block-Element ist <section>, über welches die Inhalte innerhalb eines Topics untergliedert werden. Jedem Abschnitt wird dann ein Titel zugeordnet.[39]

Ein wichtiger Bestandteil der Topic-Grundstruktur sind Metadaten. Sie repräsentieren die Hintergrundinformationen eines Topics. Beispielsweise können Informationen zum Autor oder zum Erstellungsdatum hinzugefügt werden. Die Metadaten werden im Element <prolog> platziert.[40] <prolog> enthält zahlreiche Kindelemente, die verschiedenen Metadaten auszeichnen.

```
<prolog>
  <author type="creator">Max Fritz</author>
</prolog>
```

Im obigen Beispiel gibt das Element <author> an, wer das Topic verfasst hat. Mit dem Attribut @type können zusätzliche Informationen angegeben werden. So können auch mit dem Element <copyright> die Informationen über den Urheber des Topics erfasst werden. Die weiteren Angaben dazu werden in den Attributen @type und @year gemacht.[41]

[35] vgl. Hentrich 2008, S.158
[36] vgl. ebd.
[37] vgl. ebd.
[38] vgl. ebd., S.159
[39] vgl. ebd.
[40] vgl. ebd., S.162
[41] vgl. ebd., S.163

In ein Topic können auch Multimedia-Objekte eingebunden werden. Die Elemente sind ähnlich denen in einem HTML-Dokument. Beispielsweise werden die Bilder mit dem Element <image> in das Dokument eingebettet. Der erforderliche Pfad wird mit dem Attribut @href angegeben.[42]

```
<image href="../images/hdm.png">
   <alt>HDM-Logo</alt>
</image>
```

Im <image>-Element können mehrere Attribute definiert werden. Das Attribut @height gibt die Höhe des Bilds an, @width definiert die Breite, mit dem @align kann die Ausrichtung angegeben werden, @scale gibt die Skalierung des Bildes an und @placement legt fest, wo das Bild in einem Text platziert werden kann.[43]

In einem Topic kann auch ein Stichwortverzeichnis erzeugt werden. Das Element <indexterm> sammelt alle Einträge, die dann zu einem Stichwortverzeichnis zusammengefasst werden. Die Indexeinträge haben keine Einschränkung; fast alle Elemente können solche Einträge enthalten.[44]

3.3 Topic-Typen

Die Topics enthalten verschiedene Arten von Informationen, weswegen sie auch in verschiedene Informationstypen eingeteilt werden. DITA unterscheidet folgende Topic-Typen: Aufgabe, Konzept, Referenz und Glossareintrag. Die Klassifikation der Topics ist für ihre bessere Organisation sehr wichtig, denn sonst treten viele Topics auf einmal auf, und es wird schwierig, sie effektiv zu verwenden.[45]

3.3.1 Aufgabe (task)

Ein wichtiger Bestandteil einer technischen Dokumentation sind Handlungsanweisungen. Für die Nutzer muss folgende Frage beantwortet werden: „Wie gehe ich vor?“ Die Handlungsanweisungen sind dem Topic-Typ „Aufgaben“ (tasks) zugewiesen. Sie haben eine klare Struktur und geben eine sehr präzise Schritt-für-Schritt-Anweisung, was und in welcher Reihenfolge zu tun ist.[46]

[42] vgl. Hentrich 2008, S.165
[43] vgl. ebd.,
[44] vgl. ebd., S.166
[45] vgl. Closs 2015, S.9
[46] vgl. Hentrich 2008, S.168

Die Aufgabenstruktur besteht aus verschiedenen Abschnitten, darunter die Beschreibung des Kontexts, die Voraussetzungen, die erwarteten Ergebnisse und andere Aspekte einer Aufgabe.[47] Die Grundstruktur sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE task PUBLIC "-//OASIS//ELEMENTS DITATask//EN" "../dtd/task.dtd">
<task id="task">
  <title>Computer ausschalten</title>
  <taskbody>
    <steps>
      <step>
        <cmd>Klicken Sie auf START, um...</cmd>
      </step>
    </steps>
  </taskbody></task>
```

Am Anfang des Dokuments wird die XML- und die DOCTYPE-Deklaration angegeben. In der DOCTYPE-Deklaration wird die task.dtd-Datei referenziert. Die task.dtd enthält weiter eine Referenz zur task.mod-Moduldatei, in der die Elemente für die Aufgabe definiert sind.[48]

Das erste Element in der Aufgabenstruktur ist <task>, dem ein @id-Attribut zugewiesen wird. Jede Aufgabe enthält ein <titel>- und ein <taskbody>-Element. Die Elemente <titlealts>, <shortdesc>, <prolog> und <related-links> sind optional.[49]

Der eigentliche Inhalt der Aufgabe befindet sich im <taskbody>-Element. <taskbody> hat eine spezifische Struktur und enthält folgende Elemente: <prereq>, <context>, <steps>, <result>, <example> und <postreq>. Diese Elemente sind alle optional.[50]

Das Element <prereq> enthält alle Informationen, die der Nutzer vor dem Starten der aktuellen Aufgabe benötigt. <context> stellt die Hintergrundinformationen für die Aufgabe bereit. Mit diesen Informationen erfahren die Nutzer, welchen Zweck eine Aufgabe hat. Im Element <steps> wird der Hauptinhalt der Aufgabe angegeben. Eine Aufgabe besteht aus mehreren vom Benutzer auszuführenden Schritten. <result> beschreibt das erwartete Ergebnis der Aufgabe, <example> bietet ein Beispiel, das die Aufgabe veranschaulicht oder unterstützt, und <postreq> gibt alle Schritte an, die der Benutzer nach erfolgreichem Abschluss der aktuellen Aufgabe durchführen muss.[51]

[47] vgl. Hentrich 2008, S.168
[48] vgl. ebd., S.169
[49] vgl. ebd.
[50] vgl. ebd.
[51] vgl. ebd., S.170

3.3.2 Konzept (concept)

Das Topic-Typ-Konzept beantwortet die Frage „Was ist?“ Das Konzept enthält die Hintergrundinformationen, die den Lesern helfen, die wesentlichen Informationen über ein Produkt zu verstehen. Oft beinhalten die Konzepte Informationen, die das Produkt oder einen Prozess abstrakt beschreiben; diese Konzeptinformationen helfen Benutzern jedoch, ihr bereits vorhandenes Wissen über Aufgaben zu erweitern und andere wichtige Informationen zu einem Produkt oder System aufzubauen.[52]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA Concept//EN" "../dtd/concept.dtd">
<concept id=" concept">
  <title>Informationen zum Ausschalten eines Computers. </title>
  <conbody>
    <p>Ein Computer sollte immer ordnungsgemäß beendet werden. </p>
  </conbody>
</concept>
```

Wie bei dem Aufgabe-Dokument enthält das Konzept am Anfang die XML- und die DOCTYPE-Deklaration mit dem Verweis auf die concept.dtd.[53]

Das Wurzelelement in einem Konzept ist <concept>. Jedes <concept>-Element enthält ein <title> und ein <conbody>-Element. Das <conbody>-Element ist das Hauptelement für ein Konzept und hat keine besondere Struktur. Hier können mehrere Blockelemente eingefügt werden, wie zum Beispiel die Absätze, Listen und andere Elemente.[54]

3.3.3 Referenz (reference)

Die Referenz beantwortet die Fragen „Welche Werte hat?“ oder „Welche Eigenschaften hat?“ Das Referenz-Topic beschreibt die regulären Bestandteile eines Themas oder Produkts, zum Beispiel Befehle in einer Programmiersprache. Referenzthemen können Listen, Tabellen und andere Daten enthalten, in denen die Werte und die dazugehörige Beschreibung aufgeführt sind.[55]

Referenzen können folgende Inhalte enthalten: Zutaten für Rezepte, bibliografische Listen, Kataloge und vieles weitere. Referenzthemen bieten einen schnellen Zugriff auf konkrete Fakten.

[52] vgl. Hentrich 2008, S.173
[53] vgl. ebd
[54] vgl. ebd.
[55] vgl. ebd., S. 174

Die Informationen, die ein tieferes Verständnis erfordern, sollten in einem Konzept oder einem Aufgabenthema bereitgestellt werden.[56]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE reference PUBLIC "-//OASIS//ELEMENTS DITAReference//EN" "../dtd/reference.dtd">
<reference id="kapitel8_ref">
  <title></title>
  <refbody></refbody>
</reference>
```

Eine Referenzdokument beginnt auch mit einer XML- und der DOCTYPE-Deklaration. In der DOCTYPE-Deklaration wird die reference.dtd referenziert.[57]

Das Element <reference> steht auf der obersten Ebene und ist ein Container-Element für die anderen Elemente. Das Referenz-Topic hat die gleiche Struktur wie die anderen DITA-Topics; es enthält den Titel, eine Kurzbeschreibung und den Hauptteil.[58] Innerhalb des Hauptteils werden mehrere Abschnitte, Eigenschaftslisten oder Tabellen definiert. In dem Element <refbody> wird die Struktur des Inhalts angegeben. Alle Elemente von <refbody> sind optional und können in beliebiger Reihenfolge und Anzahl vorkommen.[59]

<section> unterteilt die Inhalte in einem Referenz-Topic und organisiert die Gruppen von Informationen innerhalb eines umfassenden Themas. Das Element <section> kann nicht geschachtelt werden. Der Titel ist optional.[60]

Das Element <refsyn> ist ein spezieller Abschnitt innerhalb eines Referenzthemas. Das Element <refsyn> enthält eine kurze, schematische Beschreibung der Schnittstelle oder der übergeordneten Struktur des Subjekts.[61]

<example> enthält Beispiele, welche das aktuelle Thema veranschaulichen. Das <example>-Element hat das gleiche Inhaltsmodell wie <section>.

Das Element <table> organisiert die Informationen in einer Zeilen- und Spaltenstruktur. <simpletable> speichert die Informationen in Zeilen und Spalten und lässt keine Untertitel zu.

In dem Element <properties> stehen die Referenzinformationen, die in einer Tabelle zusammengefasst sind. <properties> listet Eigenschaften und ihre Typen, Werte und Beschreibungen auf.[62]

---

[56] vgl. Hentrich 2008, S.174
[57] vgl. ebd.
[58] vgl. ebd
[59] vgl. ebd
[60] vgl. ebd
[61] vgl. ebd.
[62] vgl. ebd.

Mit dem Element <prophead> wird der Tabellenkopf definiert. In diesem Element wird der Typ mit dem <proptypehd>, der Wert mit dem <propvaluehd> und die Beschreibung mit dem <propdeschd>-Element definiert.[63]

3.3.4 Glossareintrag (glossentry)

In einem Glossareintrag werden die einzelnen Begriffe mit den dazugehörigen Erklärungen definiert. Die Definition der Begriffe stellt sicher, dass diese nur einmal definiert werden und die Autoren den gleichen Begriffen für dasselbe Konzept verwenden.[64] Ein Glossareintrag bietet dem Leser eine gute Möglichkeit, sich mit den unbekannten Begriffen, die oft in einem Buch vorkommen, schneller vertraut zu machen.

Die Grundstruktur des Topic-Typs „Glossareintrag“ sieht wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glossentry PUBLIC "-//OASIS//DTD DITA Glossary//EN" "../dtd/glossary.dtd">
<glossentry id="glossar">
  <glossterm>HTML</glossterm>
  <glossdef>Hypertext Markup Language</glossdef>
</glossentry>
```

Der Glossareintrag beginnt mit einer XML- und der DOCTYPE-Deklaration. In der DOCTYPE-Deklaration wird die glossary.dtd referenziert. Das <glossentry>-Element ist das Wurzel-Element des Topic-Typs „Glossar“. Jedem <glossentry>-Element ist das @id-Attribut zugewiesen.[65] Die Glossareinträge enthalten kein <title>-Element. Es werden nur die Begriffe und ihre Definitionen angegeben. Der Begriff wird im Element <glossterm> erfasst, und das Element <glossdef> enthält eine Beschreibung des Glossareintrags.[66]

3.4 DITA-Map

DITA-Maps sind die Dokumente, die Inhalte sammeln und organisieren. Sie ähneln den bekannten Inhaltsverzeichnissen, haben aber deutlich mehr Funktionen. Die Map definiert die Hierarchie, also die Reihenfolge von Topics, und kann die Beziehungen zwischen den Inhaltselementen festlegen. Es können auch Metadaten und Variablen angelegt werden, die für die bessere Aufbereitung von Inhaltsprodukten sorgen. Eine Map kann die komplexen Informationsprodukte besser planen, indem sie weitere Untermaps definiert.[67]

[63] vgl. Hentrich 2008, S.174
[64] vgl. ebd. S.175
[65] vgl. ebd.
[66] vgl. ebd.
[67] vgl. Closs 2015, S.24

Ein wichtiges Element in einer Map ist <topicref>. Diese verweist auf ein oder mehrere Topics, die weiter verschachtelt werden können. Auf diese Weise können die Inhalte in der gewünschten Reihenfolge ausgeführt werden.[68]

```
<map title="DITA_Map" id="map_1">
  <topicref href="bookmap.xml">
      <topicref href="meta-daten.xml" />
  </topicref>
</map>
```

Topics werden in eine Beziehung gesetzt, die anhand von Verweisen hergestellt wird. Auch werden die Inhaltsstellen oder die Dokumente selbst in eine Beziehung gebracht. Für die Darstellung solcher Beziehungen werden Beziehungstabellen <reltable> angelegt.[69] In einem <reltable>-Element werden der Tabellenkopf, die Spalten und die Zeilen definiert. Der Tabellenkopf wird in das <relheader>-Element platziert. <relheader> enthält das <relcolspec>-Element, in dem die Anzahl der Spalten festgelegt sind. Im <relcolspec> wird mit dem Attribut @type ein Topic-Typ definiert. Dieses Attribut kann folgende Werte enthalten: concept, task, reference sowie topic.[70]

Die Zeilen werden mit dem Element <relrow> definiert und die Spalten mit <relcell>. Im folgenden Beispiel enthält <relcell> das Element <topicref>, in dem das Topic installation.xml referenziert wird. Im <relheader> ist nur eine Spalte angegeben; sie wird als Topic-Typ-Konzept festgelegt.

```
<relheader><relcolspec type="concept"/></relheader>
<relrow>
  <relcell><topicref href="installation.xml"/></relcell>
</relrow>
```

In einer Map können die Meta-Daten über das <topicmeta>-Element hinzugefügt werden. Wie bereits im Abschnitt 3.2 erwähnt wurde, sind im Topic auch die Metadaten aufgeführt. Beim Verarbeiten der DITA-Dokumente werden deren Metadaten durch die Metadaten der Map überschrieben. Metadaten können Elemente wie <author>, <copyright> oder <critdates> enthalten. Außerdem werden die Schlagwörter, die <keywords>, für die Suchmaschine definiert. <permissions> legt fest, wer auf das Topic Zugriff hat, und <prodinfo> enthält die Informationen zum Produkt.[71]

[68] vgl. Hentrich 2008, S.190
[69] vgl. Closs 2015, S.24
[70] vgl. Hentrich 2008, S.174
[71] vgl. ebd., S.194

Eine Map hat mehrere Attribute, die dem Dokument weitere Informationen hinzufügen. In einer Map werden die Topics hierarchisch angeordnet. Das bedeutet, dass die Map übergeordnete und untergeordnete Topics enthält. Wenn die Kindelemente keine Attribute haben, wird das Attribut von den Eltern vererbt.[72]

Eine Map hat folgende Attribute:[73]

- @collection-type definiert die Beziehungen zwischen Eltern- und Kindelementen,
- @format gibt an, in welchem Datentyp das Topic vorliegt,
- @linking kontrolliert die Richtung der Links in den Maps,
- @locktitle legt fest, ob der Inhalt aus dem @navtitle-Attribut ignoriert oder verwendet werden soll,
- @print gibt an, ob das Thema in die Druckausgabe aufgenommen werden soll.
- @scope identifiziert, wo die Topics im Verhältnis zur Map liegen,
- @search bestimmt, ob ein Topic in Suchindizes enthalten sein soll,
- mit @toc kann angegeben werden, ob das Topic in das Inhaltsverzeichnis aufgenommen werden soll.
- type zeigt, welche Topic-Typen in das Element <topicref> referenziert werden.

Eine Hauptfunktion von DITA ist die Wiederverwendung von Inhalten. Dazu gibt es mehrere Methoden. In einer Map kann mit dem Attribut <keydef> der Schlüssel (key) definiert werden. <keydef> enthält die beiden Attribute @key und @href. Das @key-Attribut definiert den Schlüssel, und @href referenziert das Topic.[74]

```
<map>
    <title>Defining keys</title>
    <keydef keys="example" href="file.dita"/>
</map>
```

Es ist auch möglich, die Inhalte mit dem Attribut @conref zu referenzieren und diesen Inhalt an gewünschter Stelle zu platzieren. Damit das funktioniert, muss auch das @id-Attribut angegeben werden:[75]

Wenn sich das Element in der gleichen Map befindet, wird das @conref-Attribut folgendermaßen definiert: conref="#element_id". Im anderen Fall muss die entsprechende Datei angegeben werden. conref="andere_map_datei.xml#elementid".

[72] vgl. Hentrich 2008, S.195
[73] vgl. ebd., S.196
[74] vgl. ebd., S.93
[75] vgl. ebd., S.202

4 XML – Extensible Markup Language

XML ist ein offener und flexibler Standard, erstellt vom World Wide Web Consortium (W3C) am Anfang des Jahres 1998 zum Speichern, Veröffentlichen und Austauschen jeder Art von Informationen.[76]

Der folgende Abschnitt beschäftigt sich mit den Grundlagen von XML und den Regeln für den syntaktischen Aufbau. Der Abschnitt 4.3 führt zu DTD, einem ursprünglichen Validierungsmechanismus für XML. Im Abschnitt 4.4 wird das XML-Schema vorgestellt; es ist die moderne Art, das Format eines XML-Dokuments zu beschreiben. Anschließend wird im Kapitel 4.5 XSLT, eine Methode zur Umwandlung von XML von einem Format in ein anderes, näher betrachtet.

4.1 Grundlagen

XML ist eine textbasierte Auszeichnungssprache, die für den Datenaustausch im Web entwickelt wurde.[77] Die Daten werden strukturiert, gespeichert und vereinfacht zwischen Datenbanken, Anwendungen und Organisationen übergeben. Die XML-Struktur lässt sich selbst definieren, aus diesem Grund wird diese Sprache als erweiterbar bezeichnet, was ihr Hauptvorteil ist.[78]

Im Gegensatz zu XML definieren andere Markupsprachen eine feste Anzahl von Tags. XML hingegen ist eine Metaauszeichnungssprache, bei der es sich um eine Sprache handelt, in der die Tags frei definiert werden.[79] Diese Tags müssen nach bestimmten allgemeinen Prinzipien organisiert werden, sind aber in ihrer Bedeutung flexibel. Um sicherzustellen, dass jedes Tag richtig angegeben ist, werden eine Document Type Definition (DTD) oder ein XML-Schema Definition (XSD) benötigt. DTD ist die Standard-Schemasprache für XML, die verwendet werden kann, um die Syntax und Struktur von XML zu definieren. Das XML-Schema ist eine sehr umfassende Schemasprache.[80] Es definiert auch die Strukturen von XML-Dokumenten und ist eine Empfehlung des W3C. Diese beiden Formate definieren genau, was in einem XML-Dokument möglich ist, welche Tags akzeptiert werden und wohin sie dürfen. Aus diesem Grund existieren viele Standard-DTDs, die bestimmten Arten von Dokumenten definieren, wie zum Beispiel DocBook und DITA für technische Dokumente sowie das Open Publication Format (OPF) und das Navigation Control file for XML applications (NCX) für ePubs.[81]

[76] vgl. Ali 2014, S.12
[77] vgl. Bray 2008
[78] vgl. Mertens 2013, S.3
[79] vgl. Harlod 2004, S.4
[80] vgl. Ali 2014. S.12
[81] vgl. Borgstrom 2013

Eines der Ziele von XML ist es, eine klare Trennung zwischen Inhalt und Darstellung zu erreichen. Diese Trennung ist bei XML einfach, da es keine eingebauten festen Tags wie in HTML gibt.[82] XML gibt die Muster vor, denen die Elemente folgen müssen, ohne die Namen der Elemente zu nennen. So definiert XML beispielsweise, dass Tags mit einem (<) beginnen und mit einem (>) enden. XML definiert jedoch nicht, welche Namen zwischen dem (<…>) stehen müssen.[83]

XML hat gegenüber vielen anderen Formaten eine Reihe von Vorteilen. Das XML-Markup ist sehr ausführlich. Beispielsweise muss jedes Ende-Tag mitgeliefert werden. Auf diese Weise kann der Computer häufig auftretende Fehler, wie zum Beispiel falsche Verschachtelungen, erkennen. Ein weiterer Vorteil von XML besteht darin, dass XML plattformunabhängig ist. Das bedeutet, dass jedes XML-Dokument von jedem XML-Tool gelesen und verarbeitet werden kann.[84]

Es gibt viele Bereiche, in denen XML als die führende Technologie für den Austausch von Daten eingesetzt wird. Auch für die Verlagsbranche hat XML eine große Bedeutung; immer, wenn ein Verlag crossmedial publizieren will, kommt XML zum Einsatz.[85] Im Veröffentlichungsbereich sollte XML verwendet werden, um dokumentenorientierten Inhalten eine logische Struktur, aber auch Formatierungs- und Layoutinformationen zu geben.[86] Dabei kann die einzelne XML-Datei in eine Vielzahl von Formaten konvertiert werden – HTML für Online-Präsentationen, PDF, ePubs und mehr.[87]

4.2 XML-Syntax

Die XML-Syntax wird durch die vom W3C veröffentlichte XML 1.0-Spezifikation definiert. Diese Spezifikation bietet eine Liste von Syntaxregeln, die ein XML-Dokument einhalten muss, um wohlgeformt zu sein.[88]

XML wurde für die weltweite Verwendung entworfen; daher nimmt es auch Bezug auf entsprechende Standards, wie z. B. Unicode und ISO/IEC 10646, die die Kodierung aller wichtigen Buchstaben und Zeichen der Welt erlauben, sowie RFC 1766 zur Identifikation von Sprachen.[89]

Der erste Teil eines Dokuments ist der Prolog, der mit einer XML-Deklaration beginnt:

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

[82] vgl. Fawcett 2012, S.10
[83] vgl. Harlod 2004, S.4
[84] vgl. ebd.
[85] vgl. Ott 2014, S.31
[86] vgl. Fawcett 2012, S.10
[87] vgl. Ott 2014, S.31
[88] vgl. Bray 2008
[89] vgl. Esfahani o.J.

