Bachelorarbeit

Konrad Glugla

Realisierung eines XML-basierten Desktop-Publishing-Systems zur automatischen Erzeugung des Papierlayouts am Beispiel von Flugbetriebsdokumenten der Lufthansa AG

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Konrad Glugla

Realisierung eines XML-basierten Desktop-Publishing-Systems zur automatischen Erzeugung des Papierlayouts am Beispiel von Flugbetriebsdokumenten der Lufthansa AG

Bachelorarbeitarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Michael Neitzke
Zweitgutachter : Prof. Dr. rer. nat. Bettina Buth

Abgegeben am 27. August 2007

Konrad Glugla

Thema der Bachelorarbeit

Realisierung eines XML-basierten Desktop-Publishing-Systems zur automatischen Erzeugung des Papierlayouts am Beispiel von Flugbetriebsdokumenten der Lufthansa AG

Stichworte

XML, Desktop-Publishing, XSLT, XSL-FO, Papierlayout, PDF

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der automatisierten Erzeugung des Papierlayouts aus XML-Daten. In diesem Zusammenhang wird der Einsatz von Desktop-Publishing (DTP) Lösungen, welche diesen Prozess unterstützen, für die Verarbeitung von XML-Daten in das Ausgabeformat PDF analysiert.

Die ausführliche Analyse zeigt auf, dass sich XSL-FO für die automatisierte Verarbeitung von Fließtextdokumenten am besten eignet.

Auf Basis von XSL-FO wird ein DTP-System entwickelt, das am Beispiel von Flugbetriebsdokumenten der Lufthansa AG den Prozess der Layout-Erzeugung weitestgehend automatisiert. Bei dem entwickelten System wird auf eine Übertragbarkeit und Erweiterbarkeit der Lösung geachtet, um zukünftigen Anforderungen gerecht zu werden.

Konrad Glugla

Title of the paper

Realization of an XML based desktop publishing system for the automatic generation of a paper layout by example of flight operation documents of Lufthansa AG

Keywords

XML, Desktop-publishing, XSLT, XSL-FO, paper layout, PDF

Abstract

By means of this thesis the automatic manufacturing and production of documents (based on XML Data), shown into a paper layout, will be investigated. In this context the use of a desktop publishing (DTP) solution is analyzed for the processing of XML data into the output format PDF.

The detailed analysis shows that XSL FO is the most suitable technology for the automated processing of flow text documents.

Based on XSL-FO a DTP system is developed, which automates the process of the layout production using flight operation documents, owned by Lufthansa AG, as prototype and example. Special considerations will be placed on expandability, transferability, and the potential for future upgrades to the system.

Inhaltsverzeichnis

Glossar

CALS CALS Tabellenmodell. Spezifikation unter: http://www.oasis-open.org/specs/a502.htm

DTD Document Type Definition, definiert die Grammatik einer SGML-/XML-Datei

DTP Desktop-Publishing

FAQ Frequently Asked Questions. Literatur in denen häufig gestellte Fragen, z.B. zu Produkten, beantwortet werden.

FO Abkürzung für XSL-FO

FSM Flight Safety Manual, Dokument der Lufthansa AG

LH Lufthansa AG

LSY Lufthansa Systems AG

OM-A Operations Manual - Part A, Dokument der Lufthansa AG

PDF Portable Document Format, Ausgabeformat der Firma Adobe®Systems GmbH

TTF TrueType Font

W3C World Wide Web Consortium (http://www.w3.org)

WYSIWYG What You See Is What You Get

XML Extensible Markup Language, Sprache zur Beschreibung strukturierter Dokumente

XPath XML Path Language. XPath dient zur Adressierung von Teilen eines XML-Dokuments.

XSL-FO XSL Formatting Ojects. XSL-FO ist ein XML-Vokabular zur Beschreibung von Formatieranweisungen.

XSLT XSL Transformations. XSLT dient zur Transformation von XML-Dokumenten in ein anderes Format.

1. Einleitung

An der Erstellung von umfangreichen Dokumentationen in Unternehmen sind häufig mehrere Personen beteiligt. Mit den herkömmlichen Methoden zur Erstellung von Dokumenten über Textverarbeitungsprogrammen wie Word und Co. verlieren Autoren und Übersetzer meist wertvolle Zeit mit Formatierarbeiten. Vor allem die Industrie fordert effizientere Verfahren zur Herstellung von Dokumentationen.
Diese Anforderungen führten zur Entwicklung sogenannter Desktop-Publishing-Systeme, im Folgenden mit DTP-Systeme abgekürzt, welche die Erstellung von umfangreichen Dokumenten stark automatisiert. Das Ziel eines solchen Systems ist es für Dokumente, die einer bestimmten Logik folgen, ein Layout zu generieren. Dadurch wird der Autor bei der Formatierung unterstützt und Zeit eingespart.
Es gibt zum Einen die WYSIWYG (What You See Is What You Get)-basierten DTP-Systeme, bei denen der Anwender das Layout mittels eines Tools direkt sieht und manipulieren kann. Zum Anderen gibt es Satzsysteme wie LaTeX, bei denen der Textinhalt mit Steuerzeichen beschrieben wird und somit kein bestimmter Editor notwendig ist.

In einem großen Unternehmen, wie die Lufthansa AG, in dem umfangreiche technische Dokumentationen eine wichtige Rolle spielen, ist es aus technischen und wirtschaftlichen Gründen notwendig einen Workflow zur Erstellung und Verwaltung der Dokumente zu entwickeln. Am Ende dieses Workflows steht das Publizieren der Dokumentation, z.B. für den Druck.

Eine Technologie, die häufig für solch ein Dokumentenmanagement eingesetzt wird, ist „XML“ (siehe Kapitel 2.3). XML ist eine Auszeichnungssprache, die Daten strukturiert aufbaut und es ermöglicht eine saubere Trennung von Inhalt und Layout zu erreichen.
In „The XML Companion“ von Neil Bradley heisst es:
„The Extensible Markup Language is a powerful publishing and document interchange format...
... information held in this format is self-describing - it can be extracted, manipulated and formatted to the requirements of any target audience or publishing medium.“[Bra00a, Preface]
Nach Neil Bradley ist XML eine Sprache, die für den Menschen leicht verständlich ist und sich für ein Dokumentenmanagement sehr gut eignet. Außerdem lassen sich XML-Daten aufgrund des logisch strukturierten Aufbaus sehr gut als Grundlage für den Einsatz im Desktop-Publishing nutzen.

Viele Unternehmen behelfen sich heutzutage der Mächtigkeit von XML. So enstand auch bei der Lufthansa Systems AG, im Folgenden LSY, unter dem Projektnamen ’EFOM’ (’Electronic Flight Operations Manual) ein Dokumentenmanagementsystem auf der Grundlage von von SGML, dem Vorgänger von XML. In ’EFOM’ wurde zur Erzeugung des Papierlayouts das Produkt Adobe® FrameMaker® eingesetzt. Im Zuge des Projektes „EFOM-2“ soll der Wechsel von SGML zu XML erfolgen. Für das Desktop-Publishing soll untersucht werden, ob ein anderer Prozess die Effizienz und den Automatisierungsgrad bei der Erstellung der Dokumente ins Papierlayout erhöhen kann.

1.1. Ziele

XML-Dokumente enthalten nur die Struktur und den Inhalt einer Dokumentation. Sie enthalten keinerlei Informationen zum Layout. Zur Publikation der XML-Dokumente muss eine weitere Verarbeitung stattfinden. Die Lufthansa AG, im Folgenden mit 'LH' abgekürzt, hat bestimmte Anforderungen an das Layout verschiedener Flugbetriebsdokumente. Diese Layout-Anforderungen sollen durch Verarbeitung der XML-Quellen eingehalten und umgesetzt werden.
Ziel dieser Arbeit ist es, für die 2 Flugbetriebsdokumente „Operations Manual - Part A", im Folgenden mit „OM-A" abgekürzt, und „Flight Safety Manual", im Folgenden mit „FSM" abgekürzt, eine Papierpublikation auf Grundlage der XML-Quellen zu erstellen. Das Zieldokument ist für den Druck in DIN-A5 vorgesehen und soll im Ausgabeformat PDF vorliegen.
Um die Papierpublikation zu realisieren, muss zunächst die Struktur der XML-Dokumente mit Hilfe der DTD (siehe Kapitel 2.3.1) analysiert werden. Außerdem muss geklärt werden, welche Sprache und/oder welche Tools eingesetzt werden, um das Zielformat zu erreichen.
Zur Wiederverwertbarkeit wird ein System entwickelt, das dynamisch anpassbar und erweiterbar ist.

1.2. Aufbau dieser Arbeit

Zu Beginn dieser Arbeit werden in Kapitel 2 die Grundlagen geklärt. Die Grundlagen befassen sich hauptsächlich mit der XML-Technologie und deren Verarbeitungsmöglichkeiten für ein Desktop-Publishing.
In Kapitel 3 wird noch ein kurzer Überblick über das Projekt 'EFOM' der LSY und den derzeit dort verwendeten Techniken für das Desktop-Publishing gegeben.
Im anschließenden Analyseteil werden zunächst die Anforderungen gesammelt, die von den umzusetzenden Dokumenten gestellt werden. Unter der Betrachtung dieser Anforderungen werden die verschiedenen Techniken, die heutzutage angewendet werden um hochwertige Dokumente zu erstellen, analysiert. Insbesondere wird hierbei auf die Möglichkeiten der Verwendung mit XML-Daten eingegangen, die LSY für ihre Dokumentationen bereitstellt.
Das Ergebnis der Analyse ist die Auswahl einer Technik aus den heutigen Möglichkeiten des Desktop-Publishings, welche die Anforderungen am besten erfüllt und zusätzlich eine gute Möglichkeit zur Erweiterbarkeit, in Hinsicht auf zukünftige Projekte, bietet.
Mit der ausgewählten DTP-Lösung wird in Kapitel 5 ein Entwurf des umzusetzenden DTP-Systems entwickelt. Anhand dieses Entwurfs wird im nächsten Kapitel die durchgeführte Realisierung beschrieben.
Abschließend wird eine Zusammenfassung der Arbeit und ein Ausblick für den zukünftigen Einsatz des entwickelten DTP-Systems gegeben.

2. Grundlagen

2.1. Desktop-Publishing

Desktop-Publishing, im Folgenden mit DTP abgekürzt, ist der Oberbegriff für das rechnergestützte Setzen hochwertiger Dokumente, die aus Texten und Bildern bestehen und später als Publikation, wie z.B. Broschüren, Magazine, Bücher oder Kataloge, ihre Verwendung finden. Software, die für das DTP entwickelt wird, soll dabei den Anwender bei der Erstellung der Dokumente, z.B. durch Layoutvorgaben oder automatischer Erstellung von Inhaltsverzeichnissen, unterstützen.
Eingeführt wurde das heute bekannte DTP 1985 von den Firmen Apple®, Adobe®, Aldus und Linotype. Dabei steuerte Adobe® die Seitenbeschreibungssprache PostScript®, Aldus das erste Layout-Programm (PageMaker®), Apple® den ersten voll grafikorientierten Rechner (Macintosh®) und einen PostScript®-fähigen Laserdrucker (LaserWriter) bei. Linotype lieferte die ersten PostScript®-Schriften und den ersten PostScript®-fähigen Belichter.
Während das DTP in seinen Anfangsjahren von vielen als Spielerei abgetan wurde, entwickelte sie durch die mittlerweile erreichte Professionalität der Soft- und Hardware auch in der hochprofessionellen Druck- und Satzindustrie eine breite Akzeptanz.
Seit etwa 1992 werden Printprodukte fast ausschließlich im Rahmen von DTP produziert. Dabei sind vor allem die WYSIWYG-Produkte sehr beliebt. WYSIWYG bezeichnet die Arbeit mit Tools, bei denen während der Bearbeitung am Bildschirm bereits das Layout der Ausgabe sichtbar ist (wie z.B. bei Microsoft® Word).

[WIK07, Desktop Publishing] und [Krü97].

2.2. Portable Document Format (PDF)

Das Portable Document Format ist ein plattformübergreifendes Dateiformat für Dokumente, das von der Firma Adobe® Systems entwickelt und 1993 mit Acrobat® 1 veröffentlicht wurde [WIK07, PDF].
Obwohl gesagt, wird dass das Portable Document Format teilweise PostScript® verdrängt, ist es nicht als Ersatz oder als Nachfolger von PostScript® zu verstehen. PDF ist vielmehr eine andere Art, die Inhalte von Dokumenten zu beschreiben. Jede der beiden Technologien hat ihre Daseinsberechtigung und einen Einsatzbereich, in dem sie der jeweils anderen überlegen ist [SB00].
PDF-Dateien dienen zur Darstellung von Text und Bild und werden mittlerweile sehr häufig für den Austausch von kleineren oder auch größeren Dokumenten (häufig technische Dokumentationen) eingesetzt. Dabei steht im Vordergrund das reine Lesen oder Ausdrucken dieser Dokumente, da die Bearbeitung von PDF-Dokumenten sehr eingeschränkt ist. Durch die Anwendung von Komprimiermethoden und den Verzicht gewisser Dateibeschreibungen verbrauchen PDF-Dateien meist weniger Speicherplatz als andere Formate (z.B. PostScript®). Ein großer Vorteil von PDF ist zudem, dass die gestalteten Seiten im PDF-Format auf allen Systemen (Windows®, Macintosh®, Unix®, etc.) ein identisches Aussehen haben. Zur Anzeige von PDF-Dateien wird allerdings ein geeignetes Leseprogramm vorausgesetzt, das von Adobe kostenlos zur Verfügung gestellt wird (Adobe® Reader®).

2.3. SGML/XML

SGML und XML sind Auszeichnungssprachen zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien. Sie dienen zur Trennung von Inhalt und Layout eines Dokuments. Mit diesen Konzepten sollen Daten, die in Form von SGML oder XML vorliegen, für verschiedene Anwendungszwecke weiterverarbeitet werden können.
SGML wurde bereits im Jahre 1986 als internationaler Standard eingeführt, erreichte jedoch aufgrund seiner Komplexität nie eine breite Akzeptanz. Aus diesem Grunde wurde später das vereinfachte XML eingeführt, von dem gesagt wird, es habe 80% der Mächtigkeit von SGML, aber nur 20% der Komplexität [WHB01, Seite 18].
XML ist ein Standard des World Wide Web Consortiums, im folgenden mit W3C abgekürzt, und wird in der Literatur wie folgt definiert:
„XML ist eine textbasiete Meta-Auszeichnungssprache, die die Beschreibung, den Austausch, die Darstellung und die Manipulation von strukturierten Daten erlaubt, so dass diese von einer Vielzahl von Anwendungen genutzt werden können“ [WHB01, Seite 18ff].
XML hat in den letzten Jahren sehr an Beliebtheit gewonnen, da sie Informationen beschreibt die für Mensch und Maschine verständlich sind und da mittlerweile viele Parser vorhanden sind.