Diese Erklärung enthält die Information über die Versionsnummer, die derzeit immer entweder 1.0 oder 1.1 lauten wird. Außerdem wird die Kodierung als UTF-8 angegeben, eine Variante von Unicode.[90] UTF-8 steht für *Unicode Transformation Format* und die „8" bedeutet, dass bei der Codierung 8-Bit-Werte verwendet werden. *Unicode* ist ein internationaler Standard, der für jedes Zeichen einen numerischen Wert (Codepoint) definiert. Das Ziel ist es, allen Zeichen, die von menschlichen Sprachen verwendet werden, einen eigenen Code zu geben und inkompatible Kodierungen zu beseitigen.[91]

Wenn das Dokument eine DTD verwendet, werden im XML-Prolog die erforderlichen Angaben gemacht.

```
<!DOCTYPE ebook SYSTEM "ebook.dtd">
```

XML-Dokumente sind durch hierarchische Elemente strukturiert. Ein Element besteht aus einem Start- und Ende-Tag. Beide Tags stehen zwischen zwei spitzen Klammern (<…>); beim Ende-Tag wird zusätzlich nach einer spitzen Klammer ein Schrägstrich angegeben (</ende-tag>).[92]

Ein weiterer wichtiger Baustein von XML-Dokumenten ist das Attribut. Die Attribute sind einem Element zugeordnet und stehen innerhalb des Start-Tags. Die Attribute bestehen aus einem Namen und einem durch ein Gleichheitszeichen getrennten Wert. Der Attributwert muss in Anführungszeichen stehen. Es können entweder einfache oder doppelte Anführungszeichen verwendet werden.[93]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ebook SYSTEM "ebook.dtd">
<ebook sprache="de">
        <titel>Der Prozess</titel>
        <person rolle="autor" sex ="m">
          <name>Franz Kafka</name>
        </person> </ebook>
```

In diesem Beispiel ist <titel> ein Element, das Der Prozess als Elementinhalt hat. Die folgende Zeile ist das Element <person>, mit einem Attribut @rolle. Der Attributwert ist autor. Die Elemente können auch leer sein und keinen Elementinhalt haben; sie können innerhalb von Start-Tags beliebig verwendet werden, um den Daten mehr Bedeutung zu geben.[94]

[90] vgl. Bray 2008
[91] vgl. PSF 2021
[92] vgl. Bray 2008
[93] vgl. Fawcett 2012, S.34
[94] vgl. Dournaee 2002, S.63

Der nächste Schritt nach dem Schreiben des Prologs ist das Erstellen des Wurzelelements. Alle Dokumente müssen ein einzelnes Wurzelelement haben. Alles andere im Dokument liegt unter diesem Element, um eine Hierarchie zu bilden. Die Regel, dass es nur ein Wurzelelement geben darf, ist eines der Grundprinzipien von XML.[95]

Es gibt mehrere Möglichkeiten, Zeichen in ein Dokument einzufügen. Falls (<) dieses Zeichen im Text vorkommt, kann es durch die Entity-Referenz *&lt*; ersetzt werden. Außerdem kann dieses Zeichen mit anderen Zeichenreferenzen dargestellt werden: der numerischen *&#60* oder der hexadezimalen Zeichenreferenz *&#x3C*;.[96]

In XML gibt es fünf Entity-Referenzen:

CHARACTER	REFERENCE
&	&amp;
<	&lt;
>	&gt;
“	&quot;
‘	&apos;

Die Verweise beginnen mit einem kaufmännischen &- und enden mit einem Semikolon. Die eigentliche Referenz erscheint als mittlerer Teil und ist eine Abkürzung des Zeichens. Zum Beispiel steht lt für weniger als. Die Referenzen ' und &quot*;* sind besonders nützlich, wenn ein Attributwert beide Arten von Anführungszeichen enthalten soll.[97]

Ebenso können in XML-Dokumenten Kommentare (Comments) und Prozessoranweisungen (PI) verwendet werden. XML-Kommentare sind ähnlich denen, die in HTML-Dokumenten geschrieben werden. Sie beginnen wie in HTML mit (<! --) und enden mit dem Zeichen (-->).[98] Zum Beispiel:

```
<! -- das ist ein Kommentar. -->
```

Die *Processing Instruction* (PI) ist eine alternative Möglichkeit, um Informationen an bestimmte Anwendungen zu übergeben. PI beginnt mit (<?) und endet mit (?>). Direkt nach dem (<?) kommt ein XML-Name.

```
<?xml-stylesheet href="person.css" type="text/css"?>
```

[95] vgl. Fawcett 2012, S.34
[96] vgl. Harold 2004, S.20
[97] vgl. ebd.
[98] vgl. ebd., S.22

4.3 DTD – Document Type Definition

Dokument-Typ-Definitionen (DTD) werden verwendet, um eine gültige Sprache von XML-Elementen anzugeben. Außerdem zeigt eine DTD, wie die verschiedenen Elemente eines Dokuments angeordnet sind. Sie sagt aus, welche Elemente ein Dokument enthalten darf, was jedes Element in welcher Reihenfolge enthalten muss und welche Attribute jedes Element hat.[99]

Eine DTD deklariert die Elemente, Attribute, Entitäten und Notationen, die in einem XML-Dokument verwendet werden; sie enthält einen Satz von Regeln, welche die Struktur eines Dokuments definieren.[100]

Die Elemente, die in XML-Dokumenten verwendet werden, müssen in einer Elementdeklaration deklariert werden. Die Elementdeklaration beginnt mit „<!ELEMENT", dann folgt der Name des zu definierenden Elements. Gemäß dem Elementnamen wird der zulässige Inhalt innerhalb des Elements definiert. Ein Element kann die Kriterien „Elementkinder" oder „Text" und auch eine Kombination aus Kindern und Text enthalten, oder das Element kann leer sein.[101] In der XML-Empfehlung gibt es vier Arten von Inhaltsmodellen: Element, Mixed, Empty, Any.

Wenn Elemente untergeordnete Elemente haben, werden diese durch Kommata getrennt aufgelistet und mit einer Klammer eingeschlossen. Wenn ein Element <*ebook*> nur das untergeordnete Element <titel> enthält, sollte die Deklaration wie folgt aussehen:

```
<!ELEMENT ebook (titel)>
```

Enthalten die Elemente keine untergeordneten Elemente, dann wird in der Klammer #PCDATA eingegeben. #PCDATA steht dabei für „parsed character data"

```
<!ELEMENT-titel (#PCDATA)>
```

Die DTD erlaubt es zu deklarieren, dass das Kindelement ein- oder mehrmals vorkommen darf. Das optionale Zeichen nach einem Namen oder einer Liste bestimmt, ob das Element oder die Inhaltspartikel in der Liste einmal oder mehrmals (+), keinmal, einmal oder mehrmals (*) oder keinmal oder einmal (?) vorkommen können.[102]

Beim Mixed-Content-Modell kann der Text allein stehen oder zwischen Elemente eingefügt werden.

```
<!ELEMENT-absatz (#PCDATA | verweis)>
```

[99] vgl. Harold 2003, S.32
[100] vgl. Buck 2000
[101] vgl. Fawcett 2012, S.85
[102] vgl. Buck 2000

Ein Absatz <absatz> kann entweder Text (#PCDATA) oder ein weiteres Element, <verweis>, enthalten. Dafür wird ein senkrechter Strich eingefügt, welcher eine „oder-Gruppe“ bildet.[103]

Einige Elemente können keine Inhalte enthalten. Beispielsweise kann in einem HTML-Dokument ein
-Element verwendet werden, das einen Zeilenumbruch anzeigt. In einer DTD kann ein ähnliches Element definiert werden.[104] Um ein Element mit einem leeren Inhaltsmodell zu definieren, wird einfach das Wort „EMPTY“ nach einem Elementnamen in der Deklaration eingefügt:

```
<!ELEMENT br EMPTY>
```

Ein Element kann auch einen beliebigen Inhalt haben. Wird das Schlüsselwort „ANY“ angegeben, ist als Elementinhalt jede Angabe erlaubt, auch das Element selbst,[105] beispielsweise:

```
<!ELEMENT container ANY>
```

Hier zeigt das Schlüsselwort „ANY“ an, dass alle innerhalb der DTD deklarierten Elemente innerhalb des Inhalts des <container>-Elements verwendet werden können.

Attribute werden in einer Attributlisten-Deklaration definiert. Sie beschreiben zusätzliche Eigenschaften von Elementen. Attributlisten-Deklarationen geben den Namen, den Datentyp und den Standardwert jedes Attributs an, das einem bestimmten Elementtyp zugeordnet ist.[106] Im folgenden Beispiel gibt die ATTLIST-Deklaration für das <person>-Element das Attribut *@sex* an.

```
<!ATTLIST person sex (m|w|i) #IMPLIED>
```

Eine ATTLIST-Deklaration besteht aus drei grundlegenden Teilen: dem ATTLIST Schlüsselwort, dem Namen des zugehörigen Elements und der Liste der deklarierten Attribute.[107]

Im Beispiel folgt auf das Schlüsselwort „ATTLIST“ der Name des zugeordneten Elements, <person>. Jedes Attribut wird mit drei Informationen aufgelistet: Attributname, Attributtyp und Attributwertdeklaration. Die Namen innerhalb der Attributliste müssen nur einmal vorkommen. Um einen Attributnamen zu deklarieren, muss einfach der Name so angegeben werden, wie er im XML-Dokument angezeigt wird.[108]

[103] vgl. Ott 2014, S.42
[104] vgl. Fawcett 2012, S.97
[105] vgl. Becher 2009, S.48
[106] vgl. ebd., S.49
[107] vgl. Fawcett 2012, S.103
[108] vgl. ebd.

Beim Deklarieren von Attributen wird festgelegt, welche Werte als Attributwerte erlaubt sind. Innerhalb der Elementdeklarationen kann beispielsweise angegeben werden, dass ein Element einen Text enthält. In einer DTD können zehn verschiedene Attributtypen definiert werden:

- CDATA gibt an, dass der Attributwert aus Zeichendaten besteht.
- NMTOKEN ist eine Zeichenfolge, die aus einem oder mehreren zulässigen XML-Namenstoken besteht.
- NMTOKENS ist eine durch Leerzeichen getrennte Liste von Namenstoken.
- Ein Attribut vom Typ ID muss einen XML-Namen enthalten, der im XML-Dokument einmalig ist.
- IDREF verweist auf ein Attribut vom Typ ID eines Elements im Dokument.
- IDREFS bezeichnet eine durch Leerzeichen getrennte Liste von XML-Namen, die irgendwo im Dokument als ID-Attributwerte verwendet werden.
- ENTITÄT ist der Name einer nicht geparsten Entität, die in einer ENTITY-Deklaration in der DTD deklariert wurde.
- ENTITIES bezeichnet eine durch Leerzeichen getrennte Liste von nicht geparsten Entitäten, die in der DTD deklariert sind.
- NOTATION ist der Name einer Notation, die in einer NOTATION-Deklaration in der DTD deklariert ist.
- ENUMERATION (Aufzählung) ist eine Liste aller zulässigen Werte für das Attribut, die durch senkrechte Striche getrennt werden. [109]

Entities sind definierte Kürzel und können mehrere Funktionen haben. Sie werden im gesamten XML-Dokument verwendet, um auf Abschnitte von Ersatztext, andere XML-Auszeichnungen und externe Dateien zu verweisen.[110]

Entities werden in zwei Arten eingeteilt: allgemeine Entities, die innerhalb des Dokumentinhalts verwendet werden und Kürzel für wiederkehrende Textabschnitte definieren,[111] und Parameter-Entities, die Kürzel für DTD-Bausteine enthalten.

Die allgemeinen Entitäten können innerhalb der DTD auf zwei Arten deklariert werden: Der Wert der Entität wird entweder direkt in der Deklaration angegeben oder er kann auf eine externe Datei verweisen.

```
<!ENTITY source-text “Beginning XML”>
```

[109] vgl. Harlod 2004, S.32
[110] vgl. Fawcett 2012, S.109
[111] vgl. Becher 2009, S.58

Auf das Schlüsselwort „ENTITY“ folgt der Name der Entität, in diesem Fall source-text. Wenn auf die Entität an anderer Stelle im XML-Dokument verwiesen wird, wird der Name folgendermaßen angegeben:

```
& source-text;
```

Bei externen Entities liegt der Ersetzungstext außerhalb der DTD. In diesem Fall ist die Entität mit einer externen Entity-Deklaration deklariert:[112]

```
<!ENTITY datenquelle SYSTEM "news.txt" >
```

4.4 XML-Schema

Aufgrund der mangelnden Funktionen von DTD beauftragte das W3C eine Arbeitsgruppe zur Entwicklung einer neuen Sprache für eine Strukturdefinition.[113] Das XML-Schema (XSD) ist ein Rahmendokument, welches die Regeln für XML-Dokumente definiert. Eine XSD legt die Struktur und die Semantik von Elementen und Attributen fest und kann zur Validierung des Inhalts des XML-Dokuments verwendet werden.[114]

Das Wurzelelement im XML-Schema ist das Element schema und enthält als Attribut die Namensraumdeklaration. Ein XML-Schema hat folgende Grundstruktur:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- XML-Schema Definitionen und Deklarationen-->
</xs:schema>
```

Wie das Beispiel zeigt, verwendet das XML-Schema ein Namensraumpräfix xs und muss im Wurzelelement http://www.w3.org/2001/XMLSchema deklariert werden; damit wird gekennzeichnet, dass es sich um eine XML-Schema-Spezifikation handelt.[115]

Jedes Element wird durch das XML-Schema-Element element definiert. Um dieses element weiter zu spezifizieren, werden verschiedene Attribute verwendet. Das folgende Beispiel zeigt eine einfache Form der Definition eines Elementes. Hier wird deklariert, dass das Element name als Datentyp string hat:

```
<xs:element name="name" type="xs:string"/>
```

Das Attribut @name gibt den Namen eines Elementes an, und @type liefert die Beschreibung, welche Datentypen im Element enthalten sein können. Dabei gibt es weitere Eigenschaften, die

[112] vgl. Fawcett 2012, S.110
[113] vgl. Becher 2009, S.58
[114] vgl. IBM o.J.
[115] vgl. Becher 2009, S.62

dem Element zugeordnet werden können. Um zu bestimmen, wie oft ein Element vorkommen kann, werden die Attribute minOccurs und maxOccurs verwendet. Der Standardwert für beide Attribute ist 1. Wenn die Attribute minOccurs und maxOccurs nicht vorhanden sind, muss das Element nur einmal vorkommen.[116]

Das XML-Schema besteht aus einfachen (atomaren) und komplexen Datentypen. Einfache Elemente sind solche, die keine weiteren Elemente und Attribute enthalten. Zur Deklaration werden nur zwei Attribute verwendet: @name und @type. Als Datentypen werden entweder vordefinierte oder benutzerdefinierte Datentypen angegeben. Für das XML-Schema gibt es 44 vordefinierte Datentypen anhand der Spezifikation des W3C. [117] Wenn diese Datentypen verwendet werden, muss das definierte Namensraum-Präfix vorangestellt werden.[118]

Der komplexe Datentyp hingegen kann weitere Elemente und Attribute enthalten; er wird mit complexType notiert und mittels der Elemente sequence, all oder choice definiert. Sie geben Auskunft über die enthaltenen Elemente, ob sie in fester oder beliebiger Reihenfolge vorkommen oder ob sie eine Auswahl bilden.[119]

Aus den komplexen Datentypen kann ein neuer Datentyp abgeleitet werden. Die Ableitung kann auf zwei Arten erfolgen: Ableitung durch Einschränkung und Ableitung durch Erweiterung. Bei der Ableitung durch Erweiterung werden zum bisherigen Typ neue Elemente oder Attribute hinzugefügt. Aus der Einschränkung der Datentypen wird ebenfalls eine neue Definition abgeleitet. Die Einschränkung komplexer Typen erlaubt es, sowohl den Attributen als auch den Textknoten, die in Elementen einfachen Inhalts zugelassen sind, neue Bedingungen hinzuzufügen.[120]

Die Attribute im XML-Schema werden innerhalb eines komplexen Typs an letzter Position deklariert. Die Attribute werden mit dem Element attribute definiert. Ähnlich wie in einem element enthält das Element attribut die Attribute @name, @type, @default und @fixed.[121]

Die Syntax einer Attributdeklaration sieht wie folgt aus:

```
<xs:attribute name="medien" type="buch"/>
```

Um herauszufinden, ob ein Attribut vorkommen muss, wird das Attribut @use verwendet. Wenn dieses Attribut erforderlich ist, wird als Wert required angegeben. Muss das Attribut

[116] vgl. Fallside 2004
[117] vgl. Becher 2009, S.90
[118] vgl. ebd., S.91
[119] vgl. ebd.
[120] vgl. Van der Vlist 2002, S.90
[121] vgl. ebd.

nicht vorkommen, dann ist der Wert option. Der Wert prohibited wird angegeben, wenn das Attribut nicht vorkommen darf.[122]

Im XML-Schema kann auch ein leeres Element deklariert werden. In diesem Fall hat es keinen Inhalt, sondern nur Attribute.[123]

```
<xs:element name="datentraeger">
  <xs:complexType>
    <xs:attribute name="speicher" type="xs:integer" use="required"/>
  </xs:complexType>
</xs:element>
```

XSD verwendet keine Entities und Parameter-Entities mehr. Dafür sind Element- und Attributgruppen vorhanden, die diese Funktionen ausführen. Das Element <xs:group> definiert eine Gruppe von Elementen, die als globale Elemente angelegt werden können, und <xs:attributeGroup> definiert Gruppen von Attributen, die gleichfalls global verwendet werden.[124]

Wenn eine XSD-Datei in einer anderen Datei referenziert wird, wird hier das Element <xs:include> verwendet. Das erleichtert die Wiederverwendbarkeit von XSD-Dokumenten.[125] Ein Beispiel dafür ist folgender Code:

```
<xs:include schemaLocation="beispiel.xsd"/>
```

Hier teilt <xs:include> dem Parser mit, dass er die Deklarationen aus der XSD-Datei beispiel.xsd an der Stelle verwenden soll, an der sich dieses Element befindet.