Informationen in XML werden als Baumstruktur dargestellt und bestehen grundsätzlich aus einem passenden Paar von Start-Tag (*<Tag-Name>*) und End-Tag (*</Tag-Name>*). Weitere Details zum Aufbau von XML-Dokumenten sind z.B. aus [Bra00a, Kapitel 3] zu entnehmen.

Um die Struktur von XML-Dokumenten zu beschreiben und einzuschränken werden sogenannte Schemasprachen eingesetzt (z.B. DTD). Um XML-Dokumente weiterzuverarbeiten und in ein entsprechendes Layout zu bringen, kann die Transformationssprache XSLT eingesetzt werden. Beide Sprachen werden in den nächsten Abschnitten erklärt.

2.3.1. DTD

Die DTD (Document Type Definition) ist eine Schemasprache zur Definition einer Auszeichnungssprache. Mit ihr kann zu XML-Dokumenten eine explizite Definition der nötigen und möglichen Tags und ihrer Struktur gegeben werden.
Eine DTD enthält also die erforderlichen bzw. erlaubten Tags einer Auszeichnungssprache und deren Schachtelung. Die Referenz auf seine DTD erfolgt im XML-Dokument.
Alle Elemente die im XML-Dokument vorkommen dürfen, müssen in der DTD deklariert werden. Der Aufbau einer Elementdeklaration sieht in etwa wie folgt aus:

```
<!ELEMENT AUFTRAGSKOPF (NAME, DATUM, E-MAIL)>
```

Das Element AUFTRAGSKOPF wurde deklariert. Es enthält die Unterelemente NAME, DATUM und E-MAIL, die in dieser Reihenfolge auftauchen müssen.
Die Schemasprache DTD ermöglicht eine grobe Beschreibung für den Aufbau eines XML-Dokuments. Weitere Details sind der Literatur zu entnehmen.

[WHB01, Seite 25ff] und [Bra00a, Seite 47ff].

2.3.2. XSL

XSL beschreibt einen vom W3C definierten Standard zur Transformation und Präsentation von XML-Daten. XSL besteht aus 3 Teilkonzepten:

- XSL Transformations (XSLT):

  XSLT dient zur Transformation von XML-Dokumenten in ein anderes Format.
- XML Path Language (XPath):

  XPath dient zur Adressierung von Teilen eines XML-Dokuments.
- XSL Formatting Ojects (XSL-FO):

  XSL-FO ist ein XML-Vokabular zur Beschreibung von Formatieranweisungen.

XSL Transformations (XSLT)

Die Sprache XSLT beschreibt die Umwandlung eines XML-Derivats (auch XML-Dialekt oder XML-Anwendung genannt) in ein anderes Dokument. Diesen Vorgang nennt man Transformation. Das resultierende Dokument entspricht meist der XML-Syntax, es können aber auch andere Textdateien und sogar Binärdateien erstellt werden [WIK07, XSLT].
Die Beschreibung der Transformationen geschieht in sogenannten Stylesheets. Dabei arbeitet man mit XSLT-Schablonen, die ein bestimmtes Element treffen und dann ausgeführt werden. Die Schablone

```
<xsl:template match="AUFTRAGSKOPF">
```

wird z.B. ausgeführt wenn ein Element mit dem Namen 'AUFTRAGSKOPF' auftaucht. Innerhalb solch einer Schablone können dann Transformationen beschrieben werden.
Die Transformation eines XML-Dokuments wird mit Hilfe eines XSL-Prozessors vorgenommen. Der Ablauf ist in Abbildung 2.1 zu sehen:

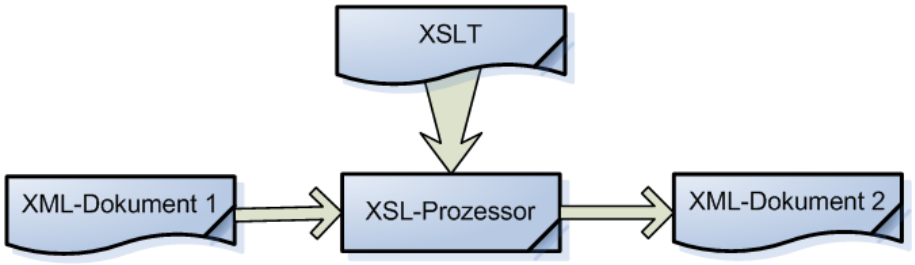


Abbildung 2.1.: Ablauf der XSL Transformation (Quelle: Eigene Darstellung in Anlehnung an [Sch07, Seite 58])

XML Path Language (XPath)

XPath ist eine Sprache, die es erlaubt, Teile von XML-Dokumenten auszuwählen. Das Auswählen der Dokument-Teile geschieht durch einen Prozessor, der einen XPath-Ausdruck „evaluiert“, also auswertet. Das Ergebnis dieser Auswertung besteht typischerweise aus Teilen des Dokuments, auf das sich der Ausdruck bezieht. Die Auswertung kann aber auch zu Zahlen (z. B. der Anzahl von Elementen eines bestimmten Typs), Wahrheitswerten (Booleschen Werten), z. B. dem Vorhandensein von Elementen eines bestimmten Typs in einem Dokument, oder Zeichenketten („strings“), z. B. dem Inhalt eines bestimmten Elements, führen.
Die Syntax von XPath wurde so entworfen, dass es leicht ist, XPath-Ausdrücke als Werte von Attributen z. B. in XSLT-Elementen zu verwenden. Sie orientiert sich an der Syntax von Pfadangaben in URIs.
Es gibt verschiedene Arten von XPath-Ausdrücken. Eine wichtige Rolle für XSLT spielen die sogenannten Lokalisierungspfade oder *location paths* von XPath. Ein Lokalisierungspfad kann aus einer Folge von Lokalisierungsschritten (*location steps*) bestehen. Jeder Schritt in einem Lokalisierungspfad besteht aus drei Bestandteilen (von denen der dritte fehlen kann):

- Angabe einer Richtung oder Achse, in die sich die Suche bewegt (z. B. der Kind-, Nachkommen-, Eltern- oder Attributachse),
- Angabe eines Auswahlkriteriums oder node selectors für die gesuchten Knoten (z. B. des Namens von Elementen oder Attributen),
- beliebig viele (optionale) Prädikate, die die ausgewählten Knoten weiter „filtern“, je nachdem, ob sie auf diese Knoten zutreffen oder nicht.

```
child::p[child::a]
```

Listing 2.1: Beispiel eines Lokalisierungsschrittes (Quelle: [TEI07]

In Listing 2.1 wird nach Kindknoten gesucht, die ein Element des Typs p repräsentieren und mindestens ein Kind mit dem Namen a haben. Zuerst, vor dem doppelten Doppelpunkt “ :: “, wird die Achse angegeben, dann das Auswahlkriterium (der Name p). Zwischen den eckigen Klammern “ [“ und “] “ folgt schließlich das Prädikat (das aus den bereits ausgewählten Knoten diejenigen herausfiltert, die Kindelemente mit dem Namen a haben).

[TEI07, Kapitel 2]

XSL Formatting Ojects (XSL-FO)

XSL-FO wird für Anwendungen eingesetzt, die man gemeinhin als Satz- und Umbruch bzw. als „Seitengestaltung und Umbruch“ bezeichnet.
Das Ausgabe-Medium ist eine Folge von Seiten in festen Maßen (Breite und Höhe). Die Schwerpunkte der XSL-FO Technologie sind dementsprechend Seitenaufteilung (z.B. Kopf-, Fußbereich, Ränder, Mehrspaltigkeit, Platzierung von Inhalten außerhalb des Textflusses u. a.), Zeilenumbruch (mit Silbentrennung), Füllung des Satzspiegels mit vertikalen Keilen (zur Erzielung gleich hoher Seiten), Spalten- bzw. Seitenumbruch (Was muss in einer Spalte/Seite zusammengehalten werden?) und Seitenanordnung/Seitenfolge (Seitenvorlagen, linke/rechte Seiten) [KP04].
Mit Hilfe von XSL-FO ist es möglich, qualitativ hochwertige Druckerzeugnisse entweder auf Papier oder auf dem Bildschirm zu erzeugen. Das Ausgabeformat ist dabei üblicherweise das Portable Document Format (PDF).

Ein Feature von XSL-FO ist die Bereitstellung von vier Richtungsanweisungen. Mit diesem Feature wird der internationale Einsatz von XSL-FO ermöglicht. Die Richungsanweisungen sollen Dokumente im europäischen Sprachraum das Schreiben innerhalb einer Zeile von links nach rechts ermöglichen. Gleichzeitig sollen Zeilen von oben nach unten angeordnet werden. Im arabischen Sprachraum dagegen wird innerhalb einer Zeile von rechts nach links geschrieben, und in der japanischen Schriftsprache schreibt man innerhalb der Zeile von oben nach unten und ordnet die Zeilen von rechts nach links. Die Bestimmung der Richtungsanweisung wird mit dem Attribut *writing-mode* definiert [TEC07, Seite 2].
Der Prozess zur Ausgabe in PDF über XSL-FO geschieht über eine zweistufige Verarbeitung. XML-Dokumente werden über die XSL Transformation in ein XSL-FO Dokument überführt. Ein XSL-FO Dokument wird mit Hilfe eines XSL-FO Renderers (oder auch XSL-FO Prozessor) in das Ausgabeformat gebracht (siehe Abbildung 2.2):

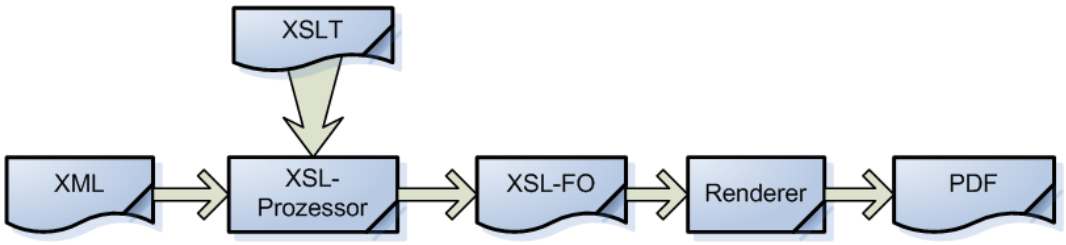


Abbildung 2.2.: Ablauf der Verarbeitung mit XSL-FO (Quelle: Eigene Darstellung in Anlehnung an [Sch07, Seite 77])

Das Konzept von XSL-FO ermöglicht die Entwicklung eines automatisierten Verfahrens zur Erzeugung der Druckausgabe aus XML-Dokumenten.

2.4. Automatisiertes Desktop-Publishing

Mit XSL-FO wurde bereits ein Konzept beschrieben, das ein automatisiertes Verfahren zur Erzeugung der Druckausgabe aus XML-Dokumenten ermöglicht. Des Weiteren gibt es Werkzeuge, die den Ablauf beim Dektop-Publishing stark automatisieren.

2.4.1. LaTeX

LaTeX ist ein Softwarepaket zur Erstellung hochqualitativer Dokumentationen. Es wird oft für mittelgroße bis große technische oder wissenschaftliche Dokumentationen eingesetzt. LaTeX ist kein Textverarbeitungsprogramm, sondern ermöglicht durch den Einsatz von Makros ein automatisches Layout. Dadurch wird der Autor um zusätzliche Formatierarbeiten entlastet. Anders als bei WYSIWYG-Systemen arbeitet der Autor hier mit einfachen Textdateien und zeichnet seinen Text mit einem logischen Markup aus. Formatierungsanweisungen, die vorgenommen werden, sind also erst bei Verarbeitung des Dokuments in ein Ausgabeformat, wie z.B. PDF, sichtbar. Dafür stellt LaTeX eine Menge von Klassen zur Verfügung, in denen für klassische Elemente aus Büchern oder Artikeln ein Layout definiert wird.
LaTeX ist unabhängig von Hardware und Betriebssystemen einsetzbar. Die Ausgabe ist auf allen Rechnerplattformen und verwendeten Druckern identisch, sofern die geeigneten Versionen installiert sind. LaTeX ist auch nicht auf die Schriftarten des jeweiligen Betriebssystems angewiesen, die oftmals für die Anzeige am Bildschirm und nicht für den Druck ausgelegt sind, sondern enthält eine Reihe von eigenen Schriftarten.
Aufgrund seiner Stabilität und der freien Verfügbarkeit für viele Betriebssysteme wird LaTeX vor allem im Bereich von Universitäten und Fachhochschulen eingesetzt.
Seit 2000 existiert für LaTeX ein Paket (’*xmltex*’), das einen XML Parser implementiert und somit in LaTeX die Verarbeitung von XML-Daten ermöglicht.
Auszüge aus [WIK07, LaTeX] und [tea07].

2.4.2. Adobe® FrameMaker®

Adobe® FrameMaker® ist eine professionelle Authoring- und Publishing-Lösung für technische Dokumentationen. Adobe® FrameMaker® ist ein Produkt mit einer eigenen grafikorientierten Oberfläche und arbeitet nach dem WYSIWYG-Prinzip. Das Softwarepaket stellt Funktionen zur Textverarbeitung zur Verfügung und ermöglicht zudem ein stukturiertes, XML-basiertes Arbeiten mit Hilfe von Schablonen. Zur Erstellung von umfangreichen Dokumentationen werden Funktionen wie z.B. automatische Nummerierung, Querverweise, Inhaltsverzeichnisse, Indizes etc., zur Verfügung gestellt. Dadurch wird das Erstellen von umfangreichen Dokumentationen vereinfacht.
Die Eigenschaften von FrameMaker® können durch die Programmierung eigener Plug-Ins mit dem FrameMaker Developer’s Kit (in C++) erweitert werden.
FrameMaker wurde ursprünglich von der Firma ’Frame Technologies’ entwickelt und Mitte der 90er Jahre von Adobe® Systems übernommen. Die Unterstützung der XML-Verarbeitung ist erst seit der Version 7.0, die im Jahre 2002 erschienen ist, verfügbar.
Auszüge aus [WIK07, FrameMaker] und [Ado07b].