Eine andere Möglichkeit, die externen XML-Module in eine XSD-Datei einzubinden, bieten die Elemente <xs:redefine> und <xs:import>. Das Element <xs:redefine> hat ähnliche Funktionen wie <xs:include>, aber dabei gibt es eine Ausnahme: Bei einfachen und komplexen Typen kann eine Definition geändert werden. Die Änderungen könne auch bei Element- und Attributgruppen vorgenommen werden.[126] Das XML-Schema verwendet auf Dokumentenebene XML-Inclusions statt der Entities. Das Element <xi:include> mit dem Attribut @href bindet die XML-Dateien oder andere Texte in eine XML-Dokument ein.[127]

XSD definiert auch keine Entities für Sonderzeichen. Für diese Funktion werden entweder Character References oder es werden die Elemente mit festen Werten verwendet.

[122] vgl. Jung o.J.
[123] vgl. Fallside 2004
[124] vgl. Ott o.J.
[125] vgl. Fallside 2004
[126] vgl. Van der Vlist 2002, S.271
[127] vgl. Fallside 2004

4.5 XSL – Extensible Stylesheet Language

Die Extensible Stylesheet Language Transformation (XSL/T) ist eine Recommendation des W3C. Die Spezifikation definiert die Syntax und Semantik von XSLT 3.0, die für die Transformation von XML-Dokumenten in andere XML-Dokumente entwickelt wurde.[128] Für die Transformation von Eingabebaum in Ausgabebaum ist der XSLT-Prozessor zuständig. Dieser Ausgabebaum kann entweder wieder ein XML- oder ein HTML-Dokument sein.

Bei XSLT handelt sich stets um eine XML-Datei mit dem Wurzelelement <xsl:stylesheet>. Diese wird nach dem XML-Prolog angegeben und enthält ein @version-Attribut, das den Versionsstand von XSLT nennt. Der Grundaufbau eins XSLT-Stylesheets sieht wie folgt aus:

```
<?xml version="1.0"?>
  <xsl:stylesheet version="3.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:template match="….">
      </xsl:template>
  </xsl:stylesheet>
```

Mithilfe des Attributs @xmlns wird ein Namensraum deklariert; neben diesem Attribut wird das Namensraum-Präfix xsl angegeben. Ein Namensraum dient dazu, ein XML-Element einer Elementgruppe (einer Markupsprache) zuzuordnen.[129]

Nach dem Element <xsl:stylesheet> kann ein <xsl:output>-Element hinzugefügt werden. Dieses gibt an, in welchem Format das Zieldokument ausgegeben werden soll. Es gibt folgende Optionen: xml als Standardformat sowie html und text. Es gibt auch die weitere Option xhtml, aber nur dann, wenn die Version 2.0 verwendet wird. Wenn das Ausgabeformat XML ist, muss das Element <xsl:output> nicht angegeben werden.[130]

Die Transformation wird durch eine Reihe von Templateregeln erreicht. Das Element <xsl:template> enthält eine Verarbeitungsanweisung und steuert damit die Ausgabe der Daten.[131] Das Element <xsl:template> enthält entweder ein @match-Attribut oder ein @name-Attribut oder beides. Das @match-Attribut wird verwendet, um ein Template mit einem XML-Element zu verknüpfen. Das @match-Attribut kann auch verwendet werden, um ein Template für das gesamte XML-Dokument zu definieren. Der Wert des @match-Attributs ist ein XPath-Ausdruck. In diesem Fall definiert match="/" das gesamte Dokument.[132]

[128] vgl. Kay 2017
[129] vgl. Bongers 2008, S.35
[130] vgl. Ott 2014, S.92
[131] vgl. ebd., S.93
[132] vgl. Kay 2017

Es ist nicht erforderlich für alle Elemente ein eigenes Template zu erstellen. Wenn alle weiteren Elemente innerhalb des Wurzelelements ausgegeben werden sollen, wird das Element <xsl:apply-templates> verwendet. Dieses Element legt die Verarbeitungsreihenfolge fest, und mit dem Attribut @select teilt es dem XSLT-Prozessor mit, welche Knoten an diesem Punkt im Ausgabebaum verarbeitet werden sollen.[133]

Für den Zugriff auf einzelne Tags wird das Element <xsl:value-of> verwendet. Ein wichtiges Attribut des value-of-Elements ist select. Mithilfe des select-Attributs wird der auszugebende Wert selektiert.[134]

XML-Dokument (Auszug):	*Stylesheet (Auszug):*	Ergebnis (Auszug):
<name> <vorname>Max</vorname> <nachname>Kurz</nachname> </name>	<xsl:template match="name"> <xsl:value-of select="vorname"/> <xsl:text> </xsl:text> <xsl:value-of select="nachname"/> </xsl:template>	... Max Kurz ...

In diesem Beispiel werden die Werte aus den Elementen <vorname> und <nachname> selektiert und in einem Tag ausgegeben. Das Element <xsl:text> erzwingt hier das Leerzeichen zwischen Vor- und Nachnamen.

Soll jedes Element aus einer Gruppe von Knoten ausgewählt werden, wird das Element <xsl:for-each> verwendet. Dieses Element sollte nicht mit einer Schleife in einer Programmiersprache verglichen werden, da die Einhaltung der Verarbeitungsreihenfolge nicht garantiert werden kann.[135]

```
<ul>
    <xsl:for-each select=”People/Person”>
        <li>
            <xsl:value-of select=”Name” />
        </li>
    </xsl:for-each>
</ul>
```

Wie in anderen Programmiersprachen verwendet XSLT auch verschiedene Möglichkeiten, um die bedingten Anweisungen anzugeben. Der erste Weg besteht darin, das Element <xsl:if> zu

[133] vgl. Harold 2004, S. 151
[134] vgl. Bongers 2008, S.964
[135] vgl. Fawcett 2012, S.253

verwenden. Dies ermöglicht einfache Tests durchzuführen. Mit dem Attribut @test wird ein boolescher Wert true oder false erzeugt. Wenn die Bedingung wahr ist, wird die Anweisung innerhalb des Elements ausgeführt.[136]

Neben der if-Anweisung wird meistens eine else-Anweisung verwendet. In XSLT gibt es jedoch keinen else-Zweig, um den Fall zusätzlich abzudecken.[137] Um mehrere Alternativen zu formulieren, gibt es ein Element <xsl:choose> mit den Kindelementen <xsl:when> und <xsl:otherwise>. Mit dem Element <xsl:when> werden die einzelnen Bedingungen dargestellt, und <xsl:otherwise> deckt den Fall ab, wenn keine der Bedingungen zutrifft.[138]

XSLT bietet zwei Elemente an, um die Knoten fortlaufend zu nummerieren oder zu sortieren. Das leere XSLT-Element <xsl:number> nummeriert die Knoten im Ergebnisbaum. Die Art der Nummerierung wird mit dem Attribut @format gesteuert.[139] Beispielsweise wird mit format="1", die Zählung mit arabischen Ziffern erzeugt. Wenn die Knoten sortiert werden sollen, wird das Element <xsl:sort> verwendet. Mithilfe des Attributs @select wird festgelegt, welche Knoten sortiert werden sollen. Das Attribut @order legt die Sortierreihenfolge fest. Dabei werden die beiden Werte angegeben: ascending (aufsteigend) oder descending (absteigend).[140]

In XSLT gibt es die Möglichkeit, in einer Variablen einen Wert zu speichern. Wenn ein Wert mehrmals benötigt wird, kann das Element <xsl:variable> eine Variable für diesen Wert definieren.[141] Mit dem Attribut @name wird der Name für die Variable festgelegt. Als Wert wird entweder ein längerer Text angegeben oder es kann mit dem Attribut @select ein XPath-Ausdruck angegeben werden, um somit der Variablen einen Wert aus der XML-Datei zuzuweisen.[142]

```
<xsl:variable name="inhalt" select="/text"/>
oder
<xsl:variable name="inhalt">
        Hier steht ein Text
</xsl:variable>
```

[136] vgl. Fawcett 2012, S.266
[137] vgl. ebd.
[138] vgl. Ott 2014, S.96
[139] vgl. ebd., S. 97
[140] vgl. Bongers 2008, S.912
[141] vgl. Ott 2014, S.102
[142] vgl. ebd.

5 HTML – Hypertext Markup Language

Die Hypertext Markup Language (HTML) ist eine textbasierte Auszeichnungssprache, die im Jahr 1989 von dem britischen Wissenschaftler Tim Berners-Lee entwickelt wurde. Seit 1994 wird es vom World Wide Web Consortium weiterentwickelt. HTML orientiert sich an SGML, aber es besteht aus festen Auszeichnungsbefehlen.[143]

Im Folgenden wird auf die Syntax, die Elemente und wichtige Funktionen von HTML eingegangen und gezeigt, wie die HTML-Dokumente strukturiert werden.

5.1 Hintergrund

HTML ist eine Kombination aus Hypertext und Auszeichnungssprache. Der Hypertext dient hier für die Verbindung von Texten mithilfe von Hyperlinks. Eine Auszeichnungssprache definiert das Textdokument innerhalb von vordefinierten Tags.[144] Bei der ersten Version von HTML wurden nur 18 Tags festgelegt. Es war möglich, einen Titel, Überschriften und Absätze zu definieren. Außerdem wurde mithilfe eines Links eine Verknüpfung zwischen den Dokumenten hergestellt.[145]

Nach der ersten Version wurde HTML mehrmals erweitert, aber ein erster Standard konnte erst im Jahr 1995 veröffentlicht werden. Diese Version war HTML 2.0 und beinhaltete eine wichtige Funktion: die Verwendung von Formularen.[146] In den folgenden Version HTML 3.2 konnten bereits dynamische Inhalte erzeugt werden. Dabei handelte es sich um ein kleines Programm, die von Java programmiert wurde. Außerdem war es möglich eine Tabelle zu erstellen.[147]

Im Jahr 1997 erschien die Version HTML 4.0. Eine Besonderheit dabei war die Verwendung von Cascading Style Sheets (CSS). Eine Grundidee hierbei war die Trennung von Information (Daten) und Darstellung. CSS ist eine Gestaltungssprache, die das Layout für die HTML-Dokumente festlegt und bildet zusammen mit HTML und XML den Kern des World Wide Web.[148]

Ein weiterer HTML-Standard wurde im Jahr 2000 definiert, nämlich die Extensible Hypertext Markup Language (XHTML). XHTML basiert auf der Version HTML 4.01, aber unter Beachtung

[143] vgl. Ott 2014, S.30
[144] vgl. Bühler 2017, S.2
[145] vgl. Fuchs 2019, S.19
[146] vgl. ebd., S. 21
[147] vgl. ebd.
[148] vgl. Krüger 2011, S.21

der XML-Syntax. Im Gegensatz zu HTML sollte XHTML erweiterbarer und flexibler sein, um mit anderen Datenformaten (wie XML) arbeiten zu können.[149]

Eine fünfte Fassung der Hypertext Markup Language entstand im Jahr 2014. Mit der Entwicklung des Internets nahm die Bedeutung der HTML erheblich zu. HTML 4 und XHTML hatten nicht genügend Funktionen, um die Übertragung von Daten und Inhalten zu gewährleisten, weswegen es notwendig war, eine weitere Version zu entwickeln.[150]

HTML5 verfügt einen breiteren Funktionsumfang. Dazu gehören die Unterstützung von Multimediainhalten und die semantische Auszeichnung von Inhalten. Mit der Einführung neuer Elemente können beispielsweise das Menü, die Kopfleiste oder der Fußbereich direkt ausgezeichnet werden.[151] Mit dem Element <canvas> kann jede Art von Grafiken oder Animationen im Browser dargestellt werden. Außerdem ist das Design unabhängig vom verwendeten Endgerät einfacher anzupassen.[152]

5.2 Der Grundaufbau einer HTML-Seite

Wie bei XML-Dokumenten wird am Anfang eines HTML-Dokuments eine Dokumenttypdeklaration angegeben. <!DOCTYPE> gibt an, um welchen Typ von Dokument es sich handelt.[153] HTML5 basiert nicht auf SGML, und die Deklaration ist sehr einfach anzugeben:

```
<!DOCTYPE html>
```

Das erste Element, in dem sich alle Elemente befinden, ist <html>. Das HTML-Dokument wird dann in zwei Bereiche unterteilt. Der erste ist der Kopfbereich, welcher in dem Element <head> steht. Die Inhalte der Seite stehen dann im zweiten Bereich innerhalb des <body>-Elements.[154]

In den <html>-Tag kann das Attribut @lang für die Sprachangabe eingefügt werden.[155] Diese Angabe ist für Suchmaschinen sehr hilfreich, um die sprachspezifischen Ergebnisse zurückzuliefern. Die Inhalte werden automatisiert verarbeitet, und der Browser kann die Rechtschreibfehler erkennen. Für das @lang-Attribut wird als Wert ein Sprachkürzel verwendet. Beispielsweise wenn als Kürzel de steht, weist das darauf hin, dass es sich um eine deutschsprachige Seite handelt; das Kürzel en wird für englischsprachige Seiten eingegeben.[156]

[149] vgl. Ott 2014, S.30
[150] vgl. Fuchs 2019, S.23
[151] vgl. ebd., S.24
[152] vgl. Mason 2020, S.31
[153] vgl. Ott 2014, S.53
[154] vgl. Fuchs 2019, S.40
[155] vgl. Hickson 2021
[156] vgl. Fuchs 2019, S.45

Im Kopfbereich werden weiterführende Angaben gemacht, die vom Browser verwendet werden, um die HTML-Seite zu organisieren. Hier werden die Informationen angegeben, welche Auskunft über den Seitentitel, den verwendeten Zeichensatz, den Stil und andere Metainformationen geben.[157]

```
<html lang="de">
    <head>
        <title>My fisrt page</title>
    </head>
    <body> Hallo Welt! </body>
</html>
```

Das <head>-Element beinhaltet folgende Elemente: <title>, <style>, <meta>, <link>, <script> und <base>. Der Titel der Seite wird im <title> definiert und dann in der Symbolleiste des Browsers angezeigt. Wird die Website zu den Favoriten hinzugefügt, wird auch dieser Titel verwendet.[158] Der Inhalt eines Seitentitels hat eine große Bedeutung für die Suchmaschinenoptimierung (SEO). Die richtige Titelangabe hilft den Suchmaschinen beim Auffinden der Seite, und die Besucher können einen ersten Eindruck von der Seite bekommen.[159]

Das Element <style> enthält die Stilinformationen und ermöglicht das Einbetten der CSS-Datei in die Dokumente. Zusätzlich zu diesem Element kann die CSS-Datei extern in einer separaten Datei definiert werden. Diese Datei wird mithilfe des <link>-Elements in das Dokument eingefügt,[160] und verlinkt die HTML-Dokumente mit einer externen Ressource.

```
<link rel="stylesheet" href="example.css">
```

Das <link>-Tag enthält zwei wichtige Attribute: das @href, in dem die Adresse der Links angegeben wird, und @rel, das den Typ der angegebenen Links definiert. Wenn das Attribute @href leer ist, definiert das Element keinen Link.[161]

Das <meta>-Element stellt verschiedene Arten von Metadaten zur Verfügung. Mit den Attributen @name und @content werden die Metadaten auf Dokumentebene dargestellt. Es können beispielsweise die von Suchmaschinen verwendeten Schlüsselwörter definiert oder auch der Autorenname angegeben werden.[162] Um eine HTML-Seite richtig und korrekt anzuzeigen, sollte eine entsprechende Zeichencodierung angegeben werden. Es gibt verschiedene Arten

[157] vgl. Fuchs 2019, S.40
[158] vgl. Agular 2008, S.19
[159] vgl. Bühler 2017, S.6
[160] vgl. ebd., S.47
[161] vgl. Hickson 2021
[162] vgl. Agular 2008, S.196

der Zeichenkodierung, die im Attribut @charset deklariert werden. Die Standardzeichencodierung für HTML5 ist UTF-8.[163]

Mit dem Element <base> kann eine Basis-URI oder eine Basis-URL angegeben werden. Es definiert, wie die Links im aktuellen Fenster geöffnet werden müssen. Im Dokument darf nur ein <base>-Element vorhanden sein. Das <base>-Element muss entweder ein @href-Attribut, ein @target-Attribut oder beides haben.[164]

Die Gestaltung einer HTML-Seite mit interaktiven Elementen und Animationen ist eine Aufgabe der Programmiersprache *JavaScript*. Um dieses Skript in einer HTML-Datei anzugeben, wird das Element <script> verwendet. Dieses Tag kann auch innerhalb des Hauptteils definiert werden. Im <script> erfolgt entweder der Aufruf einer externen Datei, die im Attribut @src festgelegt wird, oder es kann auch der Code direkt eingefügt und verwendet werden.[165]

5.3 Die Body-Elemente

Nachfolgend werden die wesentlichen Elemente des Body-Bereichs aufgeführt. Dieser Bereich enthält den gesamten Inhalt, der im Browser angezeigt wird.

Am Anfang der Seite wird meistens eine Überschrift angezeigt. Sie vermittelt dem Leser, was auf der Seite zu erwarten ist, und weckt sein Interesse. HTML bietet sechs verschiedene Ebenen, um eine Überschrift darzustellen. Diese Ebene wird von <h1> bis <h6> gebildet.[166] Die wichtigste Überschrift wird im <h1>-Tag definiert. Der Browser zeigt die Überschriften in größerer und fetter Schrift an. Die Überschrift <h1> wird am größten und die mit <h6> am kleinsten dargestellt. <h6> ist manchmal sogar kleiner als der normale Fließtext.[167]

Für die Strukturierung der Texte kommt noch ein wichtiges Element zum Einsatz: das <p>-Tag, das einen Textabsatz definiert. Die Texte werden als eigener Block dargestellt und mit einem kleinen Abstand voneinander abgetrennt.[168]

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

Eine weitere Möglichkeit, um im Text Zeilenumbrüche vorzunehmen, ist das
-Tag. Das
-Element ist ein leeres Element; dementsprechend ist kein End-Tag </br> erforderlich.[169]

[163] vgl. Hickson 2021
[164] vgl. ebd.
[165] vgl. Agular 2008, S.202
[166] vgl. Hickson 2021
[167] vgl. ebd.
[168] vgl. ebd.
[169] vgl. ebd.

Es werden nicht alle Texte im Browser dargestellt. Wenn im Code eigene Eingaben erforderlich sind, werden sie als Kommentare zwischen <!--…--> Tags eingegeben.[170] Sie werden überall im HTML-Dokument eingefügt und von den Entwicklern nur verwendet, um eine übersichtliche Darstellung des Codes zu gewährleisten.

Das <p>-Element enthält mehrere Elemente, die den Text hervorheben. Das <b>- oder -Tag bietet die Möglichkeit, die Textteile fett darzustellen. Das <b>-Tag wird nur für die geänderte Darstellung verwendet, während der -Tag für die semantische Hervorhebung geeignet ist. Darüber hinaus ist es möglich, eine kursive Hervorhebung im Text vorzunehmen. Dafür gibt es zwei Möglichkeiten: <i>- und -Tag.[171] Das <i>-Tag hat die gleiche Funktion wie das <b>-Tag; der Text wird diesem Fall nur optisch verändert. Das -Tag wird hingegen verwendet, um den hervorgehobenen Text zu definieren. Für die Hervorhebung bestimmter Textstellen wird das <u>-Tag eingesetzt, das einen Unterstrich bietet, also eine Linie unterhalb des Textinhalts.[172]

HTML hat zwei Kategorien von Elementen: Inline- und Block-Elemente. Block-Elemente enthalten andere Elemente; man kann sich dies als einen Container vorstellen. Die Block-Elemente nehmen die gesamte Breite der Seite beziehungsweise ihres Hauptelements ein und werden durch einen Zeilenumbruch von anderen Inhalten getrennt. Das am häufigsten verwendeten und beliebtesten Tag in HTML-Seiten ist .[173] Es ist ein Containerelement, das mehrere Elemente enthält und die Inhalte auf der Seite strukturiert, hat keine semantische Bedeutung, und die Eigenschaften werden mithilfe von CSS-Stilregeln zugewiesen.

Das -Tag ist ein Inline-Element, das innerhalb anderer HTML5-Elemente verwendet wird. wird häufig eingesetzt, um kleine Textabschnitte hervorzuheben.[174]

```
<div>
    <p>Dies ist mein <span>erster Absatz</span>.</p>
    <p>Das ist mein zweiter Absatz</span>.</p>
    <p>Dies ist mein letzter Absatz</span>.</p>
</div>
```

Ein weiteres Element ist der Link. Links werden mittels <a>-Tags dargestellt und spielen eine wichtige Rolle für HTML-Dokumente. Zum einen machen sie es möglich, die einzelnen Seiten

[170] vgl. Hickson 2021
[171] vgl. Fuchs 2019, S.61
[172] vgl. ebd., S.64
[173] vgl. Fuchs 2019, S.65
[174] vgl. ebd., S.68

miteinander zu verbinden, zum anderen lassen sich mit ihnen externe Inhalte in die Seite einbinden. Das <a>-Element funktioniert nur mit dem Attribut @href, in dem die Adressen von Links eingegeben werden.[175] Wenn in einer HTML-Seite eine andere Seite angegeben werden muss, sollte der Link folgendermaßen aussehen:

```
<a href="shop.html">Willkommen in unserem Shop</a>
```

Bilder und Multimediaelemente machen eine Website lebendig und ansprechend. HTML hat entsprechende Tags, um die Bilder und Medieninhalte auszuzeichnen. Um ein Bild in eine Seite einzubinden, verwendet HTML das <img>-Tag, in dem die Bilder referenziert werden.[176] Dieses Tag hat zwei erforderliche Attribute:

- @src – gibt den Pfad zum Bild an
- @alt – gibt einen alternativen Text für das Bild an, wenn das Bild aus verschiedenen Gründen nicht angezeigt werden kann.[177]

Abhängig von der Position des Bildes in Bezug auf das HTML-Dokument wird sein @src-Attribut unterschiedlich referenziert. Befinden sich beispielsweise das HTML-Dokument und das Bild in einem Ordner, wird hier statt der Bildadresse nur der Dateiname angegeben.

```
<img src = "image1.png" alt = "image1"/>
```

Wenn das HTML-Dokument auf das Bild, das sich im Unterordner „Image" befindet, verweist, dann wird der Unterordnername/Dateiname angezeigt.

```
<img src = "Image / image2.png" alt = "image2"/>
```

Wenn die Bilder eine Beschriftung haben, werden sie mit dem Tag <figcaption> dargestellt. Dabei wird eine kurze Beschreibung des Bildes und der Urheber oder der Angaben zur Lizenz gemacht.[178]

Für die Einbindung von Audiodateien kommt das Element <audio> zum Einsatz. Auch für Videodateien ist ein entsprechendes Element vorgesehen: das <video>-Tag. Beide Elemente enthalten das Kindelement <source>, in das der Dateipfad über das @src-Attribut eingegeben wird.[179] Für die Steuerung der Medieninhalte enthalten <audio>- und <video>-Elemente den Parameter control, der die Steuerungselemente definiert, genau wie in einem normalen Audio-/Videoplayer.

HTML bietet eine Möglichkeit, Listen zu erstellen. Eine Liste kann verwendet werden, um Menüs, Strukturen und ähnliche Objekte zu kategorisieren. Es gibt zwei Listenarten: ungeordnete

[175] vgl. Fuchs 2019, S.
[176] vgl. Hickson 2021
[177] vgl. ebd.
[178] vgl. Fuchs 2019, S.97
[179] vgl. Hickson 2021

und geordnete Listen. Ungeordnete Listen werden zwischen den -Tags platziert, sie werden nicht nummeriert, und die Anordnung von Listenelementen spielt hier keine Rolle. Um ein Listenelement zu erstellen, wird ein -Tag verwendet.[180]

Die geordnete Liste wird verwendet, wenn eine Reihenfolge wichtig ist. Das -Tag stellt diese dar. Die geordnete Listenelemente im Browser können in arabischen Ziffern (1, 2, 3 usw.), in lateinischen Buchstaben (A, B, C … oder a, b, c …) oder mit römischen Ziffern (I, II, III ...) angezeigt werden. Die visuellen Eigenschaften von Listen und deren Nummerierungsstil können im Browser über den CSS definiert werden.[181]

Um die Inhalte zu strukturieren, gibt es noch die Möglichkeit, eine Tabelle zu erstellen. HTML bietet hier die Elemente, die vonnöten sind, um eine Tabelle mit Spalten und Zeilen frei zu gestalten. Hier kommen folgende Elemente zum Einsatz: <table>, <tr>, <td>, <th>; es können aber auch die Elemente <caption>, <colgroup>, <thead>, <tfoot> und <tbody> verwendet werden.[182]

Das <table>-Tag definiert eine ganze Tabelle. Innerhalb von <table> werden die einzelnen Zeilen mit dem Tag <tr> definiert. Das <th>-Element kommt für Tabellenkopf zum Einsatz. In das <td>Element werden die Werte eingefügt.

```
<table>
    <tr>
        <td>Inhalt</td>
    </tr>
    <tr>
        <td>Inhalt</td>
    </tr>
</table>
```

Alle vorgestellten Elemente definieren die HTML-Struktur, zeichnen aber die Texte nicht semantisch aus. Dafür sind wesentlich mehr Elemente nötig als in HTML vorhanden sind. Um dieses Problem zu lösen, bietet HTML fast für jedes Element das Attribut @class, mit dem das HTML-Dokument semantisch strukturieren kann.[183]

```
<p class="first_paragraph"> Hier ist ein Absatz</p>
```

[180] vgl. Agular 2008, S.26
[181] vgl. Musciano 2002, S.58
[182] vgl. ebd.
[183] vgl. Ott 2014, S.57

6 Markdown

Markdown ist eine einfache Auszeichnungssprache, die von John Gruber und Aaron Swartz im Jahr 2004 entwickelt wurde. Das Hauptziel von Markdown besteht darin, den Text so einfach wie möglich zu lesen, zu schreiben und zu bearbeiten.[184] Das vorliegende Kapitel zeigt grundlegendes Wissen zur Markdown-Syntax auf. Hierfür werden alle wichtigen syntaktischen Regeln dieser Sprache vorgestellt. Die ursprüngliche Version von John Gruber ist nicht für alle Anwender ausreichend, weswegen zur grundlegenden Syntax viele weitere Elemente hinzugefügt wurden, die im Abschnitt 6.3 genauer betrachtet werden.

6.1 Hintergrund

Markdown hat eine einfache Syntax und kann in wenigen Minuten erlernt werden. Im Vergleich zu einer traditionellen Textverarbeitungssoftware oder anderen Formaten, wie HTML oder LaTeX, macht es Markdown möglich, die Texte mit weniger Befehlselementen auszuzeichnen und die Dokumente in viele Ausgabeformate zu konvertieren. [185]

Markdown funktioniert ähnlich wie HTML, aber seine Syntax ist sehr klein und entspricht nur einer sehr kleinen Teilmenge von HTML-Tags. HTML ist ein Veröffentlichungsformat, Markdown jedoch ein Schreibformat; daher befasst sich seine Formatierungssyntax nur mit Problemen, die im Klartext übermittelt werden können.[186]

Die Stärke seines Formats besteht darin, dass es immer einfach bearbeitet und dann in ein oder mehrere Formate exportiert werden kann. Ein in Markdown erstelltes Buchkapitel kann einfach in ein E-Book umgewandelt oder in eine PDF-Datei konvertiert werden. Diese Art von Flexibilität ist bei anderen Dokumentformaten nicht immer möglich oder einfach.[187]

Die Konvertierung erfolgt über einen Markdown-Editor oder mittels *Pandoc*, einem plattformübergreifenden Konvertierungstool, das Dokumente mit einer einfachen Befehlszeilensyntax von einem Format in ein anderes konvertiert.[188] Derzeit gibt es zahlreiche Erweiterungen der Markdown-Sprache (siehe Kap. 6.3). Pandoc hat seine eigene verbesserte Markdown-Erweiterung, die eine Syntax für Bibliografien und Zitate, Fußnoten, Codeblöcke, Tabellen, erweiterte Listen und Inhaltsverzeichnisse sowie eine eingebettete LaTeX-Mathematik enthält.[189]

[184] vgl. Gruber 2004
[185] vgl. Pellet 2020, S.120
[186] vgl. Gruber 2004
[187] vgl. Ovadia 2014, S.121
[188] vgl. MacFarlane o.J.
[189] vgl. MacFarlane o.J.

Einfache Auszeichnungssprachen zeichnen sich durch die Verwendung eines einfachen Satzes von Sonderzeichen aus, um verschiedene logische Inhaltsblöcke so zu bezeichnen, dass ihre visuelle Wahrnehmung so ist, wie sie ist, ohne die Einbeziehung zusätzlicher Werkzeuge.[190] So sind kursive Elemente, die mit Asterisk umgeben werden, einfach zu erkennen. Außerdem sehen die Darstellungen von Listen und Tabellen so aus, wie Listen und Tabellen in normalen Text eben aussehen sollten. Ein weiteres Beispiel für die Einfachheit der Markdown-Syntax sind Absatzumbrüche. In HTML erfordern diese die Verwendung von Tags, um den Inhalt zu strukturieren. Markdow*n* macht das Gleiche mit einem Zeilenumbruch, ähnlich wie ein Textverarbeitungsprogramm.[191]

Markdown ist also eine beliebte und fortschrittliche Methode zur schnellen Gestaltung von Inhalten, die aufgrund ihrer Vielseitigkeit und Einfachheit eine erhebliche Zeit beim Schreiben von technischen Texten, wissenschaftlichen Artikeln oder Lehrmaterial einspart.[192]

6.2 Markdown-Syntax

Alle syntaktischen Elemente, die Markdown definiert, haben ein entsprechendes Element in HTML. Ein weiterer grundlegender Aspekt, den Markdown mit HTML gemeinsam hat, ist die Trennung von Block-Level- und Span-Level-Elementen. [193]

Absätze in Markdown bestehen einfach aus einer oder mehreren aufeinanderfolgenden Textzeilen, die durch eine oder mehrere Leerzeilen getrennt sind. Dies unterscheidet sich erheblich von den meisten anderen Texten in HTML-Formaten, die jedes Zeilenumbruchzeichen in einem Absatz in ein
Tag umwandeln.[194]

Markdown bietet mehrere Syntaxen zur Darstellung eines Elements, wie zum Beispiel eine Überschrift. Es unterstützt zwei Header-Stile: Setext und Atx. Überschriften im Setext-Stil werden mit Gleichheitszeichen und Bindestrichen dargestellt, die Überschriften im Atx-Stil verwenden 1-6 Hash-Zeichen am Anfang der Zeile.[195]

```
# Überschrift 1          ## Überschrift 2
Überschrift 1            Überschrift 2
----------               ==========
```

[190] vgl. Badenko 2020, S.177
[191] vgl. Ovadia 2014, S.120
[192] vgl. Badenko 2020, S.177
[193] vgl. Rantakari 2011, S.3
[194] vgl. Gruber 2004
[195] vgl. ebd.

Um einen bestimmten Textbereich als Zitat zu kennzeichnen, verwendet Markdown Größer-als-Zeichen (>). Die Zitate können verschachtelt werden, indem zusätzliche Ebenen hinzugefügt werden (>>):[196]

```
> Dies ist ein Zitat.
> > Dies ist ein verschachteltes Zitat.
```

Markdown unterstützt geordnete und ungeordnete Listen. Die ungeordneten Listen verwenden Sternchen, Pluszeichen und Bindestriche, die geordneten Zahlen gefolgt von Punkten:

```
* eins        + eins        - eins        1. eins
* zwei        + zwei        - zwei        2. zwei
* drei        + drei        - drei        3. drei
```

Die Listenmarkierung fängt am linken Rand an und kann um bis zu drei Leerzeichen eingerückt werden, zwingend gefolgt von einem oder mehreren Leerzeichen oder einem Tabulator. Um ein Zitat in ein Listenelement einzufügen, müssen die (>) Trennzeichen des Zitats eingerückt werden.[197]

Mit der Markdown-Syntax können auch Codeblöcke erzeugt werden. Diese werden normalerweise um vier Leerzeichen oder einen Tabulator eingerückt. Vorformatierte Codeblöcke werden zum Schreiben von Programmier- oder Markup-Quellcodes verwendet. Anstatt normale Absätze zu bilden, werden die Zeilen eines Codeblocks wörtlich interpretiert. Markdown umschließt einen Codeblock sowohl in <pre>- als auch in <code>-Tags.[198]

```
Dies ist ein normaler Absatz.

    Dies ist ein Codeblock.
```

In HTML sieht ein Codeblock folgendermaßen aus:

```
<p> Dies ist ein normaler Absatz.p>
<pre><code> Dies ist ein Codeblock </code></pre>
```

Markdown verwendet Sternchen (*) und Unterstriche (__), um die Texte hervorzuheben. Ein einzelnes Symbol wird für Kursivschrift und ein doppeltes Symbol für Fettschrift verwendet.[199]

```
*Kursiver Text*      **Fetter Text**      ***Kursiver und fetter Text***
_Kursiver Text_      __Fetter Text_       ___Kursiver und fetter Text___
```

[196] vgl. Herrero 2013
[197] vgl. Gruber 2004
[198] vgl. ebd.
[199] vgl. ebd.

Markdown enthält zwei Arten von Links: Inline und Referenz. Die beiden Linkarten werden durch eckige Klammern ([]) eingeschlossen. [200]

```
Das ist [ein Link] (http://example.com/)
```

Links im Referenzstil verwenden einen zweiten Satz eckiger Klammern, in denen man eine Beschriftung der Wahl platzieren kann, um den Link zu identifizieren.[201]

```
Das ist [ein Link] [id]
```

Auch Bilder können in den Text integriert werden. Dies erfolgt durch eine Kombination aus eckigen und runden Klammern. Ähnlich wie bei den Links werden für die Bilder zwei Stile eingesetzt: Inline und Referenz[202]. Die Inline-Bild-Syntax sieht wie folgt aus:

```
![Alt text](/path/to/img.jpg)
```

Am Anfang wird ein Ausrufezeichen eingefügt, es folgt eine eckige Klammer, die den @alt-Attributtext für das Bild enthält; danach wird eine runde Klammer gesetzt, die die URL oder den Pfad zum Bild sowie ein optionales title-Attribut in doppelten oder einfachen Anführungszeichen enthält.[203]

Die Bildsyntax im Referenzstil sieht folgendermaßen aus: `![Alt text][id]`
Um ein literales Zeichen anzuzeigen, das sonst zur Formatierung von Text in einem Markdown-Dokument verwendet wird, soll ein Schrägstrich (\) vor dem Zeichen eingefügt werden.[204]

```
Die ist ein \*Beispiel mit Sternchen\*.
```

In Markdown gibt es mehrere Möglichkeiten, um eine horizontale Linie einzufügen: Es werden drei oder mehr Sternchen (***), Bindestriche (---) oder Unterstriche (___) in einer Linie eingefügt. Um eine horizontale Linie zu erzeugen, muss die Zeile davor leer bleiben. Die Zeile danach kann dann bereits den Text enthalten, der als Nächstes erscheinen soll.[205]

Viele Markdown-Anwendungen erlauben es, HTML-Tags in einem Markdown-formatierten Text zu verwenden. Die Verwendung von HTML ist beispielsweise dann hilfreich, wenn die Attribute eines Elements geändert werden, etwa die Farbe des Textes oder die Breite eines Bildes.[206] Eine Besonderheit dabei ist, dass die Markdown-Syntax innerhalb von HTML-Tags nicht verwendet werden kann. Es wird nicht funktionieren, wenn die Markdown-Syntax folgendermaßen geschrieben wird:

```
<p> *italic* und **bold**</p>
```

[200] vgl. Gruber 2004
[201] vgl. ebd.
[202] vgl. ebd.
[203] vgl. ebd.
[204] vgl. Cone 2020
[205] vgl. Herreo 2013
[206] vgl. Cone 2020

6.3 Markdown-Erweiterungen

Es gibt mehrere Erweiterungen, welche die Markdown-Syntax auf eine fortgeschrittene Weise ergänzen und modifizieren. Sie enthalten die grundlegende Syntax von Markdown und bauen darauf auf, indem sie zusätzliche Elemente wie Tabellen, Codeblöcke, Syntaxhervorhebungen, automatische URL-Verknüpfungen und Fußnoten hinzufügen.[207] Viele der beliebtesten Markdown-Anwendungen verwenden eine der folgenden einfachen Markup-Sprachen: MultiMarkdown, GitHub Flavored Markdown (GFM), Pandoc und noch viele weitere.

MultiMarkdown basiert auf Markdown und verfügt über einige zusätzliche Funktionen, die derzeit nicht in der einfachen Markdown-Syntax vorhanden sind, nämlich: Tabellen, Fußnoten, Zitate und Funktionen, die für die Ausgabe in verschiedene Formate vorgesehen sind.[208]

Um eine Tabelle hinzuzufügen, werden drei oder mehr Bindestriche (---) verwendet. Außerdem wird ein senkrechter Strich (|) eingefügt, um jede Spalte zu trennen. Aus Kompatibilitätsgründen sollte auch an beiden Enden der Zeile ein senkrechter Strich hinzufügt werden.[209]

```
| Syntax     | Bezeichnung |
| ---------- | ----------- |
| Header     | Title       |
```

Um eine Fußnotenreferenz zu erstellen, werden ein Zirkumflex und ein Bezeichner in eckigen Klammern ([^1]) hinzugefügt. Bezeichner können Zahlen oder Wörter sein, dürfen jedoch keine Leerzeichen oder Tabulatoren enthalten. In der Ausgabe werden die Fußnoten fortlaufend nummeriert, aber es ist nicht notwendig, die Fußnoten an das Ende des Dokuments zu setzen. Sie können überall platziert werden, außer in anderen Elementen wie Listen, Blockzitaten und Tabellen.[210]

```
Hier ist eine einfache Fußnote [^1]
[^1]: Dies ist die erste Fußnote.
```

GitHub verwendet GitHub Flavored Markdown (GFM), das sich in einigen wesentlichen Punkten vom Standard-Markdown unterscheidet und einige zusätzliche Funktionen bietet. Der erste Unterschied liegt in der Behandlung von Zeilenumbrüchen.[211] GFM behandelt Zeilenumbrüche in Absätzen als echte Zeilenumbrüche und ignoriert außerdem mehrfache Unterstriche in Wörtern: mehrere_Unterstriche_ignorieren.

[207] vgl. Cone 2020
[208] vgl. Leonard 2016, S. 10
[209] vgl. Cone 2020
[210] vgl. ebd.
[211] vgl. Herrero 2013

GFM unterstützt zudem die Codeblöcke. Das Einbetten von vorformatierten Codeblöcken ist möglich, indem jede Zeile des Blocks mit vier Leerzeichen oder einem Tabulator eingerückt wird. Es werden entweder drei Backticks (```) oder drei Tilden (~~~) in den Zeilen vor und nach dem Codeblock verwendet.[212]

```
```python
s = "Hello world"
print s
```
```

Pandoc enthält als erweiterte Version von Markdown eine Syntax für Tabellen, Definitionslisten, Metadatenblöcke, Fußnoten, Zitate, mathematische Formeln und vieles mehr. Die Erweiterungen können aktiviert oder deaktiviert werden, um das Verhalten genauer zu spezifizieren. Eine Erweiterung kann durch Hinzufügen +EXTENSION zum Formatnamen aktiviert und durch Hinzufügen von -EXTENSION deaktiviert werden.[213]

Metadaten können im Pandoc-Markdown-Format mit Metadatenblöcken oder in einer Markdown-Datei bereitgestellt werden. Metadaten entsprechen dem YAML-Format und werden durch Linien mit drei Bindestrichen (---) am Anfang und Ende eingefügt.[214]

```
---
title: "Instant Markdown"
author: Arturo Herreo
date: October, 2013
---
```

Die mathematischen Formeln werden innerhalb eines Paares von Dollarzeichen ($$) definiert. Die Formeln werden in allen Ausgabeformaten richtig wiedergegeben. Wie es gerendert wird, hängt vom Ausgabeformat ab.[215]

```
$$\\sum{\frac{(\mu - \bar{x})^2}{n-1}}$$
```

Um eine eingebettete TeX-Mathematik in der HTML-Ausgabe anzuzeigen, wird MathJax verwendet. Dies ist eine Open-Source-JavaScript-Bibliothek und ermöglicht es, mathematische Formeln in Markdown zu rendern. Außerdem können mathematische Inhalte in MathML, ein spezielles XML-Format zur Beschreibung mathematischer Inhalte, umgewandelt werden.[216]

[212] vgl. Cone 2020
[213] vgl. MacFarlane o.J.
[214] vgl. ebd.
[215] vgl. ebd.
[216] vgl. Mailund 2019, S.47

7 Lightweight DITA

Aus DITA wurde im Jahr 2017 die vereinfachte Version Lightweight DITA (LwDITA) abgeleitet, die im Vergleich zu DITA über einen kleineren Umfang an Funktionen, Elementen und Attributen verfügt. LwDITA definiert auch Mappings zwischen XML, HTML5 und Markdown und ermöglicht wechselweise den Einsatz von Markupsprachen.[217]

Das folgende Kapitel beschäftigt sich mit den Grundlagen von LwDITA. Es wird auf alle Authoring-Formate eingegangen, und es werden die Map- und Topic-Komponenten jeder Markupsprache angesprochen.