3. Dokumentenmanagement bei der Lufthansa AG

Flugbetriebsdokumentationen sind ein wichtiger Bestandteil im Lufthansa Konzern. Um technische Bedienungsanleitungen und Sicherheitsbestimmungen im Flugverkehr zu beschreiben, müssen umfangreiche Dokumente angefertigt werden. Da diese Dokumente ständigen Neuerungen unterliegen und dadurch ein erhöhter Verwaltungaufwand besteht, war eine Effizienzsteigerung durch ein Dokumentenmanagementsystem mit den heutigen technischen Möglichkeiten notwendig. Außerdem wurde die Anforderung für das Erzeugen von Dokumenten in verschiedenen Ausgabeformaten (HTML, PDF, etc), mit diesem Dokumentenmanagentsystem gelöst.
Im Zuge des Projektes 'EFOM' (Kapitel 3.1) wurde von der LSY ein Autoren- und Redaktionssystem entwickelt, um damit den Autoren die Arbeit mit den Dokumentationen zu erleichtern. In diesem System wird mit Dokumenten ohne Layoutinformationen gearbeitet. Es ist nur die Struktur der Dokumente hinterlegt (SGML-, bzw. XML-Daten). Die Autoren können sich dadurch gezielt auf den Inhalt konzentrieren, während die Layoutgestaltung und das Formatieren automatisch durchgeführt wird.

3.1. Das Projekt 'EFOM'

Das Projekt 'EFOM' der LSY startete im Jahr 1997 und ging im Jahre 2000 in die produktive Phase über. 'EFOM' steht für 'Electronic Flight Operations Manual' und ist ein Projekt im Auftrag der LH mit dem Ziel Flugbetriebsdokumentationen in einem automatisierten Prozess zu entwickeln, zu verwalten und zu publizieren. Dabei handelt es sich um Flugbetriebsdokumente mit Herstellerdaten (Airbus, Boeing), welche mit LH eigenen Informationen bearbeitet werden. Die LH erstellt auch Flugbetriebsdokumente für andere Airlines. Endanwender dieser Dokumente sind das fliegende Personal, die Personalvertretung und die Flotten.
Das Projekt ist bereits abgeschlossen und beinhaltet das Importieren von externen Daten (Herstellerdaten) in eine Datenbank, ein Workflowsystem zum Editieren und zur Qualitätssicherung, sowie einem Export zur elektronischen Publikation oder für die Druckausgabe.
Der Prozessablauf wird in Abbildung 3.1 anschaulich dargestellt.

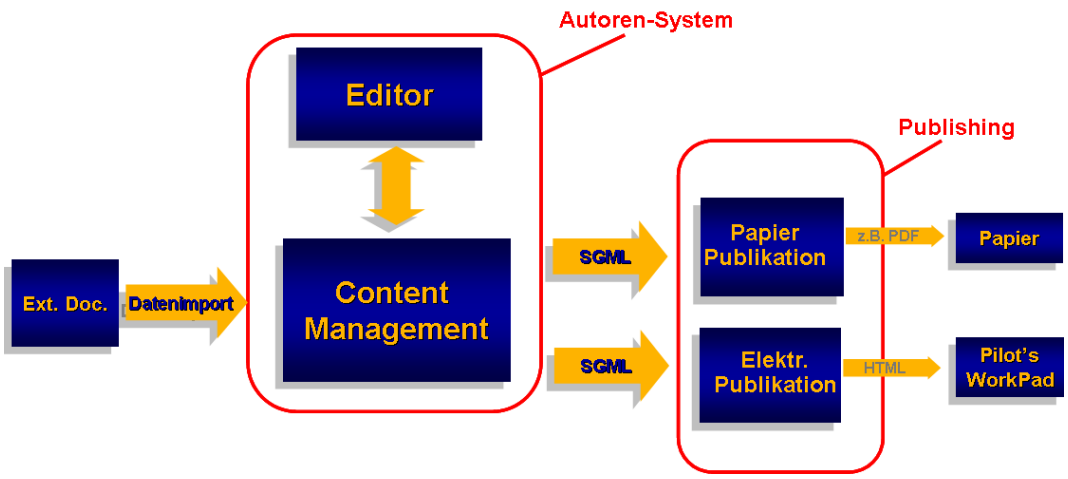


Abbildung 3.1.: EFOM: Prozessablauf (Quelle: [LSY06])

Das Content Management System wurde von der LSY eigens entwickelt und wurde unter dem Namen DocControl verkauft.
Daten, die in einer Datenbank gehalten werden, können mit dem DocControl Navigator eingesehen und mit einem integrierten XML-Editor bearbeitet werden. Durch die Arbeit mit einer Datenbank kann auf einzelne Abschnitte einer Dokumentation zugegriffen werden, ohne dabei das gesamte Dokument zu laden. Die Zugriffe auf die Daten unterliegen einem Workflow, der die Daten konsistent hält und zusätzlich eine Qualitätssicherung ermöglicht.
Die Dokumente werden strukturiert in SGML-Dateien aufgebaut. Dabei wird der strukturelle Aufbau der Dokumente durch eine LH eigene Flugbetriebs DTD beschrieben. Die Beschreibung in SGML ermöglicht die Weiterverarbeitung der Daten für verschiedene Anwendungsbereiche, wie etwa der Papierpublikation, die am Ende des Prozesses steht.

3.2. Von ’EFOM’ zu ’EFOM-2’

Das Projekt ’EFOM-2’ soll auf der Grundlage von ’EFOM’ Änderungen in der Verwaltung und Verarbeitung der Flugbetriebsdokumente durchgeführt werden, was vor allem auf technische Erneuerungen der letzten Jahre zurückzuführen ist.
Ein Hauptziel des Projektes ist die Umstellung von der SGML-Technologie zu der neueren XML-Technologie, die in den letzten Jahren an Popularität gewonnen hat und mittlerweile ein breites Spektrum an Weiterverarbeitungsmöglichkeiten bietet. Außerdem erfordert die Einführung des Airbus A380 bei der LH diese Umstellung, da die Herstellerdaten bereits im XML-Format bereitgestellt werden.
Ein weiteres Ziel ist der Übergang der objektorientierten Astoria-Datenbank zu einer relationalen Oracle-Datenbank. Gründe hierfür liegen nicht im Funktionsumfang, sondern vielmehr in der Zukunftssicherheit, da das Produkt ’Astoria’ in der Vergangenheit mehrmals aus finanziellen Gründen weiterverkauft wurde und dadurch Probleme bei der Gewährleistung von Support entstanden. Die Datenbank der Firma Oracle hingegen gehört derzeit zu den marktführenden Datenbanklösungen.
Mit dieser Umstellung verbunden ist auch die Einführung eines neuen Content Management Systems, welches mit dem Produkt ’COSIMA’ des Technologiepartners ’docufy GmbH’ erfolgt ist. ’COSIMA’ arbeitet mit relationalen Datenbanken und wird im Projekt ’EFOM-2’ bei der LSY mit eigenen Erweiterungen eingesetzt.
Der Prozessablauf bleibt nach außen derselbe wie bei ’EFOM’. Es sind lediglich einige Funktionen hinzugekommen, wie z.B. ein erweitertes Configuration Management, welches unter anderem ein redundanzfreies Arbeiten für den Autor ermöglicht. D.h. Gemeinsamkeiten zwischen Dokumentationsvarianten sollen auch gemeinsam bearbeitet werden können und nicht in jeder Variante einzeln.

3.3. Desktop-Publishing in ’EFOM’ und ’EFOM-2’

Am Ende des Dokumentenmanagements steht die Publizierung der Dokumente. Dies soll möglichst automatisch aus den vorliegenden SGML- oder XML-Daten geschehen und in verschiedenen Ausgabeformaten möglich sein.
Die Publikation für die Druckausgabe wird seit 1997 mit dem Produkt FrameMaker® der Firma

Adobe® Systems GmbH entwickelt. Die SGML-/XML-Daten werden in einem halbautomatisierten Prozess in Adobe® FrameMaker®, nachfolgend FrameMaker genannt, importiert und mit einem Layout versehen. Zur Vermeidung großer Datenmengen werden die SGML-/XML-Daten in einer Vorverarbeitung in Kapitel aufgesplittet. Aus diesen gesplitteten Dokumenten werden FrameMaker Dateien generiert, die später in FrameMaker wieder zu einem Gesamtbuch (Masterbuch) zusammengebunden werden. In einem manuellen Prozess werden die FrameMaker Dateien zu PDF-Dateien konvertiert. Durch die Zwischenschritte der Konvertierung liegt der Zeitaufwand einer Papier Publikation, je nach Umfang des zu publizierenden Dokumentes, zwischen 4 und 8 Stunden.
Zur Steigerung der Perfomanz wird ein Desktop-Publishing Prozess gesucht, der vollautomatisiert und mit weniger Zeitaufwand funktioniert.

4. Analyse

Im Grundlagenkapitel wurden bereits die Techniken beschrieben, die XML-basiert eine Darstellung erzeugen. Das Kapitel Analyse untersucht die Anforderungen mit Blick auf die Realisierbarkeit. Dabei gilt es zu entscheiden welche Technik das größte Potenzial bietet, um ein System aufzubauen, welches die Anforderungen der zu erstellenden Dokumente bestmöglicn erfüllt, sowie eine leichte Erweiterbarkeit für eventuell in Zukunft hinzukommende Dokumente mit ähnlichem Aufbau bietet. Anschließend werden die Möglichkeiten des heutigen Desktop-Publishing verglichen.

4.1. Layout-Anforderungen

4.1.1. Allgemein

Grundlage der zu generierenden PDF-Ausgaben sind XML-Dokumente. Die XML-Dokumente werden durch die von der LSY entwickelte 'ops-doc' DTD, momentan in der Version 4.2 vorliegend, beschrieben. Die nachfolgende Analyse beschäftigt sich ausschließlich mit den Dokumenten 'OM-A' und 'FSM'. Obwohl beide Dokumente derselben DTD folgen, ist das Layout beider Dokumente verschieden. Der Grundaufbau der Dokumente ist jedoch gleich. Abbildung 4.1 soll anhand eines Beispiels die Struktur der Dokumente veranschaulichen.

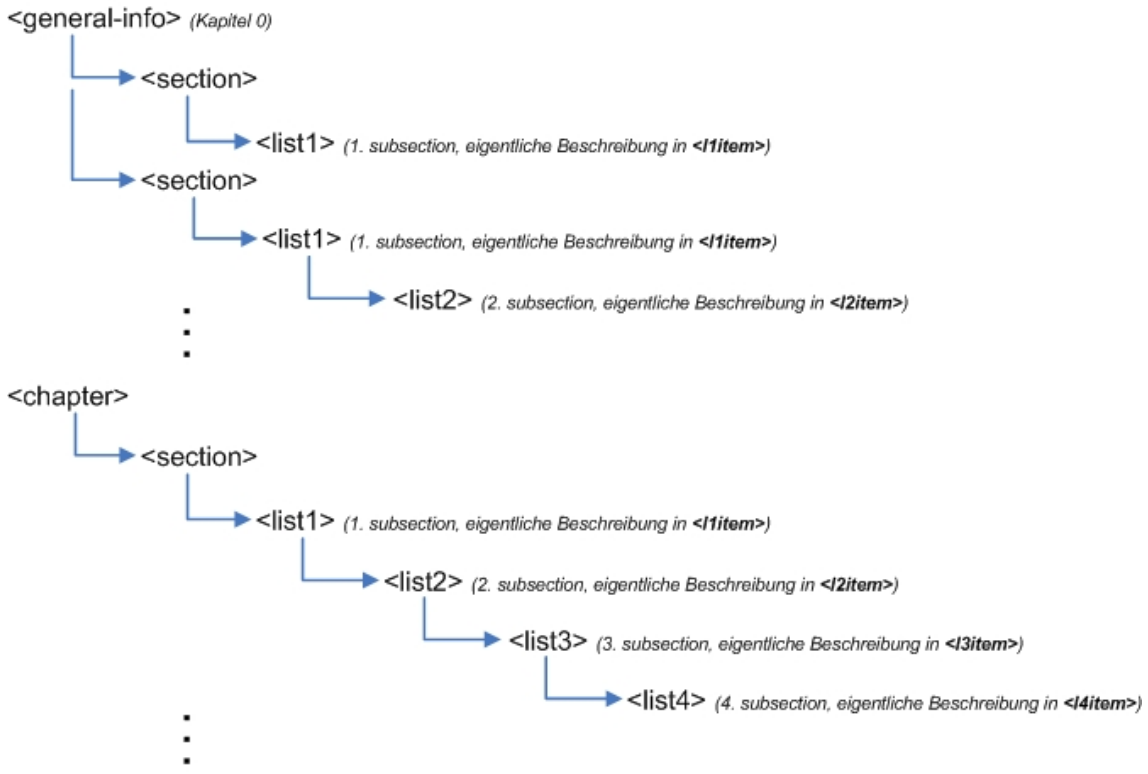


Abbildung 4.1.: grobe Struktur der DTD (Quelle: Eigene Darstellung)

Ein Dokument besteht also aus Kapiteln (*chapter*), welche Unterabschnitte bis zu einer 5.Ebene enthalten können. Dabei nimmt *general-info* eine Sonderstellung ein. So hat sie dieselbe Struktur wie *chapter*, taucht aber nur am Anfang eines Dokuments auf, wird also als Kapitel 0 dem Dokument vorangestellt. *general-info* kann noch einige zusätzliche Elemente enthalten, wie z.B. ein Glossar.
Da das zu generierende Dokument für die Papierpublikation gedacht ist und als Buch gebunden wird, werden linke und rechte Seite verschieden behandelt. So ist der Abstand zur Innenseite des Buches grösser ist als der Abstand nach aussen.
Jedes Kapitel beginnt auf einer rechten Seite. Endet ein Kapitel auch auf einer rechten Seite, so soll zusätzlich eine leere linke Seite generiert werden, um somit das neue Kapitel auf einer rechten Seite beginnen zu lassen. Ein Kapitel führt die Seitenzahlen des vorangegangenen Kapitels nicht weiter, sondern startet die Seitennummerierung neu.

Kopf- und Fußzeilen

Das Ausgabedokument soll eine Kopfzeile und eine Fußzeile enthalten. In der Kopfzeile werden Informationen angezeigt wie Dokumentenname, *chapter*- und *section*-name, Kapitel-, Seiten- und Revisionsnummer. Die Fußzeile enthält lediglich ein Copyright-Vermerk, sowie das LH Logo. Das Logo ist mit der LH eigenen TrueType-Schrift (font-name: 'Lufthansa') erstellt.
Außer der Schriftart 'Lufthansa' werden noch weitere TrueType-Schriftarten verwendet, welche intern zur Verfügung stehen. Beide Dokumente verwenden verschiedene Schriftarten.
Eine genauere Beschreibung zu Kopf- und Fußzeilen findet sich im jeweiligen Kapitel (Kap. 4.1.2 und Kap. 4.1.3).