7.1 Hintergrund

LwDITA ist kein Ersatz für den DITA-Standard. Diese Version wurde für Personen oder Organisationen entwickelt, die mit dem DITA-Standard nicht vertraut sind. Wenn viel weniger Elemente und ein kleinerer Attributsatz für die Auszeichnung von Inhalten erforderlich sind, kann LwDITA als einfaches Format für die strukturierte Erstellung und Wiederverwendung von Inhalten verwendet werden.[218]

LwDITA hat folgende Ziele:

- Eine einfachere Verwendung des DITA-Standards.
- Ein Mapping zwischen XML, HTML5 und Markdown, das es dem Einzelnen ermöglicht
    - Inhalte im Format seiner Wahl zu erstellen,
    - den Austausch und die Veröffentlichung von Inhalten, die in verschiedenen Auszeichnungssprachen vorliegen, zu vollziehen sowie
- neue und kostengünstige Werkzeuge und Anwendungen zu fördern, die LwDITA unterstützen.[219]

LwDITA hat drei Authoring-Formate:

- XDITA – Eine XML-basierte Variante
- HDITA – Eine HTML5-basierte Variante
- MDITA – Eine Markdown-basierte Variante.[220]

[217] vgl. Evia 2019, S.10
[218] vgl. ebd.
[219] vgl. ebd.
[220] vgl. ebd.

LwDITA-Dokumente, erstellt in verschiedenen Formaten, können zusammengefasst und als ein einziges Dokument veröffentlicht werden. Sie lassen sich auch problemlos in einen DITA-Standard integrieren.

LwDITA hat weniger Funktionen als der DITA-Standard, die grundlegenden Funktionen für die Inhaltsstruktur und die Wiederverwendung sind aber bereits enthalten.[221] LwDITA bietet folgende Funktionen:

- Modularer Inhalt: Jedes Modul wird als Topic bezeichnet, das aus der Map gesammelt, organisiert und veröffentlicht werden kann. Die Topics werden als XDITA, HDITA oder MDITA erstellt und als ein einziges Dokument in PDF, HTML oder in anderen Formaten ausgegeben.
- Sammlung und Organisation von Inhalten: Die Topics werden in einer Map gesammelt und organisiert. Die Map kann zum Verwalten von Links, zur Navigation und für Metadaten für Topics verwendet werden. Map-Dokumente können auch in verschiedenen Markupsprachen erstellt werden. Eine Map kann in Markdown ausgezeichnet werden, und die in XML vorhandenen Topics können in dieser Map referenziert werden.
- Verlinkung: Ein Topic kann über einen Verweis mit einem anderen Thema (oder mit einer Position in einem Topic) verknüpft werden.
- Filtern: LwDITA unterstützt das metadatenbasierte Filtern innerhalb eines Topics. Mit dem Attribut `prop` werden die Werte eines Attributs identifiziert und weiterverarbeitet.
- Wiederverwendung: Mithilfe von Inhaltsreferenzen (`conref`) können die Inhalte auf Blockebene, wie beispielsweise ein Absatz oder ein Listenelement, von einem Ort an einen anderen verschoben und so wiederverwendet werden.
- Variablenverwaltung: In LwDITA kann ein variabler Inhalt in Maps definiert und in Topics über den Key-Referenzmechanismus wiederverwendet werden (Link zu `keyref`). Die Schlüsselreferenzen funktionieren über jedes der LwDITA Autorenformate.
- Spezialisierung: LwDITA folgt mit nur einigen Einschränkungen der gleichen Spezialisierungsarchitektur wie DITA 1.3. Da LwDITA mehrere Autorenformate umfasst, ist es besonders herausfordernd, die gleichen Spezialisierungsregeln in verschiedenen Auszeichnungssprachen zu koordinieren. Nicht alle LwDITA-Formate unterstützen die Spezialisierung in gleichem Maße.[222]

[221] vgl. Evia 2019, S.11
[222] vgl. ebd.

7.2 Authoring Formate

LwDITA wurde für Benutzer entwickelt, die nicht die volle Leistungsfähigkeit von DITA benötigen, aber eine ähnliche Struktur verwenden möchten. Bei der Entwicklung von XDITA wurden verschiedene Szenarien entwickelt,[223] die auf den folgenden Zielgruppen basieren:

- Informationsentwickler, die die Texte in XML auszeichnen möchten, aber einen kleineren Satz von Elementen und Attributen für die Arbeit benötigen.
- Abteilungen, die die Kosten für die Entwicklung und Pflege von Inhalten reduzieren wollen.
- Autoren, die ihre XML-Inhalte auf HTML5- oder Markdown-basierenden Inhalten zuordnen möchten.[224]

HDITA verwendet einige HTML5-Elemente und -Attribute, um die Funktionen von DITA darzustellen. Die XDITA- und HDITA-Inhaltsmodelle sind so konzipiert, dass sie funktional äquivalent zueinander passen. Das bedeutet, dass die Inhaltskomponenten beider Formate miteinander kompatibel sind.[225]

HDITA verwendet nur einen begrenzten Umfang von HTML5-Standard. Daher werden nicht alle Elemente dargestellt. Beispielsweise gibt es kein <h3>-Element, um eine Überschrift der Ebene 3 in HDITA auszudrücken. Wenn die unzulässigen Elemente in HDITA aufgeführt werden, wird in LwDITA-Tools ein Validierungsfehler hervorgerufen.[226]

HDITA hat folgende Zielgruppen:

- Software-Entwickler, die die Dokumentation mit HTML-Authoring-Tools erstellen.
- Lehrer und Trainer, die Kursinhalte für einen Kurs, eine Website oder ein Lernmanagementsystem (LMS) erstellen möchten.
- Blogger, die Inhalte mit mobilen Geräten erstellen und bearbeiten möchten.
- Autoren, die einfache Inhalte für das Web schreiben wollen.[227]

MDITA verwendet die Erweiterungen von *Markdown*. Die Grundstruktur von MDITA entspricht der GitHub-Flavored-Markdown-Spezifikation. Mit seinem erweiterten Profil kann MDITA YAML-Header und HDITA-Elemente und -Attribute verwenden.[228]

MDITA wurde für Benutzer entwickelt, die strukturierte Inhalte mit einem minimalen Aufwand erstellen, aber auch die Vorteile des DITA-Standards nutzen wollen. MDITA kann von

[223] vgl. Evia 2018, S. 82
[224] vgl. ebd., S.83
[225] vgl. ebd.
[226] vgl. ebd.
[227] vgl. ebd.
[228] vgl. ebd. S.84

technischen Redakteuren verwendet werden, die Markdown-Dateien in DITA- oder XDITA-Topic-Sammlungen einbinden möchten. Markdown kann auch jenen Autoren nützlich sein, die für die Dokumentation von Anwendungsprogrammierschnittstellen (APIs) zuständig sind und Inhalte mit technischen Publikationen oder Marketinginhalten austauschen müssen. Außerdem kann MDITA von Softwareentwicklern verwendet werden, die die Dokumentation einfach und ohne XML-Editor erstellen möchten.[229]

7.3 Map Komponente

Wie bereits im Kapitel 3 erwähnt, verwendet DITA ein Map-Dokument für die Sammlung und die Organisation von Inhalten. LwDITA-Map hat die gleichen Funktionen, hier aber gibt es noch ein wesentliches Merkmal: LwDITA-Map ermöglicht die Zusammenarbeit und die Veröffentlichung von Inhalten in verschiedenen Markupsprachen.[230]

Die LwDITA-Map enthält alle wichtigen Kernkomponenten des DITA-Standards. DITA-OT (siehe Kap. 9.3) unterstützt nur die XDITA-Map. Für die Einbindung von MDITA- und HDITA-Formaten wird derzeit nur die XDITA-Map verwendet.[231]

Die XDITA-Map hat folgende Struktur:

```
<map id="remote-main">
        <topicmeta>
            <navtitle>Remote Lighting Network</navtitle>
        </topicmeta>
        <topicref href="introduction.md format="mdita" "/>
</map>
```

Das Beispiel zeigt die reduzierte Anzahl von Elementen, welche die XDITA-Map verwendet. Das Element <map> ist das Wurzelelement, dem das @id-Attribut zugeordnet wird. <navtitle>bietet einen alternativen Titel für das Topic an. Das Element <topicmeta> definiert die Metadaten und das <topicref>-Element referenziert ein Topic. Das @format-Attribut identifiziert das Format der referenzierten Datei.[232]

Die HDITA-Map enthält alle Grundkomponenten der DITA-Map. Das Wurzelelement wird mit dem <nav>-Element definiert. Ein Titel wird mit <h1> angegeben. Für das Referenzieren von

[229] vgl. Evia 2018, S.84
[230] vgl. ebd., S.92
[231] vgl. ebd.
[232] vgl. ebd., S.97

Inhalten wird innerhalb einer Liste das Element <a> definiert. Über das @href-Attribut werden nur HDITA-Topics referenziert.[233]

Die Syntax des MDITA-Formats ist die einfachste der drei Authoring-Formate. Die einzigen Komponenten, die in einer MDITA-Version enthalten sind, sind ein Titel und eine Liste.[234] MDITA-Maps können nur auf MDITA-Topics verweisen.

IM Folgenden wird eine MDITA-Map dargestellt:

```
# Remote Lighting Network
        - [Introduction](introduction.md)
        - [Alternative lighting setups](alternatives.md)
        - [Low power installation](low-power.md)
```

7.4 Topic-Grundstruktur

Wie im Kapitel 3 erläutert, ist neben dem Map-Dokument eine wesentliche Einheit des DITA-Standards das Topic. LwDITA definiert auch ein Topic mit dem Element <topic>, in dem die weiteren Elemente aufgeführt sind, aber in reduzierter Anzahl.[235]

```
<topic id="contact_info">
   <title>Kontakt Information</title>
   <shortdesc>Kurze Beschreibung</shortdesc>
   <body>
      <p>So erreichen Sie uns...</p>
   </body>
</topic>
```

Nach dem <topic>-Element wird das <title>-Element definiert. <title> dient als Überschrift nicht nur für das ganze Topic-Dokument, sondern kann auch in anderen Blockelementen vorkommen.[236] Zwischen dem <title>- und dem <body>-Element kann das Element <shortdesc> eingesetzt werden. <shortdesc> dient dazu, eine kurze Beschreibung des Topics darzustellen, bevor ein Thema ins Detail geht. [237]

Der eigentliche Inhalt wird im Element <body> aufgeführt. Dieses Element beinhaltet alle wichtige Blockelemente, wie Absätze, Listen, Tabellen, Definitionslisten, Codeblöcke und vieles mehr. Das Element <section> bietet eine weitere Gliederungsmöglichkeit für die Texte. Hier

[233] vgl. Evia 2018, S.101
[234] vgl. ebd.
[235] vgl. ebd., S.103
[236] vgl. Evia 2019, S.14
[237] vgl. ebd., S.15

werden die einzelnen Abschnitte definiert. Die Abschnitte können auch Überschriften haben, die innerhalb des <title>-Elements stehen.[238]

Um ein Topic in HDITA darzustellen, wird das Element <article> verwendet. Dieses steht innerhalb des <body>-Elements. Wie das <topic>-Element in XDITA enthält <article> ebenfalls ein @id-Attribut für die Identifizierung des Topics. Der Titel wird den Elementen <h1> und <title> zugeordnet. Es gibt kein bestimmtes Element oder Attribut, um eine kurze Beschreibung zu geben. Diese wird im ersten Absatz ausgedrückt. Für die Gliederung von Texten verwendet HDITA auch das Element <section>.[239]

```
<head><title>Kontakt Information</title></head>
<body>
  <article id="contact_info">
    <h1>Kontakt Information</h1>
    <p>Kurze Beschreibung</p>
    <p>So kontaktieren Sie uns..</p>
  </article></body>
```

MDITA verwendet sehr wenige Auszeichnungselemente, um ein Topic darzustellen. Für die Identifizierung eines Topics muss in anderen Formaten ein @id-Attribut angegeben werden. Im Kapitel 6 wurde schon erwähnt, dass Markdown keine Attribute definiert. Um dieses Problem zu lösen, wird hier ein erweitertes Profil von Markdown eingesetzt. Das Pandoc-Markdown-Format definiert Metadatenblöcke oben in einer Markdown-Datei, indem ein @id für das Topic angegeben wird.

Die Überschriften werden im Setext- oder Atx-Stil dargestellt (siehe Kap. 6.2). Die Darstellung weiterer Abschnitte kann mit der Überschrift auf zweiter Ebene erfolgen. Hier hat Markdown keine bestimmte Auszeichnungselemente. Aufgrund der Einfachheit der Markdown-Syntax verwendet LwDITA zusätzlich die Markdown-Erweiterungen (siehe Kap. 6.3) und einige HTML-Elemente.[240]

[238] vgl. Evia 2018, S.101
[239] vgl. ebd., S.105
[240] vgl. ebd., S.108

8 ParsX

ParsX ist der einzige XML-Standard für Publikumsverlage im deutschsprachigen Raum. Der Standard ist XML-basiert und unterstützt die Verlage in allen Arbeitsabläufen vom Manuskripteingang bis zur Ausgabe in allen Medienkanälen.[241]

In diesem Kapitel wird der ParsX-Standard näher betrachtet und dabei auf die wichtigsten Aspekte eingegangen. Nach der Einführung in den ParsX-Standard stellt der Abschnitt 7.2 die ParsX-Grundstruktur vor und zeigt, aus welchen Elementen und Attributen sie besteht.

8.1 Hintergrund

ParsX wurde erstmals für die Publikumsverlage der Georg-von-Holtzbrinck-Gruppe entwickelt, wird aber heute von vielen größeren und kleinen Verlagen als zentrales Datenformat verwendet.[242] An der Entwicklung waren mehrere Publikums- und Sachbuchverlage beteiligt; es wurden also alle spezifischen Bedarfe der Verlagssparte berücksichtigt.[243]

Aufgrund der Digitalisierung stehen die Verlage vor großen Herausforderungen. Sie müssen ihre Inhalte nicht nur in gedruckter Form, sondern auch digital veröffentlichen. Dafür brauchen sie aber entsprechende Technologien und Softwarelösungen. Längst steht schon fest, dass die Anwendung branchenspezifischer Datenstandards eine Voraussetzung für den Erfolg des digitalen Zeitalters ist.[244] Es gibt mehrere XML-Standards, die für bestimmte Verlagssparten entwickelt wurden. Beispielsweise wird die *Text Encoding Initiative* (TEI) für geisteswissenschaftliche Verlage eingesetzt, die *National Library of Medicine* (NLM-DTD) für naturwissenschaftliche Publikationen und DITA für technische Dokumentationen und vieles mehr. Bevor der ParsX-Standard entstand, gab es keinen Standard für die Publikumsverlage im deutschsprachigen Raum.[245] ParsX stellt aber einen XML-Standard zur Verfügung, der alle Anforderungen für einen modernen Verlag abdeckt. So können Verlage ihre Herausforderungen meistern und die digitale Transformation erfolgreich umsetzen.

ParsX ist modular aufgebaut; daher bekommen die Verlage nur das Angebot, das für sie zugeschnitten ist. Mit Modulen ist beispielsweise eine automatisierte Leseprobenproduktion möglich, außerdem können multimediale E-Books erstellt werden. Es können aber auch E-Book-Profile individuell angepasst werden und vieles mehr.[246] Pagina entschied sich bei ParsX ein

[241] vgl. Ott 2018, S.1
[242] vgl. Ott o.J.
[243] vgl. Ott 2019, S.1
[244] vgl. ebd.
[245] vgl. ebd.
[246] vgl. Ott 2018, S.3

XML-Schema zu verwenden. Dieses XML-Schema ist ebenfalls modular aufgebaut, und den Verlagen werden nur die für sie passenden Elemente und Attribute angeboten.[247]

Das XML-Schema ermöglicht eine Validierung von XML-Daten, die jedoch oft nicht ausreichend ist. Die Verlage wollen in ihren Daten wesentlich mehr prüfen, als es das XML-Schema kann. Zu diesem Zweck wurde in ParsX ein Schematron-Konventionsprüfer eingebaut. Dieser ermöglicht eine Plausibilitätsprüfung und kann an spezifische Bedürfnisse angepasst werden.[248]

Ein weiterer Qualitätssicherungsbaustein von ParsX ist der *DokuChecker*; mit seiner Hilfe werden sämtliche strukturellen Besonderheiten der XML-Datei abgeprüft. Jedes einzelne Zeichen wird unter die Lupe genommen, und als Ergebnis wird eine PDF-Datei mit Fehlermeldungen und Dokumentationen ausgegeben.[249]

Zur Verarbeitung der XML-Daten wird der XML-Editor *Oxygen* von *SyncroSoft* verwendet. Oxygen bietet eine angepasste ParsX-Benutzeroberfläche und eine Schaltflächensteuerung für alle Module. Im Editor können auch E-Books und Leseproben aus dem ParsX-Framework erstellt werden.[250]

Ein weiteres Werkzeug für die Bearbeitung von XML-Daten ist das *ParsX-InDesign-PlugIn*. Seine erweiterten Funktionen erlauben es, die XML-Daten sauber zu produzieren. Daneben ist der E-Book-Konverter ein mächtiges Werkzeug, das rund 200 Konfigurationsmöglichkeiten für ein E-Book-Profil definiert. Auf Knopfdruck können E-Books mit verschiedenen Profilen erstellt werden. Der eingebaute Pagina-GlyphenChecker und der Pagina-FontSubsetter sorgen für die weitere Qualitätssicherung.[251]

Der PDF-Previewer kann verwendet werden, um eine Vorschau von XML-Daten zu generieren. Mit den Vorlagen, die mit PrintCSS oder XSL-FO erstellt wurden, können die PDF-Dokumente mit unterschiedlichen Layouts ausgegeben werden. Einen XSL-FO-Renderer gibt es schon in der Basisversion. Jeder, der sich mit PrintCSS gut auskennt, kann Buchlayouts selbst erstellen oder vorhandene Templates nutzen.[252]

ParsX bietet folgende Module an:

- das Multimedia-Modul, eine Erweiterung von XML-Schema und E-Book-Konverter, für die Einbindung von multimedialen Inhalten.[253]

[247] vgl. Ott 2018, S.2
[248] vgl. ebd., S.3
[249] vgl. ebd., S.4
[250] vgl. ebd.
[251] vgl. ebd.
[252] vgl. ebd., S.5
[253] vgl. ebd.

- das Dramensatz-Modul für die Auszeichnung komplexer Bühnendialoge, Regieanweisungen oder ähnlicher Inhaltsblöcke.
- das Import-Modul, welches es ermöglicht, externe Werke oder Kapitel externer ParsX-Projekte in die XML-Struktur zu referenzieren und als Leseprobe anzubieten.
- die Datenbank-Schnittstelle/Kommandozeilen-Schnittstelle: Sämtliche ParsX-Module können außerhalb des Oxygen-Frameworks aufgerufen werden. Dies erlaubt es den Nutzern, E-Books oder PDF-Dateien von Kommandozeilen zu erstellen.
- Das Index-Modul bindet die komplexen Register in verschiedenen Varianten, und
- der ParsX-Serials-Generator splittet als eine Erweiterung von ParsX die Werke in einzelnen Serials auf.[254]

8.2 Grundstruktur

Jedes ParsX-Dokument beginnt mit einer XML-Deklaration, die die XML-Version identifiziert und mitteilt, welche Zeichenkodierung dieses Dokument verwendet:[255]

```
<?xml version="1.0" encoding="utf-8"?>
```

In der zweiten Zeile steht die URL für das Schematron-Schema, das die Validierung von ParsX-Dokumenten sicherstellt. Darauf folgend wird dann das Element <projekt> angegeben, das die oberste Ebene in einem Dokument darstellt. Die ParsX-Datei ist folgendermaßen aufgebaut:

```
<projekt>
  <meta>
      <global>
        <titel>parsX Projekt Template</titel>
      </global>
  </meta>
  <werk>
    <hauptteil>
        <kapitel>
          <abs>Inhalt des Werkes</abs>
        </kapitel>
    </hauptteil>
  </werk></projekt>
```

Im Element <meta> werden die Metainformationen zum Inhalt gespeichert. Die Metainformationen enthalten die bibliografischen Angaben über den Inhalt sowie andere Informationen,

[254] vgl. Ott 2018, S.5
[255] vgl. Schmull o.J.

wie enthaltene Sonderstrukturen und Besonderheiten, Schnittstellen zum Austausch von Informationen mit externen Systemen und vieles mehr.[256]

Im Element <werk> werden die eigentlichen Inhalte des Dokuments eingefügt. Diese unterteilen sich in drei Bereiche: <vorspann>, <hauptteil> und <nachspann> und weiter in <kapitel>-Elemente. In jedem Kapitel wird der Text abschließend strukturiert, zum Beispiel mit Titeln, Absätzen, Listen oder Tabellen.[257]

Mit sechs Überschriften-Tags, geschrieben als <u1>, <u2>, <u3>, <u4>, <u5>, <u6> und <u7> werden die Überschriften entsprechend ihrer Ebene ausgezeichnet. In den meisten Fällen entspricht die Überschriftebene der Kapitelhierarchie. Die Überschriftebene <u1> wird in Hauptkapitel angegeben, ein Unterkapitel erhält eine <u2>, das Unterkapitel eines Unterkapitels eine <u3> und so weiter.[258] Es kann auch vorkommen, dass die Überschriften in der Kapitelhierarchie nicht in ihrer Reihenfolge aufgeführt werden. Beispielsweise kann die Überschriftebene <u2> im <vorspann> für ein Vorwort angegeben werden.[259] Innerhalb des <u1>-Elements gibt es das weitere Element <titel>, welches wiederum das Element <text> enthält, in das der Überschriftstext eingetragen wird.

```
<kapitel>
  <u1>
    <titel>
      <text>Überschrift erster Ebene</text>
    </titel>
  </u1>
<abs> Das ist ein Absatz. </abs>
<abs> Das ist ein Absatz. </abs>
  </kapitel>
```

Mit dem Element <abs> wird ein Absatz definiert. <abs> kann innerhalb von Listen und Tabellen sowie Fuß- und Endnoten vorkommen.[260]

ParsX stellt mehrere Inline-Elemente zur Verfügung, die zur Auszeichnung von Fließtext verwendet werden. Wenn der Text fett ausgezeichnet werden soll, wird das Element <fett> verwendet, und mit dem Element <kursiv> wird der Text kursiv ausgezeichnet. Die Texte können auch hoch oder tief gestellt werden. Ein hochgestellter Text wird in einer kleineren Schriftart

[256] vgl. Schmull o.J.
[257] vgl. ebd.
[258] vgl. ebd.
[259] vgl. ebd.
[260] vgl. ebd.

wiedergegeben und steht über der normalen Linie des Textes, während der tiefgestellte Text unter der normalen Linie steht.[261]

ParsX verwendet auch die Elemente <unterstr> und <durchgestr>, wenn die Texte unterstrichen oder durchgestrichen werden sollen; außerdem können die Versalien, Ziffern, Initialen oder Sonderstrukturen ausgezeichnet werden.[262]

ParsX verwendet zwei Arten von Listen: geordnete und ungeordnete Listen. Die geordnete Liste wird mit dem Element <liste_geordnet> definiert und in einer bestimmten Reihenfolge erstellt. Die ungeordnete Liste wird mit dem Element <liste_ungeordnet> bezeichnet und ohne eine bestimmte Reihenfolge dargestellt.[263]

In der Regel werden die geordneten Listen arabisch nummeriert. Wird aber ein anderes Nummerierungsformat gewünscht, dann kann mit dem @zaehlertyp-Attribut ein entsprechender Nummerierungstyp definiert werden.[264] Es ist auch möglich, die Listenelemente mit abweichender Zählerweise auszuzeichnen. Dafür wird das Element <liste_manuell> verwendet.

Die Listenelemente können auch verschachtelt werden. Beispielsweise kann eine untergeordnete Liste in eine übergeordnete Liste eingefügt werden. Die einzelnen Listenpunkte werden mit dem Element <eintrag> und <abs> ausgezeichnet.

```
<liste_geordnet>
        <eintrag>
          <abs>Nummerierter Aufzählungspunkt 1</abs>
        </eintrag>
        <eintrag>
          <abs>Nummerierter Aufzählungspunkt 2</abs>
        </eintrag>
</liste_geordnet>
```

Mit dem Element <bild> wird eine Abbildung eingefügt. Die Bilder werden jedoch technisch nicht direkt eingefügt, sondern nur verlinkt. Dafür wird das leere Element <abbildung> verwendet, das ein wichtiges Attribut @quelle enthält, in dem der Pfad zum Bild angegeben wird.[265]

```
<bild>.<abbildung quelle="abbildung.jpg"/></bild>
```

ParsX verwendet HTML-ähnliche Elemente, um Tabellen auszuzeichnen. Die Elemente <tabelle> und <table> schließen eine ganze Tabelle ein. Der Tabellenkörper wird mit dem Tag

[261] vgl. Schmull o.J.
[262] vgl. ebd.
[263] vgl. ebd.
[264] vgl. ebd.
[265] vgl. ebd.

<tbody> gebildet. Das Element <tr> definiert eine Tabellenzeile, die mit dem <td>-Tag ausgezeichnet werden. Mit dem Element <thead> wird eine Tabellenkopfzeile definiert. Die Zeilen- oder Spaltenüberschriften werden mit <th> ausgezeichnet.[266]

Für die Auszeichnung semantischer Strukturen enthält ParsX entsprechende Elemente. Wenn im Dokument Formeln vorhanden sind, werden diese entweder innerhalb von <formel> als eine Abbildung eingefügt, oder mit dem XML-Dokumentformat *MathML* dargestellt. ParsX definiert auch Zitate, die mit dem Element <zitat_inline> oder auf der Blockebene mit <einschub type="zitat"> bezeichnet werden.[267]

```
<einschub typ="zitat">
   <abs>Ein Zitat auf Blockebene</abs>
   <abs>Ein Zitat auf Blockebene</abs>
</einschub>
```

ParsX definiert mit den Elementen <fussnote> und >endnote> die Fußnoten und Endnoten. Das Fußnotenelement wird in einem Absatz an der Stelle eingefügt, an der die Fußnotenreferenz (Zahl oder Symbol) erscheinen soll.[268] Bei der Verarbeitung erscheint das Fußnotenreferenzsymbol dort, wo das <footnote>Start-Tag erscheint. Zum Beispiel:

```
<abs>Das ist eine Fußnote <fussnote><abs>ich bin eine Fußnote</abs></fussnote></abs>
```

Die Verlinkung im ParsX-Dokument wird mit dem Element <verweis> definiert. ParsX unterscheidet zwischen internen und externen Verweisen. Mit dem Attribut verweis-intern werden Verweise innerhalb eines Dokumentes angegeben. Das Verweisziel wird mit dem Attribut id zugewiesen und als Hyperlink umgesetzt. Die externen Ressourcen werden mit dem Attribut verweis-extern referenziert.[269]

[266] vgl. Schmull o.J.
[267] vgl. ebd.
[268] vgl. ebd.
[269] vgl. ebd.

9 Technische Merkmale von MixedMarkup

Im Kapitel 7 wurde bereits die Grundlage von Lightweigt DITA dargestellt. In diesem Kapitel werden nun die technischen Eigenschaften von LwDITA aufgezeigt, innerhalb deren der MarkupMix funktioniert. Dazu werden die LwDITA-Komponenten genau analysiert und die für die Transformation benötigten Technologien vorgestellt.

Um LwDITA-Dokumente in das Zielformat zu überführen, werden drei Komponenten benötigt:

(1) Topic: Wie bereits erwähnt, ist „das Topic das zentrale Informationsobjekt in DITA, in dem Informationen erfasst und gespeichert werden."[270] In LwDITA können die Topics in verschiedenen Markupsprachen erfasst und in einer Map referenziert werden.

(2) Map: Eine Map dient der Sammlung und Organisation von Topics. Wie der Verweis zu den Topics funktioniert, wird im Abschnitt 9.2 erläutert.

(3) Transformation: Es gib drei Möglichkeiten, um eine gewünschte Ausgabe aus einem Map-Dokument produzieren zu können. Die erste davon ist ein LwDITA Plug-In für DITA Open Toolkit (DITA-OT), die zweite Möglichkeit bietet Oxygen Batch Converter Plug-In, und die dritte Variante ermöglicht die Konvertierung von Markdown in HTML und die Transformation von HTML in ein DITA-Dokument. Eine ausführliche Beschreibung dieser drei Möglichkeiten erfolgt im Abschnitt 9.3.

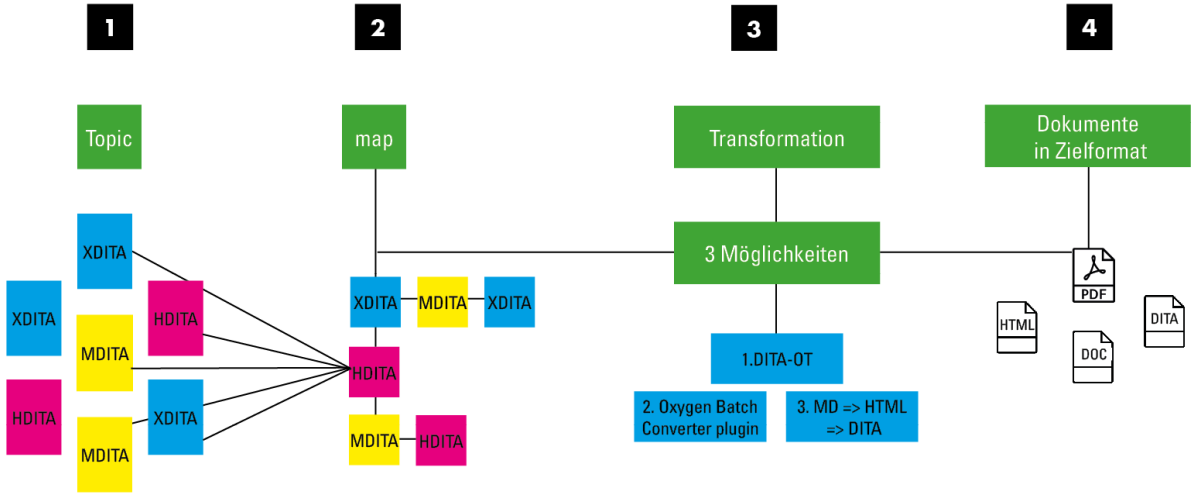


Abbildung 1: DITA-Komponente

[270] Hentrich 2008, S.156

9.1 XDITA-Topic

In einem Topic werden die Inhalte als unabhängige Informationsblöcke aufgeteilt und in einer eigenen Datei gespeichert.[271] Ein Topic muss einen Titel und optional ein <body>-Element enthalten. Die Abbildung 2 zeigt ein Topic (1) als einen Inhaltsblock, in dem die Strukturierung des Textes erfolgt, zum Beispiel mit Überschriften (2), Abschnitten (3), Absätzen, oder Listen (4).

Die Topics werden in einem Map-Dokument referenziert. Die Verweise dürfen nicht unterhalb des Topics liegen; es kann also nicht auf einzelne Elemente innerhalb eines Topics verwiesen werden.[272]

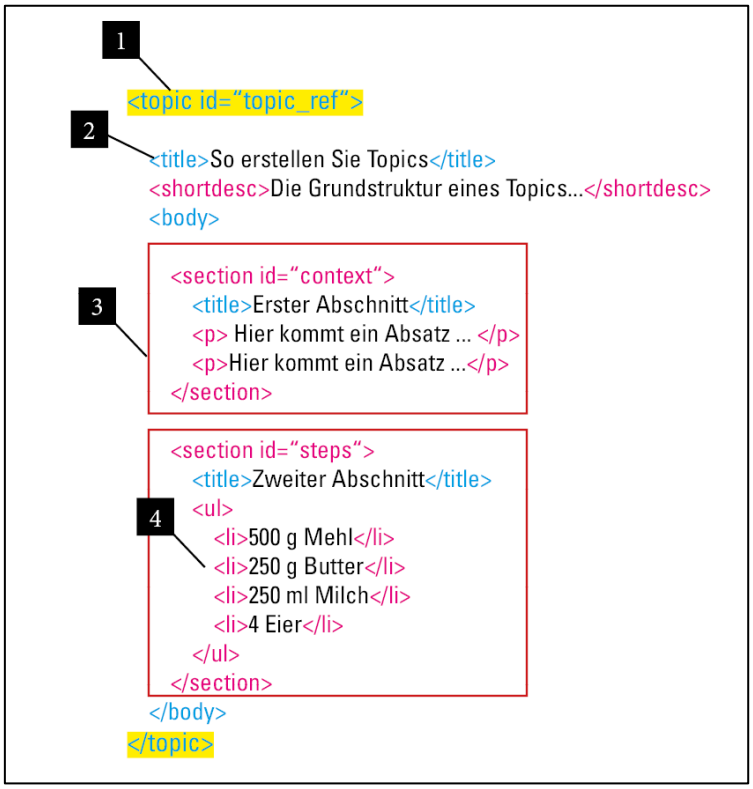


Abbildung 2: XDITA-Topic

Ein weiteres Merkmal eines Topics ist die Kompatibilität. In jedem Format müssen die Topics strukturell miteinander kompatibel sein. Das heißt, wenn der Inhalt in einem MDITA- oder HDITA-Dokument erstellt wird, sollte wie in XDITA ein Titel und ein @id-Attribut für die Identifizierung eines Topics angegeben werden. Außerdem werden die Überschriften bis zur zweiten Ebene definiert. Die MDITA und HDITA-Formate verwenden eine Teilmenge ihre Konstrukte, um die Kompatibilität mit LwDITA-Dokumenten zu gewährleisten.

[271] vgl. Day 2005, S.5
[272] vgl. ebd.

9.2 XDITA-Map

Das wesentliche Merkmal einer Map besteht darin, die Topics in strukturierten Informationssammlungen zu organisieren.[273] Die Topics können in drei verschiedenen Markupsprachen erstellt und in einer XDITA-Map referenziert werden. Um ein Mapping zwischen verschiedenen Markupsprachen zu ermöglichen, kommen drei wichtige Elemente zum Einsatz:

- das Element <topicref> (1)
- das Attribut @href (2)
- und noch ein weiteres Attribut @format (3)

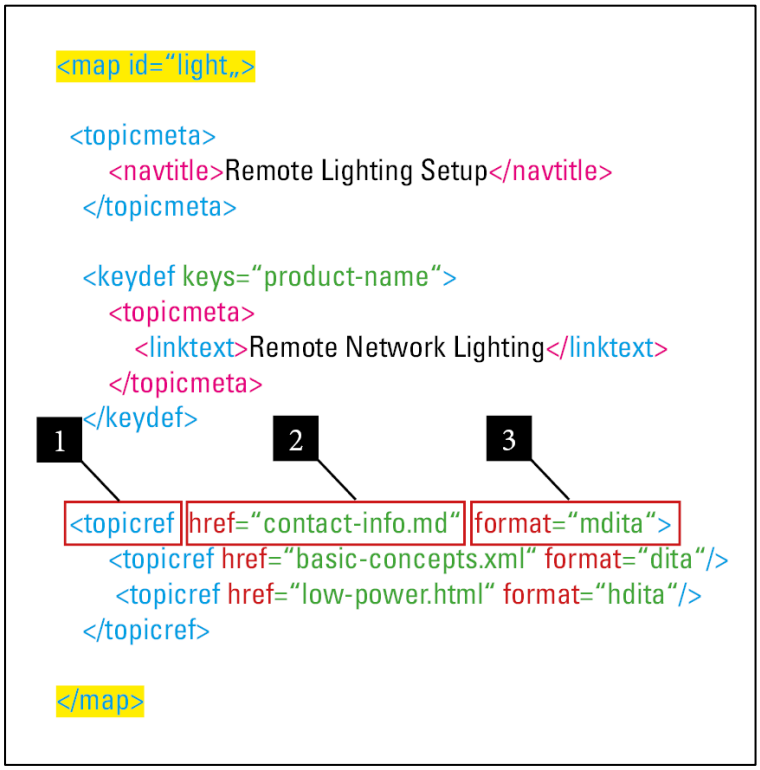


Abbildung 3: XDITA-Map

Das <topicref>-Element ist der grundlegende Baustein in einem Map-Dokument und ähnelt in seiner Funktion dem <a>-Element von HTML. Hier wird ein Hyperlink definiert, der verwendet wird, um auf ein Topic in einer Map zu verweisen.

Das Ziel des Links gibt das Attribut @href an, das dem <topicref>-Element zugeordnet wird. Der Wert eines DITA-@href-Attributs muss eine gültige URI-Referenz[274] sein.[275] In der Abbildung 3 ist der Dateiname eines Topics „contact-info.md" als Verweisziel angegeben. Dieses Topic ist in einer Markdown-Datei erstellt. Zur Identifizierung von Dokumentformaten wird das

[273] vgl. Anderson 2018, S.42
[274] Ein Uniform Resource Identifier (URI) ist eine kompakte Folge von Zeichen, die eine abstrakte oder physische Ressource identifizieren.
[275] vgl. Anderson 2018, S.43

Attribut @format vorgeschrieben. Mögliche Werte für das @format-Attribut sind dita, mdita, hdita. Für Markdown-Dokumente wird als Attributwert mdita festgelegt.

9.3 Transformation

LwDITA-Maps können mit mehreren Transformationsszenarien in eine Vielzahl von Ausgaben, wie PDF, XHTML oder EPUB, generiert werden. Bevor die LwDITA-Maps in ein Zielformat überführt werden, sollten die referenzierten Topics, die in verschiedenen Formaten angelegt sind, in das LwDITA-Projekt integriert werden. Es gibt verschiedene Optionen, um die Verwendung von MDITA und HDITA als Quelldokumentformat zu ermöglichen:[276]

Der DITA-Standard verwendet DITA Open Toolkit zum Verarbeiten von DITA-Dokumenten. Das Tool enthält die XSLT-Stylesheets für die Produktion von Ausgabemedien.[277] Um weitere Funktionen hinzuzufügen, verwendet DITA-OT mehrere Plug-Ins. So werden zum Beispiel die Markdown- und HTML-Dateien über das LwDITA-Plug-In unterstützt.[278] Das LwDITA-Plug-In enthält einen benutzerdefinierten Parser für Markdown- und HTML-Dokumente und einen Transformationstyp zum Generieren von Markdown aus einem DITA-Dokument. DITA-OT-Plug-Ins können einen völlig neuen Transformationstyp hinzufügen.[279] Dies kann in einem <transtype>-Element definiert werden:

```
<transtype name="markdown"/>
```

Darüber hinaus können mithilfe des Oxygen Batch Converter Plug-In die Markdown- und HTML-Dateien in DITA-Formate konvertiert werden. Das Plug-In enthält eine Liste der verfügbaren Konvertierungstypen, wie HTML to DITA, Markdown to DITA, HTML to XHTML, Markdown to DocBook4 und andere mehr. Wenn einer der Konvertierungstypen ausgewählt wird, wird ein Dialogfeld im Oxygen XML Editor geöffnet, in dem die Optionen für die Konvertierung konfiguriert werden können.[280]

Eine dritte Möglichkeit bieten der Pandoc-Konverter und der XSLT-Transformation. Pandoc ist ein befehlszeilenbasiertes Tool, das mehrere Dokumentenformate ineinander konvertiert. Mithilfe von Pandoc werden die Markdown-Dateien in ein HTML-Dokument konvertiert. Im nächsten Schritt wird die XSLT eingesetzt, die das HTML-Dokument in das DITA-Format transformiert.

[276] vgl. Jitianu 2019, S.19
[277] vgl. Anderson 2019
[278] vgl. ebd.
[279] vgl. Elovirta 2017
[280] vgl. Cosmin 2021

10 Anwendung des LwDITA-Ansatzes auf ParsX

Nachdem im vorhergehenden Kapitel die technischen Merkmale analysiert wurden, soll nun überlegt werden, wie diese auf andere XML-Standards angewendet werden können. Die Übertragbarkeit dieser Merkmale wird durch eine beispielhafte Anpassung des Branchenstandards ParsX geprüft.

LwDITA besteht aus zwei wichtigen Bausteinen: Topic und Map. Diese Struktur muss auf ein ParsX-Dokument übertragen werden. Es soll einerseits ein Inhaltsobjekt (ähnlich wie Topic) definiert werden, andererseits eine Map-Struktur, in der diese Inhaltsobjekte referenziert werden.