Der Inhaltsbereich (*body*)

Der eigentliche Inhalt des Dokuments, der sogenannte *body*, besteht vorwiegend aus Fließtext ohne sonderlich kompliziertem Layout. Häufig vorkommende Elemente sind *<para>*, für einfachen Fließtext (ein para-Element beschreibt einen Absatz), und *<table>* für Tabellen. Es gibt nur wenige Elemente die einer besonderen Beachtung bedürfen.
Eines davon ist das Element *index*. Ein Index wird im FSM nicht erstellt, jedoch findet es Verwendung im OM-A. Das Layout wird im nächsten Abschnitt (Kapitel 4.1.2) beschrieben.

Tabellen sind nach dem CALS Tabellenmodell modelliert[1]. Dieses Modell ermöglicht den Einsatz von *colspec* Elementen, welche die Spalten definiert, und *spanspec* Elementen zur Definition von verbundenen Spalten. Zur Berechnung des Tabellenaufbaus muss also auf diese Elemente zurückgegriffen werden. Die eigentliche Tabelle hat einen Kopfbereich (*thead*), einen Fußbereich (*tfoot*) und einen Inhaltsbereich (*tbody*). Kopf- und Fußzeilen müssen auf jeder Seite der Tabelle sichtbar sein. D.h., dass bei einem eventuell vorkommendem Seitenumbruch der Tabelle die Kopf- und Fußzeilen neu generiert werden müssen.
Abbildung 4.2 zeigt den strukturellen Aufbau von Tabellen nach dem CALS Modell.

[1] Siehe dazu: http://www.oasis-open.org/specs/a502.htm

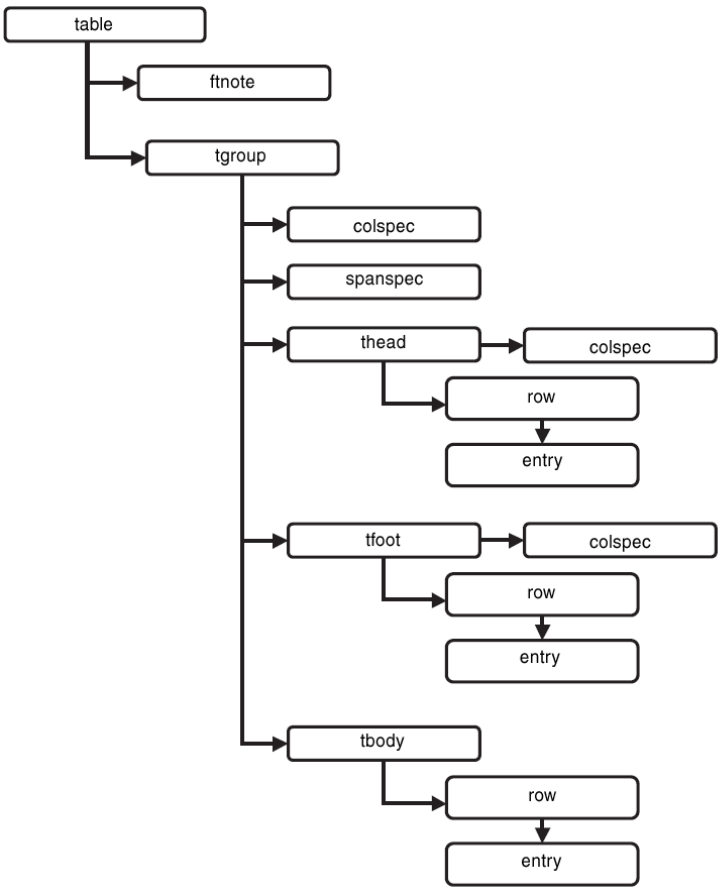


Abbildung 4.2.: Struktur des CALS Tabellenmodell (Quelle: Eigene Darstellung)

Eine Tabelle hat diverse Attribute, die das Layout beeinflussen. Beispielhaft seien hier genannt:

- *frame*, *colsep* und *rowsep*: Zur Anzeige von Begrenzungslinien der Tabelle
- *size*: Zur Grössenänderung verschiedener Tabelleneigenschaften
- *spacing*: Zur Veränderung von Abständen innerhalb einer Zelle

Unnummerierte Listen (*unumlist*) können theoretisch unendlich tief geschachtelt sein. Ein Eintrag einer Liste wird mit einem vorangestellten Symbol als Label gekennzeichnet. Bei Verschachtelungen muss jede Ebene entsprechend eingerückt werden.
Bei nummerierten Listen (*numlist*) bekommen nur die ersten 2 Ebenen ein Label. Auf der ersten Ebene werden die Einträge mit laufender Nummer markiert ([’1)’, ’2)’, ’3)’, ...]), und auf der zweiten Ebene mit Kleinbuchstaben ([’a)’, ’b)’, ’c)’, ...]). Ab der 3. Ebene entfällt das Label. Ansonsten ist der Aufbau derselbe wie bei den unnummerierten Listen.

Weitere Merkmale

Die Dokumente OM-A und FSM erhalten zusätzlich ein Inhaltsverzeichnis. Das Inhaltsverzeichnis selbst ist nicht im XML-Dokument beschrieben, sondern geht aus der Struktur der Daten hervor. Der Aufbau des Inhaltsverzeichnisses ist bei beiden Dokumenten ähnlich. Im Inhaltsverzeichnis sollen Kapitel, *sections* und *subsections* der 1. Ebene (*l1item*) mit Abschnittsnummer

(Attribut *fixnum*) und Seitenzahl dargestellt werden. Weitere Details zu den Inhaltsverzeichnissen werden im jeweiligen Abschnitt (Kapitel 4.1.2 bzw. Kapitel 4.1.3) besprochen.

Ein weiteres besonderes Merkmal ist die Vergabe der Kapitel- und Abschnittsnummern. Jedes Kapitel und jeder Abschnitt bekommt einen Titel (*title*) und eine Nummer. Diese Nummer wird zusammengesetzt aus den *fixnum*-Attributen und wird dem Titel vorangestellt. Die abschnittsbeschreibenden Elemente (*chapter, section, l1item* etc.) haben ein Attribut *fixnum*, worin nur die Nummer des aktuellen Abschnitts hinterlegt ist. Befinden wir uns z.B. in einem *l1item*, so müssen die *fixnum*-Attribute von *section* und *chapter* ausgelesen werden, um die Nummerierung für den Abschnitt zu erstellen.

Layout-Spezifikation

Zusätzlich zum logischen Aufbau ist auch die optische Platzierung der Elemente auf dem Papierlayout sehr wichtig. Für jedes Dokument sind die Abstände und Schriftgrößen genau definiert, so dass jedes Element seinen definierten Platz auf der Seite findet.
Das Layout ist sehr individuell gestaltet, so ist z.B. vorgegeben, wie groß der Abstand eines Textes in einer Tabellenzelle zu den Zellenrändern ist. In diesem Fall gibt es durch Setzen eines Attributs verschieden definierte Abstände.
Der folgende Code-Auszug ist aus dem 'Element Definition Document', kurz EDD, des FrameMaker® Prozesses, der im Projekt 'EFOM' eingesetzt wurde, entnommen. Das EDD beschreibt bei FrameMaker® anhand einer DTD das Layout-Verhalten der einzelnen Elemente. Hier das Element *tgroup* aus *table*:

```
Element (Table): Tgroup
 ...
 1. If context is: Table[Size="Small"]
  Default font properties
    Size: 10 pt
   Basic properties
    Paragraph spacing
     Space below: 4 pt
```

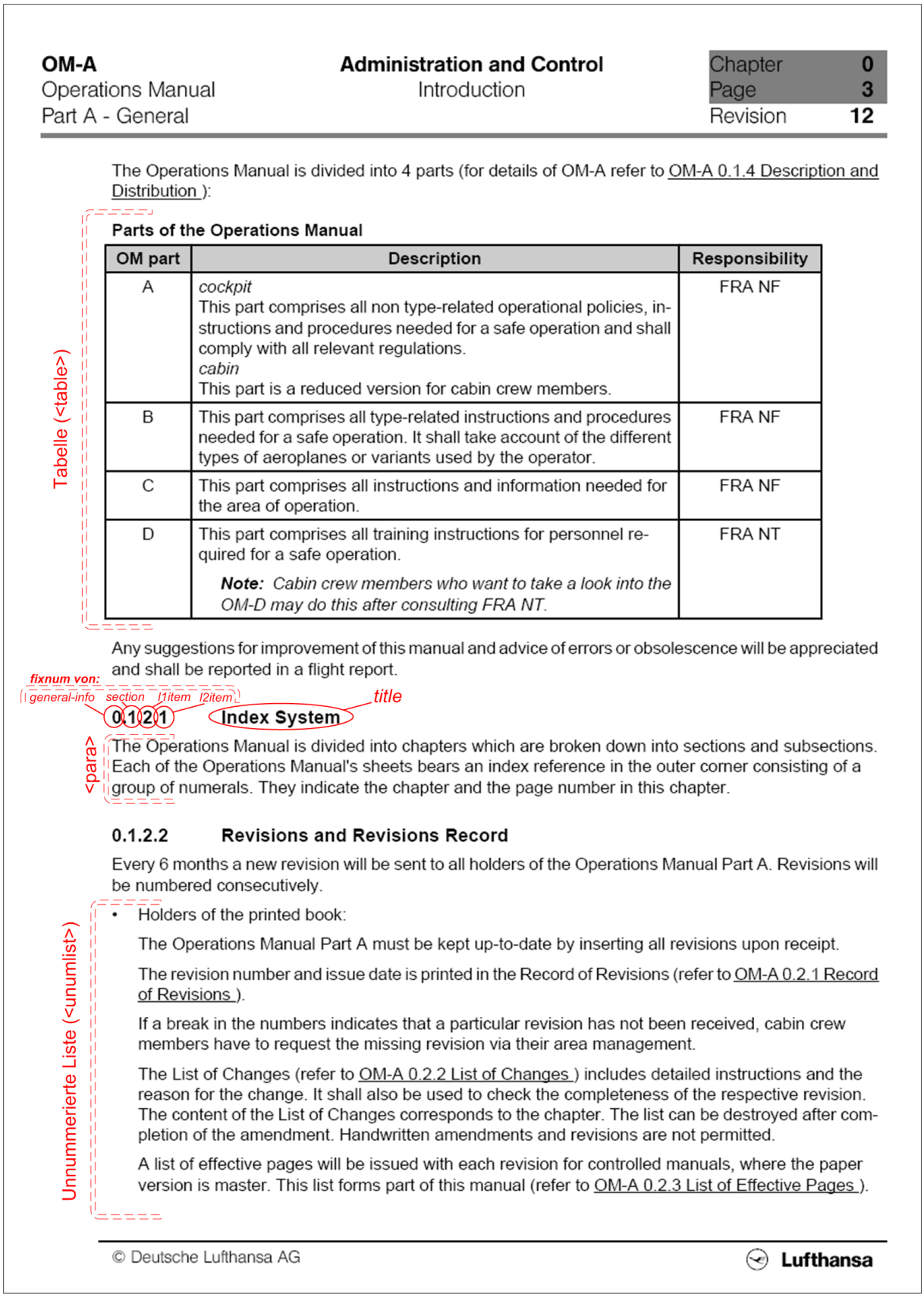
Beispielhafter Auszug aus der opsdoc-edd (Quelle: [LSY07])

Wenn das Attribut *size* von *table* auf „small“ gesetzt ist, verändert sich nicht nur die Schriftgröße, sondern auch der Abstand des Textes innerhalb einer Zelle zum Zellenrand.

Insgesamt gibt es sehr viele solcher genau definierten Abstände. Alle Layout-Vorgaben sollen möglichst eingehalten werden.
Die Layout-Vorgaben wurden gesammelt und sind als Anlage auf der CD[2] zusammengefasst.

Abbildung 4.3 zeigt eine Beispielseite aus dem Dokument OM-A mit einigen der bereits genannten Elemente.

[2] Siehe dazu: *CD:<//LSY/Doku/layout-definition.xls>*



The Operations Manual is divided into 4 parts (for details of OM-A refer to OM-A 0.1.4 Description and Distribution):

Tabelle (<table>)

Parts of the Operations Manual

OM part	Description	Responsibility
A	*cockpit* This part comprises all non type-related operational policies, instructions and procedures needed for a safe operation and shall comply with all relevant regulations. *cabin* This part is a reduced version for cabin crew members.	FRA NF
B	This part comprises all type-related instructions and procedures needed for a safe operation. It shall take account of the different types of aeroplanes or variants used by the operator.	FRA NF
C	This part comprises all instructions and information needed for the area of operation.	FRA NF
D	This part comprises all training instructions for personnel required for a safe operation. ***Note:*** *Cabin crew members who want to take a look into the OM-D may do this after consulting FRA NT.*	FRA NT

Any suggestions for improvement of this manual and advice of errors or obsolescence will be appreciated and shall be reported in a flight report.

fixnum von: general-info section l1item l2item

0.1.2.1 **Index System** *title*

<para>

The Operations Manual is divided into chapters which are broken down into sections and subsections. Each of the Operations Manual's sheets bears an index reference in the outer corner consisting of a group of numerals. They indicate the chapter and the page number in this chapter.

0.1.2.2 Revisions and Revisions Record

Every 6 months a new revision will be sent to all holders of the Operations Manual Part A. Revisions will be numbered consecutively.

Unnummerierte Liste (<unumlist>)

- Holders of the printed book:

  The Operations Manual Part A must be kept up-to-date by inserting all revisions upon receipt.

  The revision number and issue date is printed in the Record of Revisions (refer to OM-A 0.2.1 Record of Revisions).

  If a break in the numbers indicates that a particular revision has not been received, cabin crew members have to request the missing revision via their area management.

  The List of Changes (refer to OM-A 0.2.2 List of Changes) includes detailed instructions and the reason for the change. It shall also be used to check the completeness of the respective revision. The content of the List of Changes corresponds to the chapter. The list can be destroyed after completion of the amendment. Handwritten amendments and revisions are not permitted.

  A list of effective pages will be issued with each revision for controlled manuals, where the paper version is master. This list forms part of this manual (refer to OM-A 0.2.3 List of Effective Pages).