Die folgende Auflistung fasst noch einmal die wesentlichen technischen Merkmale von Mixed-Markup zusammen:

- In einer Map werden die Topics referenziert, nicht die einzelnen Elemente, die innerhalb eines Topics definiert sind. Außerdem sollten Topics in jedem Format strukturell angepasst werden, damit sie untereinander kompatibel sind.
- Eine Map besteht aus drei wichtigen Elementen, die ein Mapping zwischen verschiedenen Markupsprachen ermöglichen: das Element <topicref> und die Attribute @href und @format.
- Um alle referenzierten Topics in das LwDITA-Projekt zu integrieren, ist eine Transformation erforderlich. Dabei werden verschiedene Transformationsmöglichkeiten eingesetzt.

Nachfolgend wird die ParsX-Dokumentstruktur analysiert und geprüft, inwieweit alle bereits definierten Merkmale auf den ParsX-Standard angewendet werden können.

10.1 ParsX-Kapitel

In diesem Abschnitt wird ein Inhaltsobjekt definiert, in dem die Texte strukturiert werden. In LwDITA haben wir ein Topic als Inhaltsobjekt. In einem ParsX-Dokument nimmt diese Funktion das Element <kapitel> ein. <kapitel> enthält den eigentlichen Inhalt und besteht aus vielen weiteren Inline- und Blockelementen, wie Absätze, Überschriften, Listen, Tabellen oder Unterkapitel.

Gemäß dem LwDITA-Ansatz können die Topics in unterschiedlichen Markupsprachen angelegt werden. Eine Voraussetzung dafür ist die Kompatibilität zwischen allen Formaten.

In der Abbildung 4 ist ein Kapitel in Markdown und ParsX angelegt. Beide Formate haben die gleichen Strukturen und enthalten die gleichen Inhaltselemente, wie Überschriften (1), Absätze (2), weitere Kapitel (3) und ungeordnete Listen (4). Der einzige Unterschied besteht hier in den Auszeichnungselementen.

Wenn ein Kapitel als Markdown oder HTML angelegt wird, sollte dieses strukturell an ein ParsX-Kapitel angepasst werden. Inwieweit die Markdown-Struktur in der ParsX-Kapitelstruktur abgebildet werden kann, wird im Abschnitt 10.3 beschrieben, indem eine Transformation erfolgt.

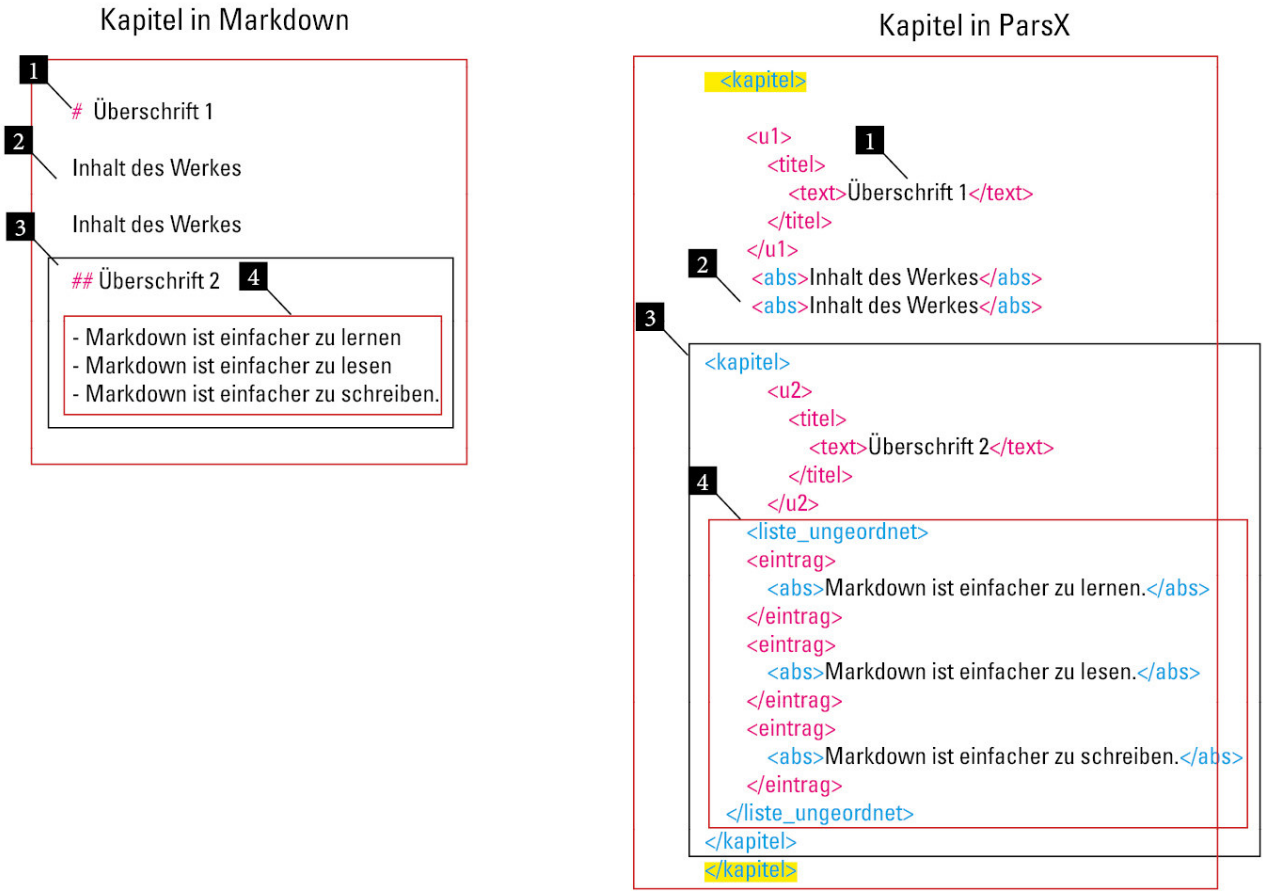


Abbildung 4: Kapitel in Markdown und ParsX

10.2 ParsX-Map

In folgenden Abschnitt wird die ParsX-Map dargestellt, in der die Kapitel referenziert werden. Für den Verweis zu den Kapiteln sind hier drei grundlegende Elemente erforderlich: ein Element ähnlich wie <topicref>, das einen Hyperlink definiert, und zwei Attribute. Das erste Attribut gibt ein Linkziel an, das zweite identifiziert die Dokumentenformate.

Bevor diese Elemente definiert werden, sollte zuerst die ParsX-Grundstruktur genau betrachtet werden:

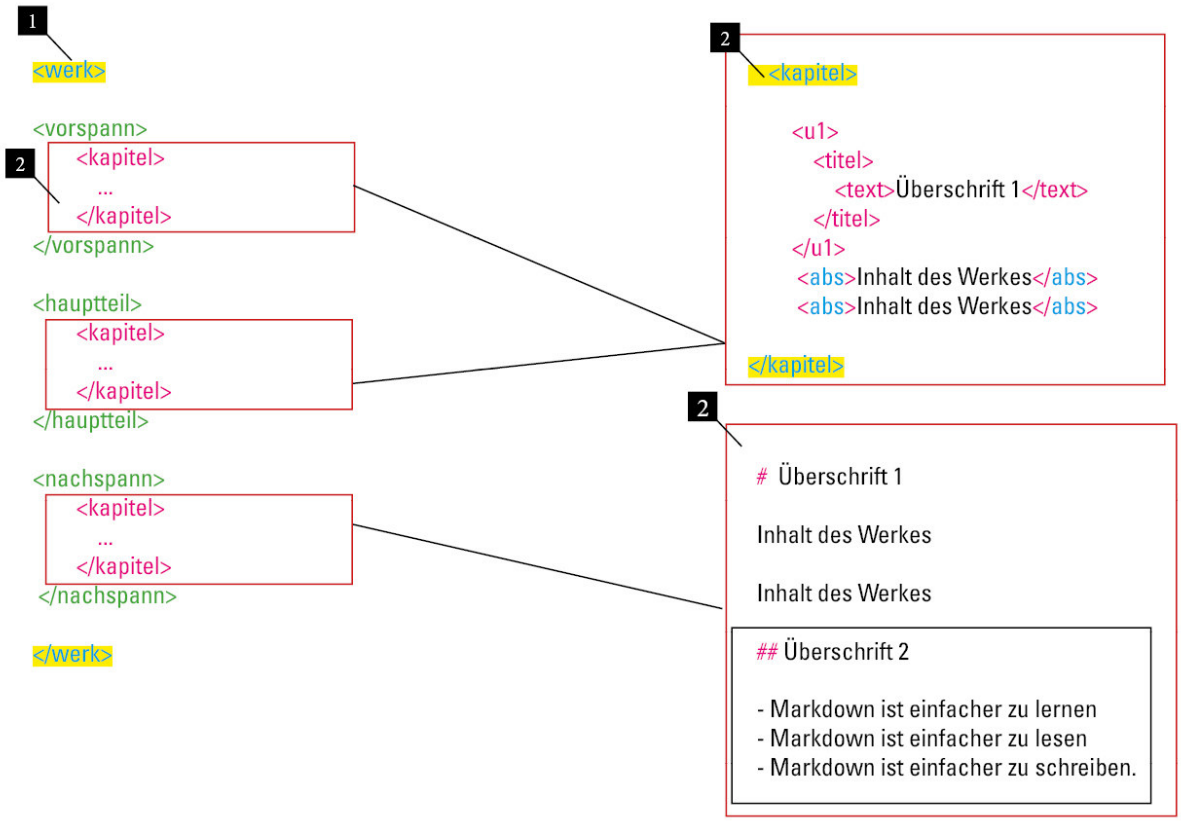


Abbildung 5: ParsX-Grundstruktur

Im Element <werk> (1) werden die eigentlichen Inhalte des Dokuments eingefügt. Diese unterteilen sich in drei Bereiche: <vorspann>, <hauptteil> und <nachspann> und weiter in <kapitel>-Elemente (2). In jedem Kapitel wird der Text abschließend strukturiert.[281]

Für das Referenzieren der Kapitel, sollte in den Elementen: <vorspann>, <hauptteil> und <nachspann> ein Element aufgeführt werden, das einen Hyperlink definiert. Eine ähnliche Funktion

[281] vgl. Schmull o.J.

in ParsX enthält das Element <import-kapitel>, das zum Einbinden von Kapiteln aus bestehenden Projekten verwendet wird.[282] Durch ein verpflichtendes @system-ressource Attribut wird der URI einer externen Ressource eingegeben, in diesem Fall der Dateiname des Kapitels.

ParsX definiert kein Element oder Attribut für die Identifizierung der Dateiformate. Daher wurde dem ParsX-Schema das Attribut @format hinzugefügt. Dem @format-Attribut wird das <import_kapitel>-Element zugeordnet.

In der folgenden Abbildung ist eine ParsX-Map angelegt. <import_kapitel> verweist auf das Kapitel mit dem Dateinamen „in.md“ (2). Die Datei ist im Markdown-Format erstellt, weshalb als Wert für das @format-Attribut auf md gesetzt wird (3).

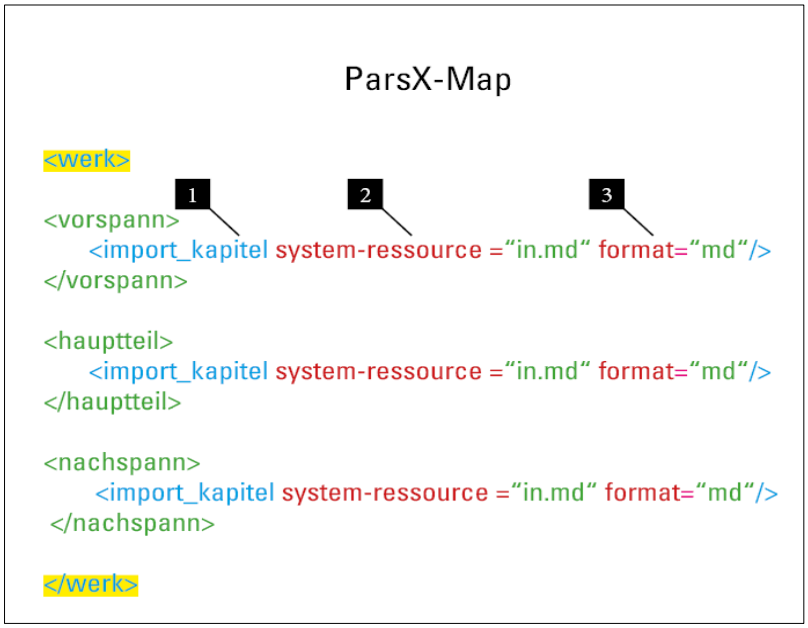


Abbildung 6: ParsX-Map

10.3 Transformation

Nach der vorangegangenen Analyse wird in diesem Abschnitt die Übertragbarkeit von definierten Merkmalen auf den ParsX-Standard geprüft. Dies erfolgt durch eine beispielhafte Transformation der ParsX-Map. Im Abschnitt 9.3 wurden mehrere Möglichkeiten für eine Transformation vorgestellt. In den ersten beiden Varianten werden die Markdown-Dateien im DITA-Standard mithilfe eins Plug-Ins transformiert. Für ParsX gibt es aber bisher kein Plug-In, das mehrere Formate in ParsX transformieren kann; daher wird in dieser Arbeit die dritte Variante verwendet, in der der Pandoc-Konverter und die XSTL-Transformation eingesetzt werden.

[282] vgl. Dünckel 2021

Für die Umsetzung wurde der Ordner *„mapTransform“* angelegt, in dem die folgenden Dateien vorhanden sind:

- Unterordner *„xsd“*, der das ParsX-Schema mit allen Modulen beinhaltet.
- Unterordner *„xsl“* mit zwei XSLT-Dateien: die erste für die Transformation von HTML in ParsX und die zweite für die Konvertierung der ParsX-Map in ein komplettes ParsX-Dokument.
- Markdown-Datei: *„in.md“.*
- ParsX-Map-Datei: *„ParsXMap.xml“.*

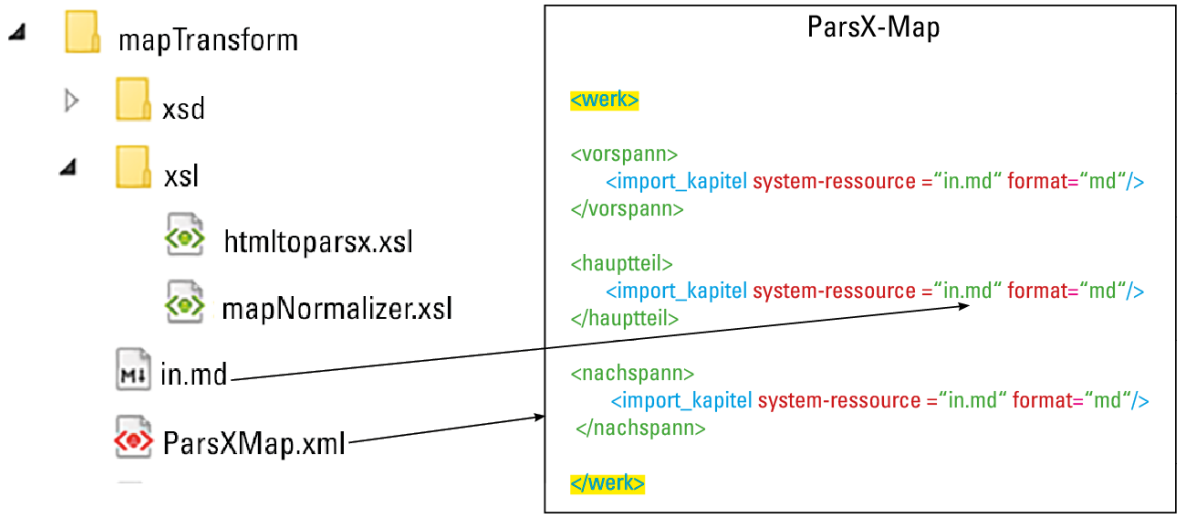


Abbildung 7: Ordnerstruktur

Alle Vorgänge zwischen der Konvertierung von Markdown bis zur kompletten ParsX-Datei werden von der Datei *„mapNormalize.xsl“* ausgelöst. Die Transformation funktioniert folgendermaßen:

Zuerst wird mithilfe des Pandoc-Konverters aus der Markdown-Datei ein HTML-Dokument erzeugt. Das Dokument wird dann in den Ordner *„xsl“* eingefügt.

```
<xsl:variable name="call" select="concat('pandoc -s  --from markdown --mathml --to html5
',$path,'/',$filename,' -o ',$path,'/xsl/',$filenamehtml)"/>
<xsl:message select="$call"/>
```

Im nächsten Schritt erfolgt eine Transformation von HTML in ein ParsX-Dokument. Aus dem Ordner *„xsl“* wird dann die XSLT-Datei *„htmltoparsx.xsl“* aufgerufen.

```
<xsl:variable name="v"
        select="
        transform(map {
        'stylesheet-location': 'htmltoparsx.xsl',
        'source-node': $doc
        })"/>
<xsl:copy-of select ="map:get($v,'output')"/>
```

Das Resultat dieser Transformation wird dann genau an der Stelle eingesetzt, wo vorher in einer Map <import_kapitel> stand.

Die folgende Abbildung zeigt das Ergebnis der Transformation: eine vollständige ParsX-Datei, die vom ParsX-Schema erfolgreich validiert wurde.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="https://tools.parsx.de/xsd/v4.0/validate/common/parsx.sch" type="application/xml" schematypens="htt
<projekt xmlns="https://www.parsx.de/ns/4.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="urn:parsx xsd/parsx.xsd"
         version="4.0">
   <meta> [4 lines]
   <werk>
      <vorspann> [686 lines]
      <hauptteil>
         <kapitel>
            <ul>
               <titel>
                  <text>Markdown Syntax</text>
               </titel>
            </ul>
            <bild>
               <abbildung quelle="img/cover.jpg" ersatztext="Cover"/>
               <legende>
                  <abs>Cover</abs>
               </legende>
            </bild>
            <bild>
               <abbildung quelle="img/first.jpg"
                          ersatztext="Abbildung 1: Markdown verwendet einfache Zeichen zur Formatierung von Text."/>
               <legende>
                  <abs>
                     <fett>Abbildung 1:</fett> Markdown verwendet einfache Zeichen zur Formatierung von Text.</abs>
               </legende>
            </bild>
            <abs>Kommt trotzdem jedes Mal die gleiche Liste heraus. Wenn gewünscht, kann man seine Listen von Hand ri
            <liste_ungeordnet>
               <eintrag>
```

Abbildung 8: Komplettes ParsX-Dokument

Fazit

In der vorliegenden Arbeit ging es um die Beantwortung folgender Fragen:

- Inwieweit kann der LwDITA-Ansatz auf andere Publikationsstandards aus der Verlagsbranche übertragen werden?
- Welche Voraussetzungen und Bedingungen müssen vorliegen, damit der LwDITA-Ansatz bei anderen XML-Standards einsetzbar ist?

Für die Beantwortung der Fragen wurden LwDITA analysiert und die technischen Merkmale erarbeitet. Anhand definierter Merkmale wurde der LwDITA-Ansatz auf den ParsX-Standard übertragen.

Zuerst wurde ein Kapitel als Inhaltsobjekt definiert, in dem der eigentliche Inhalt strukturiert wird. Bei der Umsetzung wurde das Kapitel im Markdown-Format angelegt und in ParsX-Map referenziert. Das Hauptaugenmerk lag dabei auf der Kompatibilität zwischen den beiden Formaten. Nach der Transformation stellte sich heraus, dass das Markdown-Dokument vollständig in ein ParsX-Dokument konvertiert werden konnte.

Der nächste wichtige Punkt war die Erstellung einer ParsX-Map. Um den Verweis zu den Kapiteln herzustellen, sollten hierbei zwei wichtige Komponenten, das Element <import_kapitel> mit dem Attribut @system-ressource, verwendet werden. Eine Voraussetzung, die nicht ParsX erfüllen konnte, war die Identifizierung der Formate von referenzierten Kapiteln. Daher wurde dem ParsX-Schema das @format-Attribut hinzugefügt.

Zum Schluss erfolgte eine Transformation der ParsX-Map. Die Transformation lief in verschiedenen Vorgängen ab, aus denen ein vollständiges ParsX-Dokument erzeugt werden konnte. Letztendlich wurden alle referenzierten Kapitel in ein ParsX-Dokument eingesetzt und das ganze Dokument erfolgreich validiert.

Zusammenfassend lässt sich sagen, dass alle aus dem LwDITA definierte Merkmale erfolgreich auf den ParsX-Standard übertragen wurden. Die Kombination verschiedener Auszeichnungssprachen funktionierte auch beim ParsX-Standard einwandfrei. Die Markupsprachen beschreiben die Struktur des Dokuments; deswegen ist es letztendlich nicht wichtig, in welchem Format der Inhalt vorhanden ist, sondern wie der Inhalt in dem jeweiligen Format strukturiert ist. Eine einheitliche Struktur schafft eine einwandfreie Konvertierung zwischen verschiedenen Auszeichnungssprachen.

Literaturverzeichnis

Agular, Robert R: HTML und CSS: Praxisrezepte für Einsteiger.
Heidelberg: Redline GmbH 2008

Ali, Khadija Abied: "XML Technology." In: International Journal of Electrical & Computer Sciences IJECS-IJENS, Nr. 03 vom Juni 2014. S.12

Anderson, Robert D./ Eberlein, Kristen James: Darwin Information Typing Architecture (DITA) Version 1.3 Part 3: All-Inclusive Edition. 2018.
URL: http://docs.oasis-open.org/dita/dita/v1.3/dita-v1.3-part3-all-inclusive.pdf [Zugriff: 20.08.2021]

Anderson, Robert D/ Elovirta, Jarno/ Sheen, Roger W. Fienhold: „DITA Open Toolkit.
The open-source publishing engine for content authored in the Darwin Information Typing Architecture.“ 2019. URL: https://www.dita-ot.org/ [Zugriff: 12.09.2021]

Babenko, Vitalina/Yatsenko, Roman/Migunov, Pavel/Salem, Abdel-Badeeh: MarkHub Cloud Online Editor as a modern web-based book creation tool. Ukraine: National University 2020. URL: http://ceur-ws.org/Vol-2643/paper09.pdf [Zugriff 15.07.2021]

Becher, Margit. XML: DTD, XML-Schema, XPath, XQuery, XSLT, XSL-FO, SAX, DOM. Dortmund: W3l GmbH, 2009.

Bongers,Frank: XSLT 2.0 & XPath 2.0. Deutschland: Galileo Press 2008

Borgstrom, Liam: „Publishing In The Digital Environment“. 2013.
URL: https://pub310.pressbooks.com/chapter/4-1-name-title-subtitle/
[Zugriff 19.07.2021]

Bray, Team/ Paoli, Jean/Sperberg-McQueen, C. M./Maler, Eve: Extensible Markup Language (XML) 1.0 (fifth Edition). W3C Recommendation 26 November 2008.
URL: https://www.w3.org/TR/xml/ [Zugriff 25.07.2021]

Bray, Team/ Paoli, Jean/Sperberg-McQueen, C. M./Maler, Eve: Extensible Markup Language (XML) 1.0 (fifth Edition). W3C Recommendation 26 November 2008.
URL: https://www.w3.org/TR/xml/ [Zugriff 25.07.2021]

Buck, Lee/Goldfarb, Charles F./ Prescod, Paul: Datatypes for DTDs (DT4DTD) 1.0.
W3C Recommendation 2000.
URL: https://www.w3.org/TR/dt4dtd/ [Zugriff 26.07.2021]

Bühler, Peter/Böhringer, Joachim/ Schlaich, Patrick: HTML5 und CSS3: Semantik - Design - Responsive Layouts. Berlin: Springer-Verlag 2017.

Closs, Sissi. DITA–der topic-basierte XML-Standard: Ein schneller Einstieg. Springer-Verlag 2015

Closs, Sissi: Single Source Publishing. Modularer Content für EPUB & Co. Frankfurt: Software und Support Media GmbH 2011

Cone, Matt. Markdown Guide. Independently Published, 2020. URL: https://lesen.amazon.de/?asin=B07G7JB641 [Zugriff 18.07.2021]

Cosmin, Duna Marius: „Oxygen Batch Documents Converter add-on for Oxygen XML Editor." 2021. URL: https://github.com/oxygenxml/oxygen-resources-converter [Zugriff: 12.09.2021]

Day, Don/Priestley, Michael/Schell, David: Introduction to the Darwin Information Typing Architecture. IBM corporation 2005. URL: https://people.cs.vt.edu/~kafura/CS6604/Papers/Darwin-Information-Typing-Architecture.pdf [Zugriff 12.07.2021]

Dournaee, Blake: XML security. New York: Mcgraw-Hill, 2002.

Dünckel, Björn/Schmull, Heino: parsX 4 XSD. Dokumentation. 2021. URL: https://tools.parsx.de/xsd/v4.0/doku/ [Zugriff: 30.07.2021]

Eberlein Khristen/ Harrison Nancy: „OASIS Darwin Information Typing Architecture (DITA) Technical Committee. o.J. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita [Zugriff 12.07.20201]

Elovirta, Jarno: „Lightweight DITA for DITA-OT." 2017. URL: https://github.com/jelovirt/dita-ot-markdown [Zugriff: 12.09.2021]

Engels, Christian: „Bringen Sie mit Information Mapping Struktur in Ihre Doku!". o.J. URL: https://www.docuneers.de/bringen-sie-mit-information-mapping-struktur-in-ihre-doku/ [Zugriff 09.07.2021]

Esfahani, Mehrschad Zaeri: „XML-Technologien: Unicode." O.J. URL: https://www.parsqube.de/xml-technologien/xml/xml-einfuehrung/unicode/ [Zugriff 25.07.2021]

Evia, Carlos/Houser, Alan: Lightweight DITA Version 1.0. Working Draft 01. OASIS Darwin Information Typing Architecture (DITA). URL: https://www.oasis-open.org/committees/download.php/65658/lwdita.pdf. [Zugriff 19.07.2021]

Evia, Carlos: Creating Intelligent Content with Lightweight DITA. New York: Reutledge Verl. 2018.

Fallside, David C./ Walmsley, Priscilla: XML-Schema Part 0: Primer (Second Edition). W3C Recommendation 28 October 2004.
URL: http://www.w3.org/TR/xmlschema-0/ [Zugriff: 28.07.2021]

Fawcett, Joe/Danny Ayers/Liam RE Quin: Beginning XML. 5., akt. Aufl.
New Jersey: John Wiley & Sons Verl. 2012

Finke, Katrin: Terminologiemanagement - Information Mapping und Funktionsdesign. Seminararbeit, Johannes-Gutenberg-Universität Mainz 2009

Fuchs, Paul: HTML5 und CSS3 für Einsteiger: der leichte Weg zur eigenen Webseite.
Deutschland: BMU Media GmbH 2019

Gruber, John/Swartz, Aaron: “Markdown,” 2004.
URL: http://daringfireball.net/projects/markdown [Zugriff 19.07.2021]

Harold, Elliotte Rusty / W. Scott Means: XML in a Nutshell. 2., akt. Aufl.
Newton: O'Reilly Media 2004

Harold, Elliotte Rusty. Processing XML with Java: a guide to SAX, DOM, JDOM, JAXP, and TrAX.
Boston: Addison-Wesley Professional 2003.

Harold, Elliotte Rusty: XML 1.1 Bible. 3., akt. Aufl. New Jersey: John Wiley & Sons Verl. 2004.

Hentrich, Johannes: DITA: der neue Standard für technische Dokumentation.
München: XLcontent Verlag 2008.

Herrero, Arturo: Instant Markdown. Birmingham: Packt Publishing Ltd, 2013.
URL: https://lesen.amazon.de/?asin=B00ESX181A [Zugriff 19.07.2021]

Hickson, Anthony: HTML Living Standard. Last Updated 20 September 2021.
URL: https://html.spec.whatwg.org/ [Zugriff: 28.07.2021]

Horn, Robert: „Information Mapping: New Tool to Overcome the Paper Mountain“. In: Educational Technology. Vol. 14, No. 5 (May, 1974), pp. 5-8 (4 pages).

IBM Corporation: „XML schema definition (XSD) assets“. o.J.
URL: https://www.ibm.com/docs/en/iis/
11.5?topic=types-xml-schema-definition-xsd-assets [Zugriff 27.07.2021]

Jitianu, Alex: DITA and Markdown. Syncro Soft SRL: 2019.
URL: https://www.oxygenxml.com/events/2019/Tekom_2019/dita_md.pdf
[Zugriff: 12.09.2021]

Jung, Benjamin: „Komplexe Typen“.O.J.
URL: https://www.homepage-webhilfe.de/XML/XSD/komplexetypen.php
[Zugriff: 28.07.2021]

Kay, Michael: XSL Transformations (XSLT) Version 3.0. W3C Recommendation 8 June 2017.
URL: https://www.w3.org/TR/xslt-30/

Kornelsen, L./Lucke, U./, Tavangarian, D./Waldhauer, M./Ossipova, N.: Strategien und Werkzeuge zur Erstellung multimedialer Lehr- und Lernmaterialien auf Basis von XML.
In: Engels, G. & Seehusen, S. (Hrsg.), DeLFI: Die 2. e-Learning Fachtagung Informatik, Tagung der Fachgruppe e-Learning der Gesellschaft für Informatik e.V. Bonn: Gesellschaft für Informatik e.V. 2004

Krüger, Sandra/Balzert, Helmut/Beims, Hans-Dieter: HTML5, XHTML & CSS: Websites systematisch & barrierefrei entwickeln. 2., akt. Aufl. Deutschland: W3L-Verlag 2011

Leonard, Sean: Guidance on markdown: Design philosophies, stability strategies, and select registrations. RFC, IETF, 2016.
URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc7764.txt.pdf [Zugriff 17.07.2020]

MacFarlane, John: „Pandoc-Benutzerhandbuch“. o.J.
URL: https://pandoc.org/MANUAL.html#authors [Zugriff: 16.07.2021]

Mailund, Thomas: Introducing Markdown and Pandoc: Using Markup Language and Document Converter. Deutschland: Apress 2019.

Mason, Philip: SAS Stored Processes: A Practical Guide to Developing Web Applications. Deutschland: Apress 2020.

Mertens, Peter: XML-Komponenten in der Praxis. Springer-Verlag, 2013.

Musciano, Chuck/Bill Kennedy: HTML & XHTML: The Definitive Guide. 3., akt. Aufl. Sebastopol: O'Reilly Media 2002.

Ott, Tobias/Fischer, Tobias: Whitepaper parsX. Konzept und Module. pagina GmbH Publikationstechnologien 2018. URL: https://www.pagina.gmbh/fileadmin/user_upload/downloads/Whitepaper-parsX_Februar2018.pdf [Zugriff 19.07.2021]

Ott, Tobias: „Entities in XSD“. O.J.
URL: https://www.pagina.gmbh/xml-hintergruende/pagina-das-kompendium/themenkomplex-ii-xml/xml-schema-als-dtd-alternative/entities-in-xsd/?L=o%27%22#cite_ref-1 [Zugriff: 28.07.2021]

Ott, Tobias: „parsX. Das XML-Framework für kleine und mittelständische Buchverlage“. o.J.
URL: https://www.pagina.gmbh/produkte/parsx/ [Zugriff: 30.07.2021]

Ott, Tobias: Crossmediales Publizieren im Verlag. Berlin: Walter de Gruyter Verl. 2014

Ott, Tobias: parsX wird offener Herstellerstandard. 2019.
URL: https://www.parsx.de/wp-content/uploads/2019/09/Interview-T.Ott_.pdf [Zugriff: 30.07.2021]

Ovadia, Steven: Markdown for librarians and academics. Behavioral & social sciences librarian. London: Taylor & Francis Group 2014
URL: https://academicworks.cuny.edu/cgi/viewcontent.cgi?article=1006&context=lg_pubs [Zugriff 15.07.2021]

Parker, Kesi: „Topic-Based Authoring“. 05.03.2021. URL: https://medium.com/technical-writing-is-easy/topic-based-authoring-6430df08cc17 [Zugriff 19.07.2021]

Pellet, Jean-Philippe, Gabriel Parriaux, and Nora Anna Escherle: A Markdown-based Open Toolchain for Writing, Processing, and Publishing Bebras Tasks.
Lausanne: University of Teacher Education 2020.
URL: https://orfee.hepl.ch/bitstream/handle/20.500.12162/4464/issep2020-poster-description.pdf?sequence=4&isAllowed=y [Zugriff 15.07.2021]

Python Software Foundation: „Introduction to Unicode Definitions.“
URL: https://docs.python.org/3/howto/unicode.html [Zugriff 25.07.2021]

Rantakari, Ali: Adapting a Markdown Compiler’s Parser for Syntax Highlighting. 2011.
URL: https://hasseg.org/peg-markdown-highlight/peg-markdown-highlight.pdf [Zugriff 14.07.2021]

Roberts, Tom: „Modularizing Your Content for Simplicity and Productivity“. 11.05.2015.
URL: https://blog.doculabs.com/modularizing-your-content-for-simplicity-and-productivity [Zugriff 09.07.2021]

Schmull, Heino/Dünckel, Björn: Tutorial. O.J.
URL: https://tools.parsx.de/xsd/v4.0/#/tutorial [Zugriff: 30.07.2021]

Sean Leonard: Anleitung zum Markdown: Designphilosophien, Stabilitätsstrategien und ausgewählte Registrierungen. 2016.
URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc7764.txt.pdf. [Zugriff 20.07.2021]

Sheen, Roger: „DITA Open Toolkit 3.6 Documenation“. o.J. URL: https://www.dita-ot.org/dev/ [Zugriff 19.07.2021]

Standard, O. A. S. I. S.: Darwin Information Typing Architecture (DITA) Version 1.2. 2010.
URL: http://docs.oasis-open.org/dita/v1.2/spec/DITA1.2-spec.pdf [Zugriff 12.07.2021]

Syncro Soft: „Oxygen XML Editor User Manual. Working with Markdown Documents in DITA.“ o.J. URL: https://www.oxygenxml.com/doc/versions/23.1/ug-editor/topics/markdown-dita.html [Zugriff 19.07.2021]

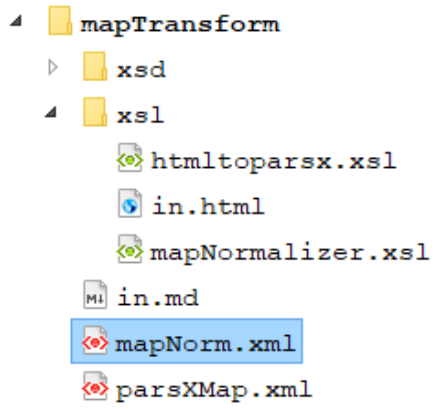
Van der Vlist, Eric: XML schema. Newton: O'Reilly Media 2002

Wiemer, Philip: „Single-Source-Publishing – eine Datenquelle für alle Ausgabekanäle“. 2020.
URL: https://www.mds.eu/blog/single-source-publishing-eine-datenquelle-fuer-alle-ausgabekanaele. [Zugriff 05.07.2021]

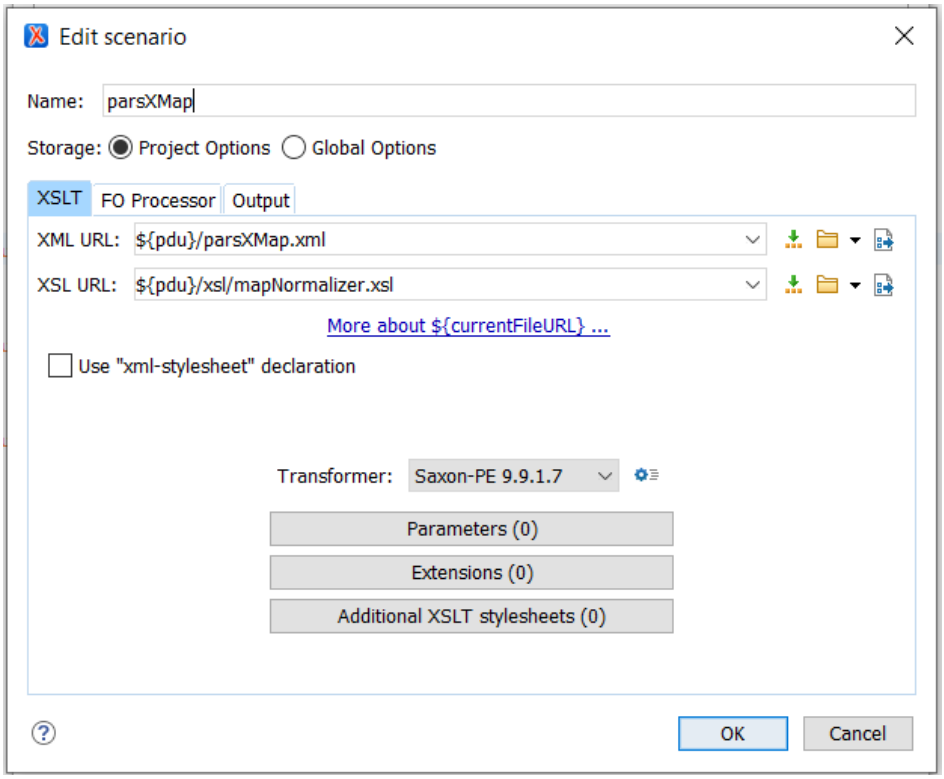
Abbildungsverzeichnis

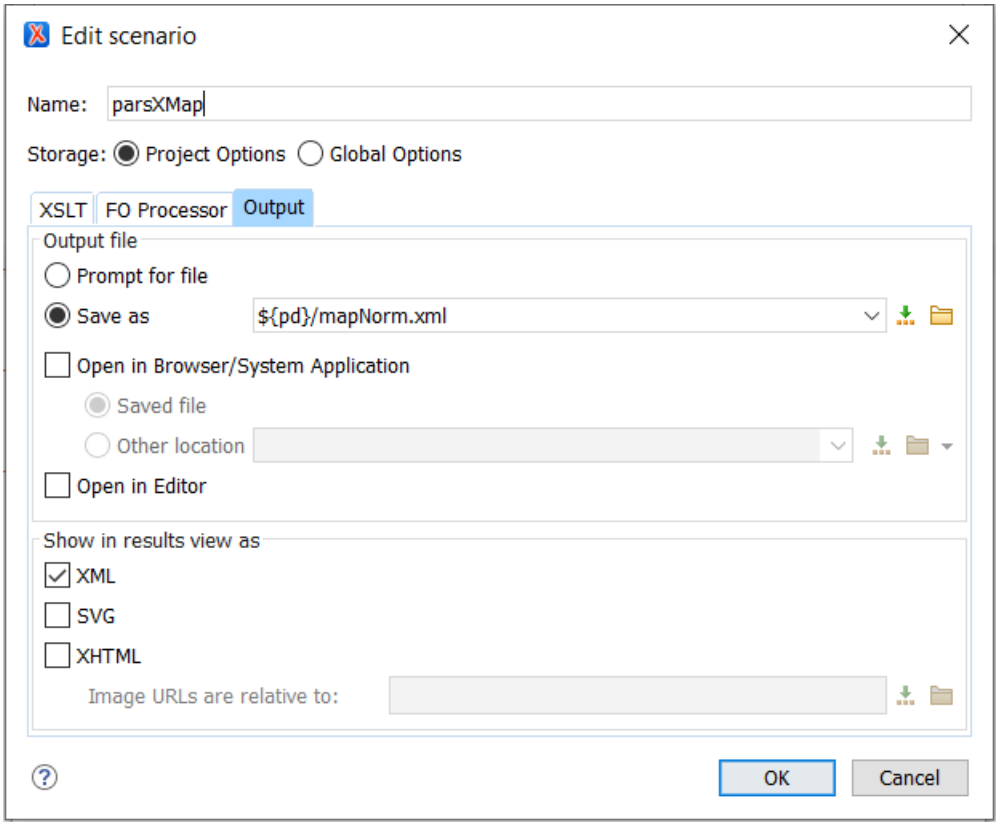
Anhang

Dieser Anhang ist in elektronischer Form im Verzeichnis „mapTransform“ auf CD zu finden.



Transformationsszenario erstellen




Edit scenario
Name: parsXMap
Storage: Project Options Global Options
XSLT FO Processor Output
Output file
Prompt for file
Save as ${pd}/mapNorm.xml
Open in Browser/System Application
Saved file
Other location
Open in Editor
Show in results view as
XML
SVG
XHTML
Image URLs are relative to:
OK
Cancel

Ehrenwörtliche Erklärung

Hiermit versichere ich, Teona Burnadze, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Markup-Mix beim Single Source Publishing. Abbildung des Lightweight-Ansatzes von DITA auf den Branchenstandard ParsX“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.“

Ort, Datum	Teona Burnadze