Abbildung 4.3.: Beispiel einer OM-A Seite mit Inhalt (Quelle: Dokument OM-A der LSY)

4.1.2. Operations Manual - Part A (OM-A)

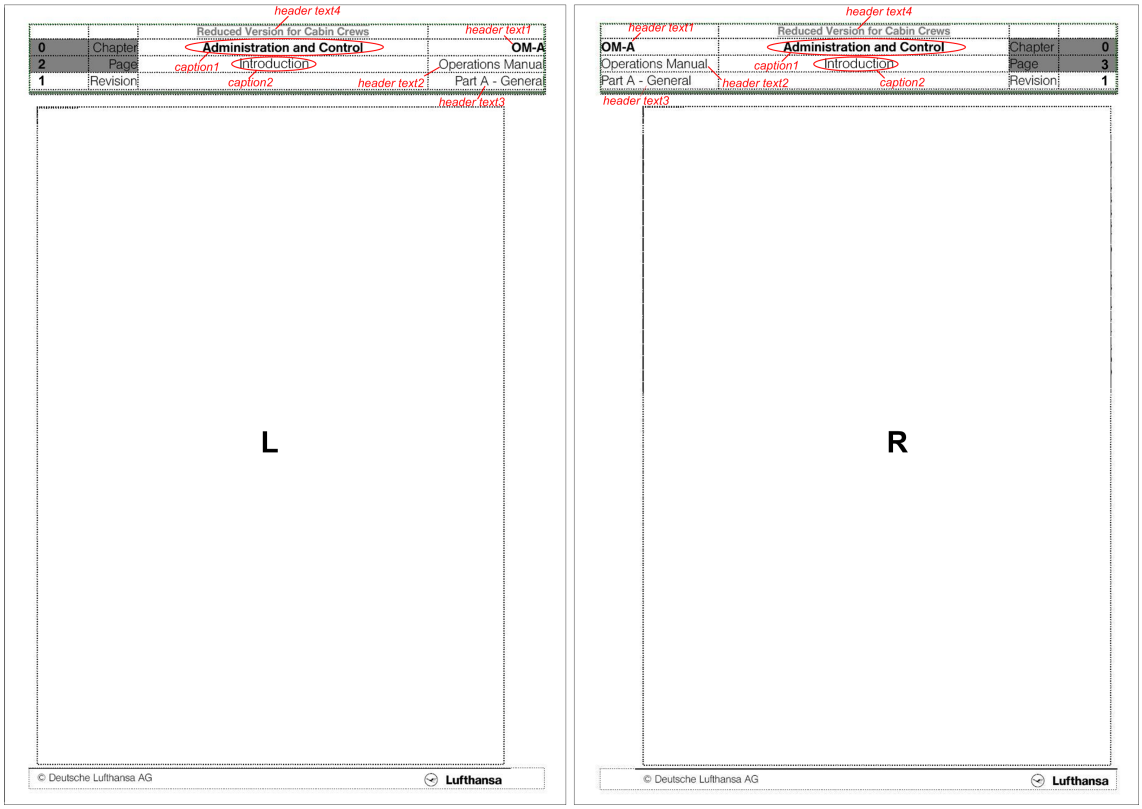


Abbildung 4.4.: Exemplarischer Aufbau der linken und rechten Seite vom OM-A (Quelle: Eigene Darstellung)

Die Seitenvorlagen für die linke und rechte Seite des 'Operations Manual - Part A' sind in Abbildung 4.4 exemplarisch dargestellt. Die Rahmen auf der Abbildung sind nur zur Verdeutlichung der Aufteilung der Bereiche aufgezeichnet.
Der Kopfbereich hat fünf Zeilen und vier Spalten. Der Aufbau des Kopfbereichs ist bei linker und rechter Seite spiegelverkehrt. D.h. die Informationen zur Revisionsnummer und der aktuellen Kapitel- und Seitennummer sind immer auf der Aussenseite zu finden. In der Innenseite sind für die drei dargestellten Texte folgende Strings vorgegeben:

- *header text1*: OM-A
- *header text2*: Operations Manual
- *header text3*: Part A - General

Mittig angezeigt sind drei verschieden generierte Überschriften. Der erste Text (*header text4*) beschreibt optional eine einschränkende Gültigkeit des Dokuments (Beispiel: 'Reduced version for cabin crews'). Das Element *publishing-properties* in *general-info* definiert diese optionale Anmerkung. Die Überschriften (*caption1* und *caption2*) beschreiben den Titel des aktuellen Kapitels bzw. der aktuellen *section*.
In der untersten Zeile ist ein horizontaler Trennstrich der den Kopfbereich vom Inhaltsbereich optisch aufteilt.
In der Kopfzeile werden die Schriften 'Lufthansa Light' und 'Lufthansa Medium' verwendet[3].

[3]Weitere Details können der *layout-definition* entnommen werden (*CD:<//LSY/Doku/layout-definition.xls>*)

Für den Inhaltsbereich (Body) wird die Schriftart 'Arial' verwendet. Ein Inhaltsverzeichnis wird jeweils am Anfang eines Kapitels generiert und enthält Informationen zu *sections* und *subsections* (*section* und *l1item*) des aktuellen Kapitels. Das Inhaltsverzeichnis ist in 3 Spalten aufgeteilt, so dass sich eine Tabelle als Layoutmittel eignet. In der ersten Spalte steht die *section*- bzw. *subsection*-Nummer, in der zweiten Spalte der zugehörige Titel und in der dritten Spalte die zugehörige Seitenzahl. Die Informationen zu den *sections* werden zur Übersichtlichkeit fett gedruckt und zusätzlich wird vor jede *section* eine Leerzeile generiert.

Leere Seiten (*blank pages*) die im Dokument generiert werden, wenn ein Kapitel auf einer rechten Seite endet (Leere Seiten werden als linke Seiten eingefügt damit ein neues Kapitel auf einer rechten Seite beginnen kann), bekommen im OM-A den Vermerk „Intentionally Left Blank". Dieser wird horizontal zentriert mit 95mm Abstand vom oberen Textrand dargestellt. Kopf- und Fußzeilen sollen hierbei normal behandelt werden.

Der Index der im OM-A generiert wird weicht vom normalen Layout ab. Er wird zweispaltig ausgedruckt und in Kapitel 0.1.7 hinein generiert. Der Abstand zwischen den zwei Spalten beträgt exakt 7mm. Im Index werden alle Einträge gesammelt, die im gesamten Dokument mit dem Element *<index>* gekennzeichnet wurden. Diese werden alphabetisch sortiert und mit den Abschnittsnummern, in denen das jeweilige Stichwort zu finden ist, versehen (siehe Abbildung 4.5).

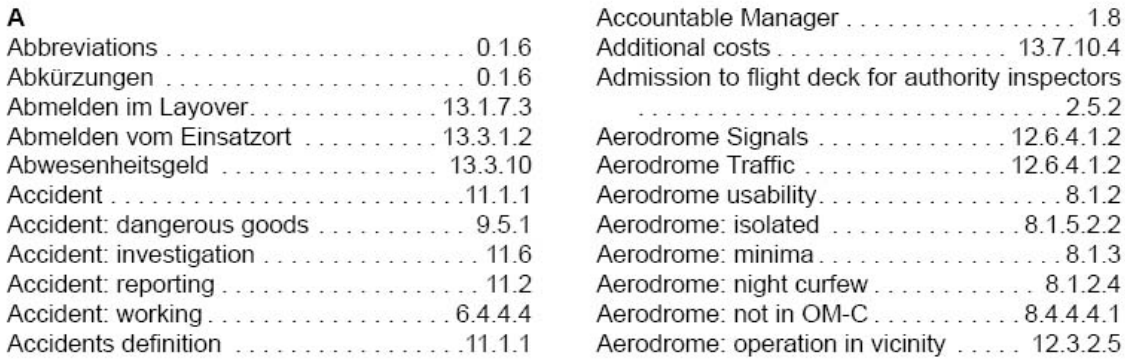


Abbildung 4.5.: zweispaltiger Aufbau vom Index (Quelle: Dokument OM-A der LSY)

An dieser Stelle soll nicht jedes einzelne Detail des Layouts beschrieben werden. Wichtig ist die Auflistung der besonderen Elemente, sowie die Verdeutlichung der strengen Vorgaben, die an das Layout gegeben sind. Viele der Layoutvorgaben werden im Kapitel 6 wieder aufgegriffen und anhand der Umsetzung beschrieben.

4.1.3. Flight Safety Manual (FSM)

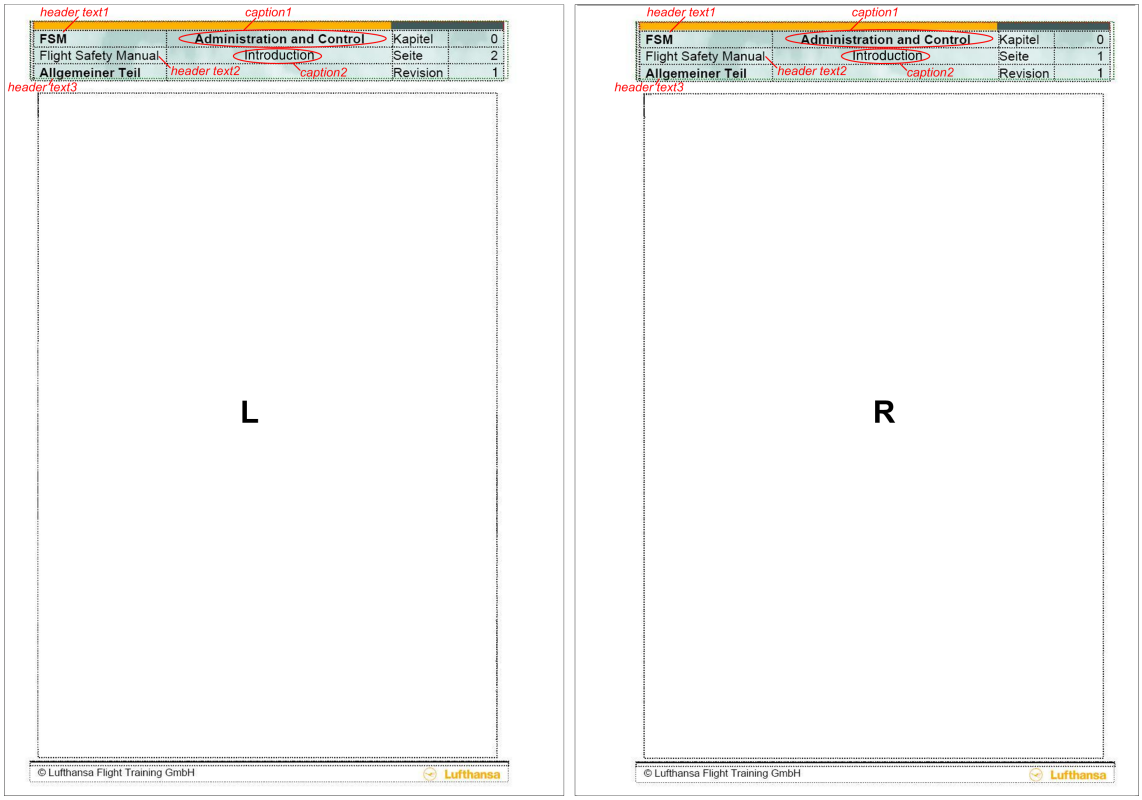


Abbildung 4.6.: Exemplarischer Aufbau der linken und rechten Seite vom FSM (Quelle: Eigene Darstellung)

Der Aufbau der linken und rechte Seite des 'Flight Safety Manual' ist in Abbildung 4.6 dargestellt. Anders als beim OM-A ist der Kopfbereich der beiden Seiten nicht spiegelverkehrt, sondern identisch aufgebaut (nur mit unterschiedlichen Abständen zum Seitenrand). Außerdem ist ein Hintergrundbild im Kopfbereich hinterlegt. Hierbei handelt es sich um eine einfache png-Grafik, die über den gesamten Kopfbereich gespannt wird. Revisions-, Kapitel- und Seitennummern sind in diesem Dokument immer im Kopfbereich auf der rechten Hälfte zu finden. Mittig sind die Informationen zum Kapitel- und *section*-Namen (*caption1* und *caption2*) enthalten. Auf der linken Hälfte sind die Texte *header text1* und *header text2* fest vorgegeben:

- *header text1*: FSM
- *header text2*: Flight Safety Manual

header text3 ist hier nicht fest vorgegeben sondern bekommt eine Sonderbehandlung. Der hier angezeigte Text wird bestimmt durch das Element *publishing-property*. Ist z.B. im Dokument eine *publishing-property* mit folgenden Werten vorhanden,

```
<publishing-property name="appltype" value="opsdoc-fsm-allg-de">
```

so bekommt *header text3* den Wert: „Allgemeiner Teil“. Weitere mögliche Werte für die *publishing-properties* sind der *layout-definition* zu entehmen[4].
Die Standardschriftart, die im Kopf-, Fußzeilen und auch im Inhaltsbereich (Body) verwendet

[4] *CD:<//LSY/Doku/layout-definition.xls>*

wird, ist 'Helvetica'. Das FSM bekommt ein komplettes Inhaltsverzeichnis am Anfang des Dokuments (Kapitel 0.1). Der Aufbau ist dem Inhaltsverzeichnis des OM-A ähnlich. Lediglich werden zusätzlich zu den Informationen der *sections* und *subsections*(*l1item*) auch die Kapitelnummern und Kapitelnamen angezeigt. Eine Leerzeile wird in diesem Inhaltsverzeichnis zwischen den einzelnen Kapiteln generiert. Ansonsten ist auch hier die Darstellung von Abschnittsnummer, Titel des Abschnitts und zugehöriger Seitennummer, zu finden.

Beim FSM bekommen die *blank pages* in deutschen Dokumenten den Vermerk: „Aus redaktionellen Gründen frei". Dieser wird ebenfalls horizontal zentriert und bekommt hier einen Abstand von 100mm zum oberen Textrand. Bei Dokumenten mit englischer Sprache lautet der Vermerk wie beim OM-A: „Intentionally Left Blank". Die Unterscheidung welcher Sprache das Dokument angehört, erfolgt durch das Attribut *language* im *root*-Element des Dokuments.

Die Elemente *<warning>*, *<caution>* und *<note>* werden im FSM mit Grafiken besonders hervorgehoben. Dies verdeutlicht die Wichtigkeit der enthaltenen Informationen. Hier ist das Layout etwas aufwendiger als der normale Fließtext, da mit grafischen Elementen gearbeitet wird, die an eine bestimmte Stelle, wieder mit definierten Abständen, platziert werden müssen. Auf der linken Hälfte dieser Elemente befindet sich eine PNG-Grafik. Rechts davon ist der Text dieses Elements zu finden. Ober- und unterhalb wird das Element mit horizontalen Trennbalken eingeschlossen (siehe Abbildung 4.7).

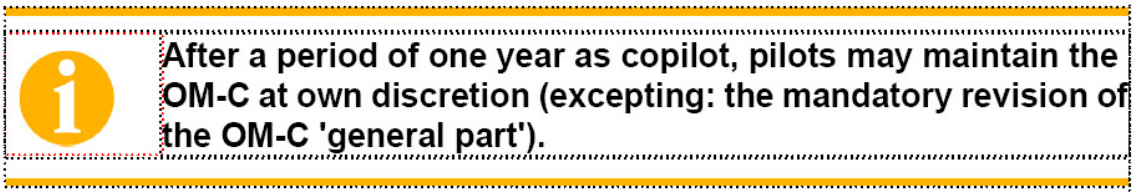


Abbildung 4.7.: FSM: ein *<note>*-Element (Quelle: Eigene Darstellung)

Auch hier soll nicht weiter auf Details eingegangen werden. Die wichtigsten Unterschiede zwischen dem OM-A und dem FSM sind bereits genannt worden[5].

[5]Weitere Details zum FSM Layout: *CD:<//LSY/Doku/layout-definition.xls>*

4.1.4. Zusammenfassung

Beide Dokumente, das OM-A und das FSM, haben einen sehr ähnlichen Aufbau. Elemente bei denen wenige Gemeinsamkeiten vorhanden sind und daher getrennt behandelt werden sollten, sind die Inhaltsverzeichnisse, das Erstellen eines Indexes und die Elemente 'warning', 'caution' und 'note'.
Die wichtigsten Eigenschaften der Dokumente wurden während der Analyse der Layout-Anforderungen gesammelt. Die wichtigsten Anforderungen an ein Desktop-Publishing-System die sich daraus ergeben, sind:

- individuelle Layoutgestaltung (Setzen von Abständen etc.)
- verschiedenes Layout für linke und rechte Seite
- Generieren von blank pages
- Kopfzeilen- und Fußzeilenbehandlung
- Verarbeitung von CALS-Tabellen
- zweispaltiges Layout für *index*
- Nutzung von TrueType-Fonts
- Nutzung von png-Grafiken
- Erzeugen grafischer Elemente (horizontale Balken etc.)

4.2. Untersuchung der DTP-Lösungen

In diesem Abschnitt werden nach Sammlung der Anforderungen in Kapitel 4.1 die verschiedenen Lösungsansätze zur Realisierung eines XML-basierten Desktop-Publishing-Systems besprochen. Betrachtet werden hier nur drei Lösungsansätze, die momentan eine große Bedeutung im heutigen DTP haben. Folgende Lösungsmöglichkeiten werden verglichen:

- LaTeX
- XSL-FO
- Adobe® FrameMaker®

Den meisten Komfort bieten die WYSIWYG-Tools (hier: Adobe® FrameMaker®), bei denen das Layout über eine grafische Oberfläche direkt sichtbar und manipulierbar ist. Die größte Kontrolle über das Layout ist über die eigene Beschreibung in Textsatzsystemen, wie XSL-FO und LaTeX, erreichbar. Der Nachteil ist, dass das Layout selbst programmiert werden muss.

4.2.1. LaTeX

Layout und Logik

LaTeX soll den Anwender bei den mühsamen Layout-Arbeiten unterstützen, indem es vorgefertigte Klassen bereitstellt, in denen bereits das grundsätzliche Layout vorgegeben ist. Die drei vorhandenen Standardklassen sind die Klassen 'book' für komplette Bücher, 'report' für umfangreiche Dokumente und 'article' für mittelgroße und kleine Dokumente.
Für dieses Projekt wäre die Wahl auf die Klasse 'book' gefallen, da es sich um umfangreiche technische Dokumentationen in Buchform handelt. Standardeinstellungen bei einem 'book' sind z.B. beidseitiger Druck und das Generieren eines neuen Kapitels auf der nächsten freien rechten Seite.
Das Seitenlayout ist mit Standardeinstellungen vordefiniert, so dass bei nicht so strengen Layout-Vorgaben keine/kaum Änderungen mehr nötig sind. So kann sich der Anwender auf den logischen Aufbau des Dokuments konzentrieren und muss nicht jede Feinheit des Layouts definieren. In unserem Beispiel sind die Anforderungen jedoch sehr streng, was eine eigene Definition des Seitenlayouts unumgänglich macht.
Die Abstände, um das Seitenlayout festzulegen, sind über Parameter selbst definierbar. Abbildung 4.8 zeigt den Aufbau einer LaTeX-Seite mit den Parametern zur Abstandsdefinition.
Der Abstand nach links von rechten Seiten wird beispielsweise durch folgenden Befehl neu definiert:

```
\setlength{\oddsidemargin}{12mm}
```

Die Kopf- und Fußzeilen des Dokuments können bei rechter und linker Seite getrennt behandelt werden. Die Standard Kopf- und Fußzeilen, die in LaTeX mit dem Befehl:

```
\pagestyle{headings}
```

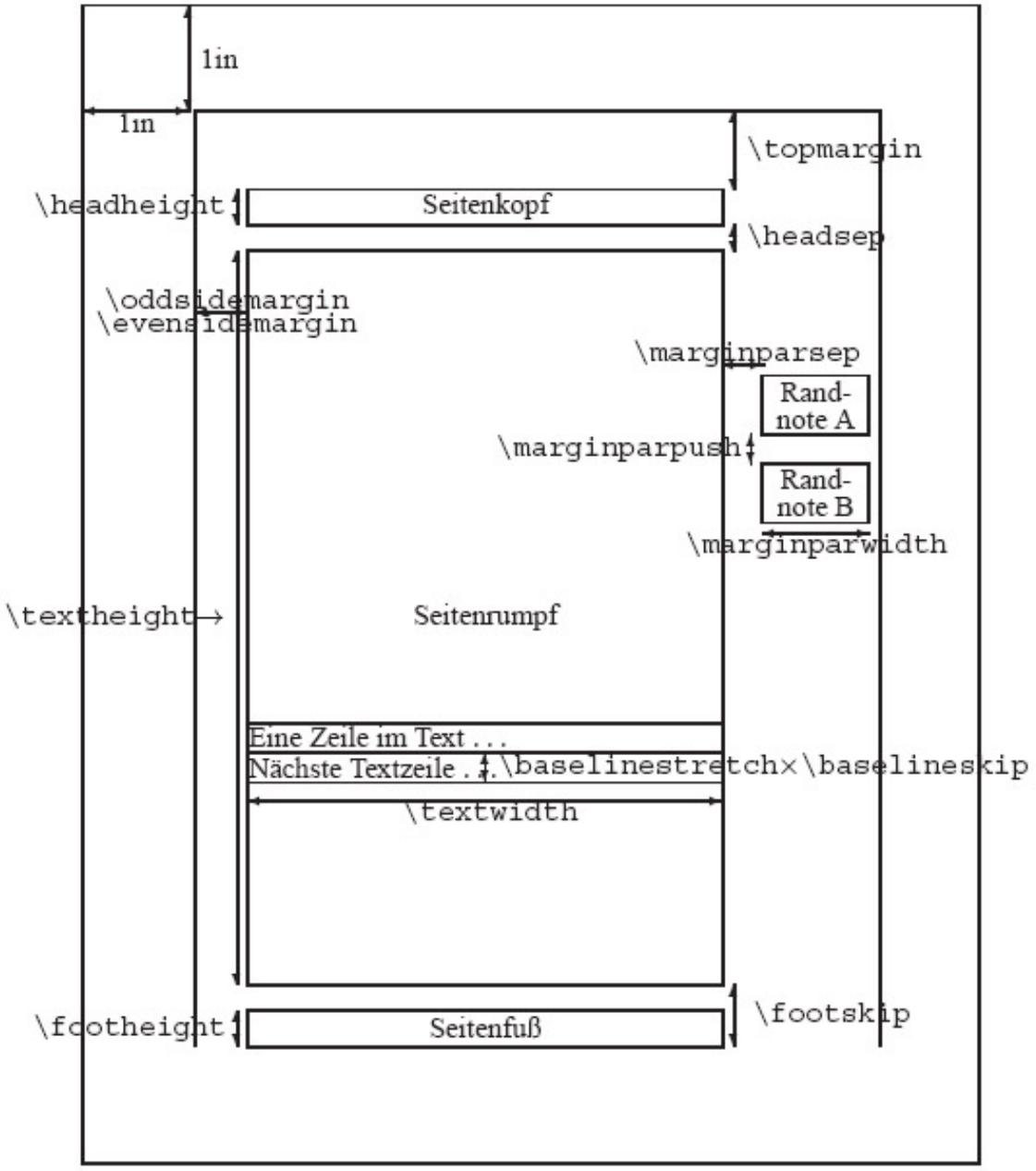


Abbildung 4.8.: LaTeX: Grundaufbau der Seite mit Parametern zur Abstandsdefinition (Quelle: [Jür95, Seite 20])

eingebunden werden, eignen sich aber nicht zur Nutzung in unserem konkreten Fall, da nur Kopfzeilen ohne Fußzeilen generiert werden und die Kopfzeile lediglich aus Seitennummer, *section*-Name und *subsection*-Name besteht. Diese sind auch nicht erweiterbar.
Für die Definition eigener Kopf- und Fußzeilen gibt es das Paket *fancyheadings*, welches zunächst eingebunden werden muss. Mit diesem Paket können Informationen in einen linken, mittleren und rechten Bereich einer Kopf- bzw. Fußzeile geschrieben werden. Kopf- und Fußzeilen bieten standardmäßig einen Trennbalken an, dessen Größe mit den Befehlen:

```
\setlength{\headrulewidth}{x pt}
\setlength{\footrulewidth}{x pt}
```

angepasst werden kann.
Werden die Kopfzeilen der vorliegenden Dokumente OM-A und FSM in drei Bereiche aufgeteilt, entstehen Probleme bei der Beschreibung der Kapitel-, Seiten- und Revisionsnummer (im FSM der rechte Bereich). Der angezeigte Text steht linksbündig und die Nummer dazu rechtsbündig (siehe Abbildung 4.6). Dafür wird eine Aufteilung in vier Bereiche notwendig, was mit dem Paket 'fancyheadings' leider nicht möglich ist. Somit benötigt dieses Problem einer weiteren Untersuchung zur Lösungsfindung, welche hier nicht weiter ausgeführt wird.

Die Umsetzung von Standardelementen, wie z.B. Tabellen und Listen, ist in LaTeX recht einfach. Listen werden beispielsweise wie folgt erstellt:

```
\begin{itemize}
  \item erster Punkt
  \item zweiter Punkt
  \item dritter Punkt
\end{itemize}
```

Dagegen ist das Verändern des vorgegebenen Layouts für individuelle Anforderungen bei jeglichen LaTeX-Elementen recht umständlich und meist mit einer langen Internetrecherche verbunden, da die vorhandene Literatur zu LaTeX selten auf die Details eingeht. Bei Listen ist z.B. standardmäßig ein Zeilenabstand nach oben vorhanden. Um diesen Abstand zu entfernen wird man in Internetforen beispielsweise folgenden Lösungsansatz finden:

```
Definition einer neuen Umgebung:

 \newlength{\tabitemtopsep}
   \addtolength{\tabitemtopsep}{\baselineskip}
 \newlength{\tabitembottomsep}
   \addtolength{\tabitembottomsep}{\baselineskip}
 \newenvironment{tabitemize}{
   \hbox{}\vspace{-\tabitemtopsep}\begin{list}{\textbullet}{
   \setlength{\labelsep}{0.2cm}
   \setlength{\leftmargin}{1.0em}
   \setlength{\topsep}{0.5ex}
   \setlength{\parsep}{0.5ex}  % {0.5ex plus0.2ex minus0.9ex}%
   \setlength{\itemsep}{0ex}}  % {0ex plus0.2ex}}%
  }{%
   \hbox{}\vspace{-\tabitembottomsep}\end{list}%
  }

Anwendung der neuen Umgebung:

 \begin{tabitemize}
   \item erster Punkt
   \item zweiter Punkt
   \item dritter Punkt
 \end{tabitemize}
```

Listen in LaTeX: Verändern des Abstandes nach oben (Quelle: [MRU07])

Zur Erstellung von Tabellen hat LaTeX ein sehr eigenes Modell, welches dem CALS-Table Modell in keinster Weise ähnelt. Eine Verarbeitung von CALS Tabellen in LaTeX ist daher sehr aufwendig. Eine Möglichkeit wäre eine Vorkonvertierung über XSLT[6].

[6] Ein Lösungsansatz ist zu finden unter: http://www.vistair.com/newtbl/XSLTableCode.html

Für zweispaltige Abschnitte gibt es das Paket 'multicol'. Nach Erstellen eines zweispaltigen Abschnitts, kann mit diesem Paket der Abstand der Spalten verändert, sowie ein zusätzlicher vertikaler Trennbalken mit einer bestimmten Dicke versehen werden. Der Abschnitt kann mitten im Dokument auftauchen und ist einfach anzuwenden. Zusätzlich kann der Abschnitt mit einem Titel versehen werden. Die genaue Positionsbestimmung des Titels ist wieder etwas aufwendiger und in den meisten Referenzen zu LaTeX nicht zu finden.

LaTeX generiert bei einer 'book'-Klasse eine *blank page* am Ende eines Kapitels, wenn es auf einer rechten Seite endet. Diese *blank page* verfügt nur über die Kopf- und Fußzeile und enthält keinen Inhalt. Zur Generierung eines Inhaltes muss der Befehl '\cleardoublepage' neu definiert werden:

```
\makeatletter
\def\cleardoublepage{\clearpage\if@twoside \ifodd\c@page\else
  \hbox{}
  \vspace*{\fill}
  \begin{center}
    This page intentionally contains only this sentence.
  \end{center}
  \vspace{\fill}
  \thispagestyle{empty}
  \newpage
  \if@twocolumn\hbox{}\newpage\fi\fi\fi}
\makeatother
```

Neudefinition des Befehls 'cleardoublepage' (Quelle: [Oos04])

TrueType-Fonts und grafische Elemente

LaTeX benutzt nicht die Windows-Fonts, sondern stellt ein eigenes Kontingent an Schriftarten zur Verfügung. Die Standardschriften bei LaTeX sind:

- roman (Proportionalschrift mit Serifen)
- sans serif (Proportionalschrift ohne Serifen)
- typewriter (Nichtproportionalschrift mit Serifen)

Darüber hinaus existieren weitere Schriftarten die über bestimmte LaTeX-Befehle aufgerufen werden können (siehe z.B. [Jür00, Kapitel 29].
Zur Nutzung von TrueType-Fonts in LaTeX reicht es leider nicht aus sie als ttf-Datei vorliegend zu haben, um sie dann über einen Befehl einzulesen.
Die Schritarten befinden sich bei LaTeX in sogenannten tfm(TeX font metrics)-files. Die Verbindung zu diesen Schriftarten wird über fd(font definition)-files hergestellt. Bei der Konvertierung in ein Ausgabeformat (PDF oder PS) werden die TeX Fonts umkodiert, so dass sie beispielsweise von Adobe®'s PDF interpretiert werden können ('afm' = Adobe font metrics). Bei diesem Schritt muss dann unter anderem wieder eine Verbindung zwischen den TeX fonts und den Adobe font metrics hergestellt werden. Dies geschieht in den map-Dateien.
Um letztendlich TrueType-Fonts nutzen zu können, müssen mehrere Schritte durchgeführt werden. Der erste Schritt ist die Umkonvertierung der TrueType-Fonts in afm- und tfm-files. Weitere

Schritte sind unter anderem das Editieren von map- und fd-files.[7].
Die Nutzung von TrueType-Fonts ist mit Arbeit verbunden. Außerdem kostet es zusätzliche Verwaltungsarbeit, da sichergestellt werden muss, dass auf jedem System, auf dem das Desktop-Publishing eingesetzt werden soll, alle benötigten Dateien für den Einsatz eigener TrueType-Fonts zur Verfügung stehen.

Bei Grafiken ist LaTeX für die Arbeit mit dem EPS-Format (Encapsulated PostScript) konzipiert worden. Seit Bestehen des Programms 'pdfLaTeX' können darüber hinaus Grafiken in den Formaten JPEG, PNG und PDF ohne Qualitätsverlust eingebunden werden. 'pdfLaTeX' startet LaTeX im PDF-Modus und ermöglicht so die direkte Erzeugung von PDF-Dokumenten aus LaTeX-Code [WIK07, pdfLatex]. Zudem unterstützt sie alle Eigenschaften des PDF-Formats.
Grafiken können nun mit Hilfe des packages *graphicx* mit dem Befehl *\includegraphics* eingebunden werden. Diesem Befehl können auch Parameter übergeben werden, um das Bild, z.B. in Größe oder Drehung, anzupassen.

Der Funktionsumfang zum Erstellen von Rahmen und Strichen ist in LaTeX sehr groß. Wie einfach das Erstellen von Trennbalken in Kopf- und Fußzeile ist wurde bereits auf Seite 29 erwähnt. Auch das Erstellen von Horizontalen Balken innerhalb des Inhaltsbereichs ist nicht sonderlich kompliziert. Mit dem Befehl:

```
\rule{16cm}{1mm}
```

erhält man bereits einen horizontalen Balken von 16cm Breite und 1mm Höhe.

LaTeX's Verbindung zu XML

Für die Arbeit mit XML-Daten gibt es für LaTeX das Paket 'xmltex'. 'xmltex' ist ein nichtvalidierender XML-Parser und ermöglicht die Konvertierung von XML-Daten in ein LaTeX-Dokument.
Die Beschreibung, wie xmltex mit den Elementen des XML-Dokuments umgehen soll, erfolgt in sogenannten Paketdateien (Endung '.xmt'), die von 'xmltex' über eine Konfigurationsdatei(Endung '.cfg') geladen werden. Diese Paketdateien sind das Bindeglied zwischen XML und LaTeX und bestimmen den Aufbau des LaTeX-Dokuments. Ein Beispiel zur Definition des Elementes *<emph>* (fettgedruckter Text) über 'xmltex':

```
XML:    <emph>Fettgedruckter Text</emph>

xmltex: \XMLelement{emph}{}{\xmlgrab}{\textbf{#1}}

LaTeX:  \textbf{Fettgedruckter Text}
```

Elemente werden also mit der Anweisung:

```
\XMLelement{element}{attribute}{begincode}{endcode}
```

und Attribute mit der Anweisung:

```
\XMLattribute{attribute-name}{befehl}{standardwert}
```

[7] Eine genaue Anleitung findet sich unter: http://www.radamir.com/tex/ttf-tex.htm

beschrieben, wobei „XMLattribute“ nur in „XMLelement“ benutzt werden darf. Ein Beispiel für den Umgang mit Attributen ist wie folgt dargestellt:

```
XML:    <dokument typ="scrartcl">Text</dokument>

xmltex: \XMLelement{dokument} {
          \XMLattribute{typ}{\doctyp}{article}
        } { \documentclass{\doctyp}
            \begin{document}
          } {\end{document}}

LaTeX:  \documentclass{scrartcl}
        \begin{document}
           Text
        \end{document}}
```

Umgang mit Attributen über das Paket ’xmltex’ (Quelle: [Rad07])

Wie hier zu sehen ist, ist der Aufbau eines ’xmltex’-Pakets recht simpel. Die Syntax von LaTeX ist aber etwas ungewöhnlich und dadurch schwerer lesbar als z.B. bei XML-Dokumenten. Dies lässt den Code ziemlich kompliziert erscheinen und schadet gleichzeitig der Übersichtlichkeit.
Ein weiteres Manko ist der Funktionsumfang von ’xmltex’. Das Gegenstück zu ’xmltex’ zur Transformation von XML-Daten wäre das vom W3C spezifizierte XSLT. Über XSLT gibt es über XPath eine gute Möglichkeit jeden möglichen Knotenpunkt im XML-Dokument zu erreichen. Diese Möglichkeiten sind bei ’xmltex’ sehr beschränkt, da sie über keine vergleichbaren Funktionen wie XPath verfügt. Der Funktionsumfang von ’xmltex’ kommt auch aus diesem Grunde nicht an XSLT heran.

Schlussfolgerung

Insgesamt ist das Layoutverhalten bei LaTeX-Dokumenten etwas umständlich, da man sich an die vorhandenen Pakete halten muss und dadurch individuell gestaltete Layouts schwer realisierbar sind.
Hinzu kommt, dass für anspruchsvollere Dokumente oft Pakete geladen werden müssen, die nicht im Standardpaket von LaTeX enthalten sind. So kommt es vor, dass Pakete auch mal nachinstalliert werden müssen.
LaTeX eignet sich hauptsächlich für Dokumente, bei denen es keine strengen Layoutvorgaben gibt. In solchen Anwendungsgebieten ist LaTeX eine große Hilfestellung zum Erstellen von umfangreichen, strukturierten Dokumenten, wie z.B. Diplomarbeiten, die in Kapitel- und Unterkapitel aufgeteilt sind.

4.2.2. XSL-FO

XSL-FO, oder abgekürzt FO, ist ein Teilkonzept des XSL-Konzepts (siehe Kapitel 2.3.2). Es ist das Zielformat einer Konvertierung von XML-Daten über die Transformationssprache XSLT.
FO Dokumente basieren auf der XML-Syntax und beschreiben unter anderem den Seitenaufbau, die Anordnung der Seiten und den Zeilenumbruch und werden von sogenannten FO-Prozessoren in ein entsprechendes Layout gebracht.

Die Arbeit mit XSL-FO

Bei XSL-FO sind standardmäßig keine Template-Klassen mitgeliefert, so dass das Layout der Seiten von Grund auf programmiert werden muss. Dies ist aufwändiger, hat aber den Vorteil dass das Layout leichter individuell zu gestalten ist, was ja im Fall der vorliegenden Dokumente, welche ein sehr eigenes Layout haben, gefordert ist.
Bei der Arbeit mit XSL-FO sollte zunächst überlegt werden, wie die Anordnung der Seiten aufgebaut werden soll. Für die einzelnen Seiten im Dokument werden Seitenvorlagen erstellt (Aufbau der Seiten mit Abständen etc.), welche dann im Dokumentenfluss verwendet werden. Für die Dokumente OM-A und FSM müssten Seitenvorlagen für eine linke, eine rechte und eine leere (linke) Seite definiert werden. Das Dokument würde mit einer rechten Seite beginnen und abwechselnd eine linke und eine rechte Seite generieren. Die leere Seite soll dann eingesetzt werden, wenn ein Kapitel auf einer rechten Seite endet (siehe Kapitel 4.1, Seite 19). Diese Logik wird am Anfang einer FO-Datei beschrieben.

Aufbau der FO-Datei

Eine FO-Datei ist ein wohlgeformtes XML-Dokument. Der Grundaufbau von FO-Dateien ist immer derselbe. Das Wurzelelement heißt *<fo:root>*. Kindelemente sind ein *<fo:layout-master-set>*, optional eine *<fo:declaration>* und mindestens eine *<fo:page-sequence>*.

Innerhalb des Elements *<fo:layout-master-set>* werden die Seitenvorlagen und die Logik zur Anordnung der Seiten beschrieben. Hier kann mit Hilfe des Elements *<fo:page-sequence-master>* die Anordnung der bereits erstellten Seitenvorlagen bestimmt werden.
Im folgenden Beispielcode wird eine rechte Seitenvorlage den ungeraden Seiten und eine linke Seitenvorlage den geraden Seiten zugeordnet. Zusätzlich wird eine leere Seitenvorlage als *blank page* definiert. Wie diese *blank page* in das Dokument gelangt wird später erläutert.

```
<fo:page-sequence-master master-name="chapter">
  <fo:repeatable-page-master-alternatives>
    <fo:conditional-page-master-reference
                        master-reference="right-page"
                        odd-or-even="odd"
                        blank-or-not-blank="not-blank"/>
    <fo:conditional-page-master-reference
                        master-reference="left-page"
                        odd-or-even="even"
                        blank-or-not-blank="not-blank"/>
    <fo:conditional-page-master-reference
                        master-reference="left-blank-page"
```

```
                    blank-or-not-blank="blank" />
  </fo:repeatable-page-master-alternatives>
</fo:page-sequence-master>
```

XSL-FO Code zur Anordnung der Seiten (Quelle: Eigene Beschreibung)

Mit dem *<fo:declaration>*-Element können Farbprofile angelegt werden. Diese sind aber optional und werden selten eingesetzt, da sie von den bisher verfügbaren Formatierprogrammen nicht unterstützt werden.

Die *<fo:page-sequence>*-Elemente beschreiben den eigentlichen Inhalt eines Dokuments. Dieses Element referenziert eine Seitenvorlage oder einen *<page-sequence-master>* mit einer Abfolge von Seitenvorlagen und füllt dann die verschiedenen Bereiche einer Seite mit Inhalt.

Layout der Seiten

XSL-FO teilt eine Seite grundsätzlich in bis zu fünf verschiedene Regionen ein. Einen Bereich für den eigentlichen Inhalt der Seite (*body-region*) und vier weitere Regionen für Kopf-, Fußzeilen und Randbemerkungen. Anfangs war für den Inhaltsbereich nur eine *body-region* vorgesehen. Seit der XSL-Version 1.1 können für den Inhaltsbereich neuerdings mehrere *body-regions* definiert werden. Dies schafft eine noch stärkere Individualität bei der Layout-Gestaltung der Seiten. Die Größe und die Aufteilung der Regionen geschieht im Element *<fo:simple-page-master>*. Eine Seite bekommt Seitenränder mit dem Attribut *margin*. Der Kopfbereich beginnt am oberen Seitenrand und bekommt mit dem *extent* Attribut eine Definition der Ausdehnung dieses Bereichs. Der Fußbereich bekommt ebenfalls mit dem *extent* Attribut eine Größenangabe und richtet sich am unteren Seitenrand aus. Der Inhaltsbereich (Body) wird wiederum mit den *margin* Attributen vom Seitenrand aus bemessen.
Folgender Code zeigt beispielhaft, wie der Aufbau einer linken Seite des OM-A in XSL-FO aussehen könnte:

```
<fo:simple-page-master
            master-name="left-page"
            page-height="297mm"
            page-width="210mm"
            margin-top="7mm"
            margin-bottom="5mm"
            margin-left="9mm"
            margin-right="8mm">
<fo:region-body
            margin-top="30mm"
            margin-bottom="9mm"
            margin-left="3mm"
            margin-right="15mm" />
<fo:region-before
            region-name="header-l"
            extent="26mm" />
<fo:region-after
            region-name="footer-l"
            extent="8mm" />
</fo:simple-page-master>
```

XSL-FO Code für eine linke Seite des OM-A (Quelle: Eigene Beschreibung)

Nach Beschreibung der Seitenvorlagen in XSL-FO mit den Werten des vorangegangenen Listings, ergibt sich folgende Aufteilung des Seitenlayouts:

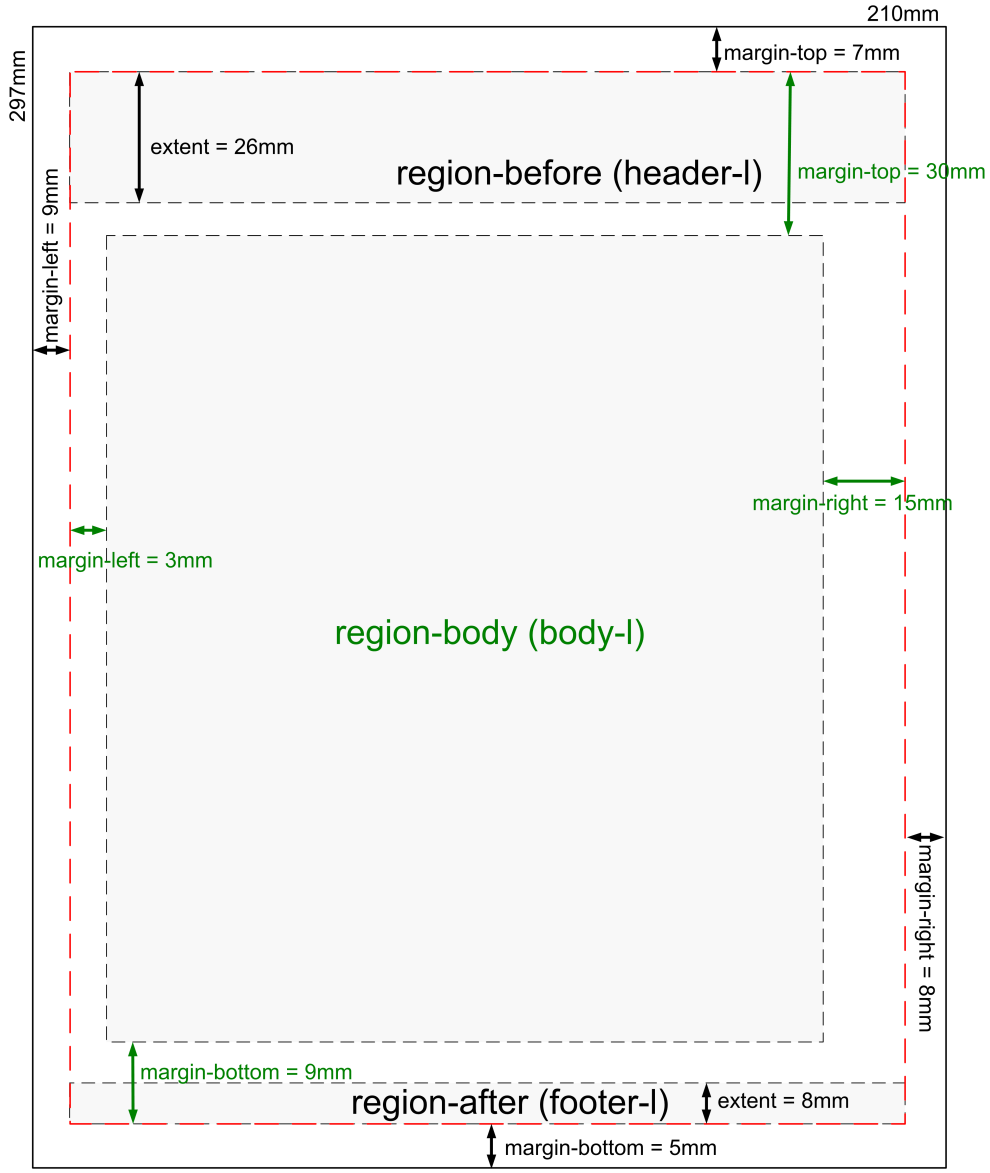


Abbildung 4.9.: XSL-FO Aufbau einer linken Seite des Dokuments OM-A (Quelle: Eigene Darstellung)

Nachdem die Seitenvorlagen definiert sind, können die Bereiche mit Inhalt gefüllt werden.

Kopf- und Fußbereich

Die typografischen Mittel die in XSL-FO verfügbar sind, können in allen Regionen einer Seite uneingeschränkt genutzt werden. Es gibt also keinen Unterschied im Aufbau des Inhalts

zwischen den verschiedenen Regionen. Dadurch fällt der Aufbau des Kopf- und Fußbereichs nicht sonderlich schwer. Zum Aufbau dieser Bereiche wird oft das Tabellenmodell genutzt, da hiermit die verschiedenen Informationen, die in einem Kopf- oder Fußbereich enthalten sind, gut aufgeteilt werden können.
Zur Unterstützung von typischen Funktionen eines Kopfbereichs gibt es in XSL-FO das Element *<fo:page-number>* zur Anzeige der aktuellen Seitenzahl und das Element *<fo:retrieve-marker>* für lebende Kolumnentitel (Kapitel- oder *section*-Name).
Mit *<fo:marker>* können beispielsweise Titel von *<chapter>*- oder *<section>*-Elementen markiert und über *<fo:retrieve-marker>* ausgegeben werden. *<fo:retrieve-marker>* sucht ein zugehöriges *<fo:marker>*-Element auf der aktuellen oder den vorhergehenden Seiten.

Für jede vorhandene Seitenvorlage muss der Inhalt für den Kopf- und den Fußbereich explizit definiert werden.
Das Layout für diese Bereiche kann dadurch komplett individuell gestaltet werden.

Der Inhalt

Der Inhalt wird in eine *<fo:page-sequence>* geschrieben. Einer *page-sequence* kann man unter anderem folgende Attribute geben:

- *force-page-count*:

  Kann beispielsweise mit dem Wert *end-on-odd* belegt werden um die *page-sequence* zwangsweise auf einer ungeraden Seite enden zu lassen. Hier kommt eventuell die bereits erwähnte *blank page* ins Spiel. Weitere mögliche Werte sind:
  auto, even, odd, end-on-even, no-force.

- *initial-page-number*:

  Dieses Attribut ist für die automatische Seitennummerierung wichtig. Mit dem Standardwert *auto* beginnt die erste Sequenz mit der Seitennummer 1. Folgende Sequenzen führen die Nummerierung des Vorgängers fort. Es kann auch eine Zahl eingegeben werden. Somit wird z.B. erreicht, dass jede *page-sequence* bei der Seitennummer 1 beginnt. Weitere mögliche Werte sind:
  auto-odd, auto-even.

- *language*:

  Definiert die Sprache einer *page-sequence* und hat Auswirkung auf die automatische Worttrennung wenn ein Text über den Seitenrand hinaus geht. Als Wert wird hier ein 2-letter Code nach [ISO639] oder ein 3-letter Code nach [ISO639-2] erwartet.

Der Inhaltsbereich (Body) des Dokuments bekommt ein Element *<fo:flow>*, in den der komplette Inhalt hinein „fließt“. Die Verteilung des Inhalts auf mehrere Seiten geschieht automatisch durch den FO-Prozessor, der die FO-Datei lädt.
Die restlichen Bereiche (Kopf- und Fußzeilen) bekommen mit dem Element *<fo:static-content>* einen statischen Inhalt. Ein statischer Inhalt kann entweder ein fest vorgegebener Text oder ein generierter Inhalt, wie z.B. eine Seitennummerierung, sein.

Ein zentrales Konzept in XSL-FO zur Beschreibung der Inhalte ist das Element *<fo:block>*. Er wird benutzt, um alle möglichen Elemente, wie Tabellen, Bilder, Listen, Absätze usw. in rechteckige Anzeigebereiche einzusetzen. Mit der Aneinanderreihung der Blöcke füllt sich die Seite.

Blöcke, die als Ganzes nicht mehr in den Seitenraum passen, werden nach den gegebenen Umbruchregeln zwischen den Seiten geteilt [KP04, Kapitel 6.3].
Diesen Blöcken können Attribute gegeben werden, um z.B. Abstände zum Bereichsrand, Einzüge oder Schriftgrößen zu definieren.

Für Tabellen stellt XSL-FO ein eigenes Konzept zur Verfügung. Der Aufbau der Tabellen ist dem CALS Tabellenmodell ähnlich. Wie beim CALS Tabellenmodell können die Spalten einer Tabelle spezifiziert werden (*<fo:table-column>* parallel zu *colspec* bei CALS).
Es existieren Elemente für einen Kopf- und einen Fußbereich der Tabelle (*<fo:table-header>* und *<fo:table-footer>*), die bei einem eventuellen Seitenumbruch automatisch wiederholt werden.
Die Struktur der FO-Tabellen soll anhand der folgenden Grafik verdeutlicht werden.

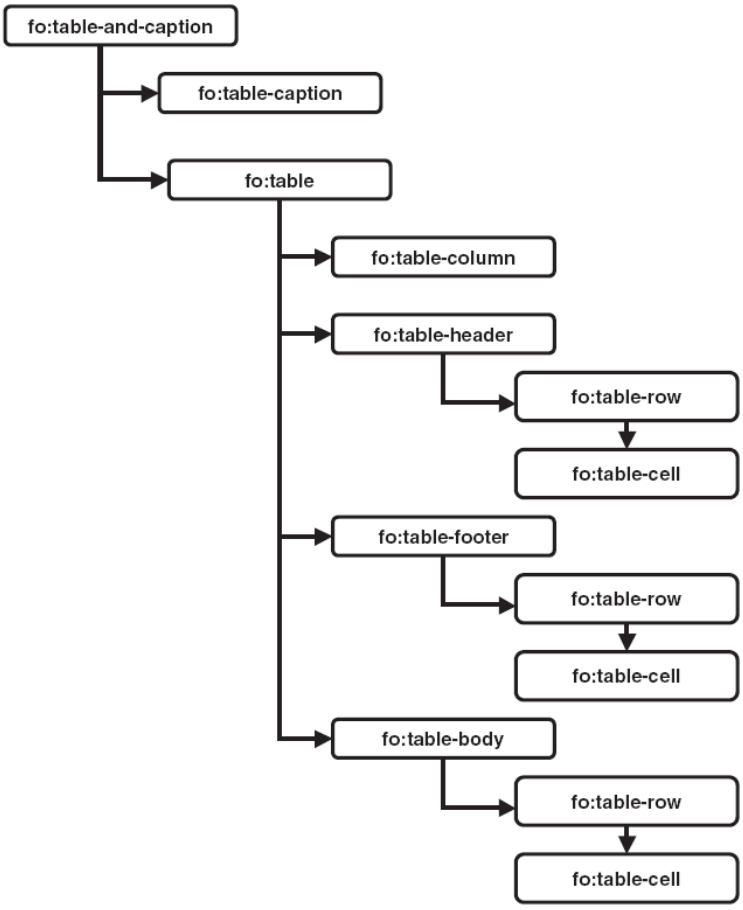


Abbildung 4.10.: Struktur von FO-Tabellen (Quelle: [KP04])

An der Abbildung 4.10 ist die Ähnlichkeit zum CALS Tabellenmodell erkennbar. Auch in der Literatur wird beschrieben, dass sich Tabellen mit der CALS-Tabellenstruktur gut als Ausgangspunkt für eine Umsetzung in XSL-FO eignet (siehe z.B. [KP04, Kapitel 6.10]).
In XSL-FO können den Tabellen auch eine Reihe an Eigenschaften gegeben werden, um das Aussehen oder das Verhalten zu definieren. Im Folgenden ist ein Auszug der Attribute für *<fo:table>* dargestellt:

- *margin*- und *padding*-Attribute:

  Mit diesen Attributen können Abstände der Tabelle zum Seitenrand (oder des umgebenden Blocks) definiert werden.

- *border*-Attribute:

  Mit den border-Attributen kann das Aussehen der Tabellenrahmen modifiziert werden.

- *table-omit-footer-at-break*:

  Hiermit wird festgelegt, ob der Fußbereich der Tabelle auf den Folgeseiten wiederholt werden soll.

Die *padding*-Attribute gelten auch für die anderen Tabellenelemente. Für *<fo:table-cell>* können sie beispielsweise verwendet werden um die Abstände der Tabelleninhalte zum Zellenrand zu definieren.

Ein zweispaltiges Layout kann erreicht werden, indem man den Body der Seitenvorlagen mit dem Attribut: 'column-count ="2"' versieht oder indem man mit mehreren *body-regions* arbeitet.
Werden die *body-regions* zweispaltig aufgebaut ('column-count ="2"'), so kann diese *body-region* später, wie vorgesehen, zweispaltig eingesetzt werden oder sie kann durch das Attribut *span*, welches beispielsweise im Element *<fo:block>* anwendbar ist, spaltenübergreifend genutzt werden. Für das Attribut *span* können aber nur die Werte 'none' oder 'all' definiert werden. Mit der Anwendung dieser FO Eigenschaft kann ein Dokument mit ein- und zweispaltigem Layout erzielt werden.
Eine andere Möglichkeit ist der Einsatz mehrerer *body-regions*, was seit der XSL-Version 1.1 ermöglicht wird. Erstellt man eine linke und eine rechte *body-region*, so kann der Inhalt von der linken in die rechte *body-region* „fließen". Hierbei können die *body-regions* nicht überspannt werden (kein *span* möglich). D.h., dass für Dokumente mit ein- und zweispaltigem Layout verschiedene Seitenvorlagen erstellt werden müssten.

TrueType-Fonts und grafische Elemente

Schriftarten werden in FO mit dem Attribut *font-family* dem entsprechenden Abschnitt übergeben. Dieses Attribut kann bereits im *<fo:root>*, in einzelnen *<fo:block>*, *<fo:page-number>* oder weiteren Elementen definiert werden. Der Attributwert wird vererbt. D.h., wenn in *<fo:root>* bereits eine Schriftart definiert ist, wird diese Schriftart auch in allen Blöcken, die dieses Attribut nicht überschreiben, verwendet.
Die zur Verfügung stehenden Schriftarten hängen vom jeweiligen FO Prozessor, welcher am Ende aus einer FO-Datei eine Ausgabe in PDF generiert, ab. Es stehen standardmäßig drei Schriftarten bei allen FO Prozessoren zur Verfügung. Dies sind die Schriftarten 'Times Roman', 'Helvetica' und 'Courier'.
Die meisten FO Prozessoren unterstützen TrueType- und PostScript Type 1-Fonts. Nicht vorhandene Schriftarten müssen eventuell vorerst auf dem System installiert (bei Windows-Systemen genügt die Ablage der ttf-Dateien in dem '/Windows/Fonts' Ordner) und anschliessend im FO Prozessor konfiguriert werden.
Der Name der Schriftart entspricht dem Namen der Schriftartendatei (ohne Endung). Ein Beispiel:
Existiert eine Schriftart mit dem Namen 'Lufthansa.ttf', so kann diese Schriftart in einem *<fo:block>* mit folgendem Ausdruck angewendet werden:

```
<fo:block font-family='Lufthansa'>
```

Die FO-Prozessoren unterstützen auch eine Einbettung der verwendeten Schriftarten in die PDF-Datei, um das Dokument portabel zu machen. Wird diese Funktion nicht verwendet, so kann die PDF-Datei nur auf Systemen korrekt angezeigt werden, welche die verwendeten Schriftarten installiert haben.

Bei der Einbindung von externen Grafiken hängt die Auswahl der verfügbaren Formate wieder vom jeweiligen FO Prozessor ab. Standardmäßig kann aber jeder FO Prozessor mit den Formaten PNG, JPEG, GIF, SVG und EPS umgehen.

Zur Erzeugung von horizontalen Linien kann das Element *<fo:leader>* innerhalb von Blöcken verwendet werden. Mit *<fo:leader>* lassen sich horizontale Linien in verschiedenen Mustern (punktiert, liniert, gestrichelt) und selbst definierter Dicke (Vorgabewert 1pt) erstellen. Zusätzlich lässt sich die Farbe und die Länge der Linie bestimmen. Mit dieser Funktion lassen sich recht einfach Trennbalken für Kopf- und Fußbereich erstellen.

Schlussfolgerung

Wer sich viel mit der XML-Sprache auseinandersetzt findet sich schnell in XSL-FO zurecht. XSL-FO ist recht einfach zu verstehen und dennoch sehr umfangreich in seinen Layoutmöglichkeiten. Mit XSL-FO lassen sich ohne viel Aufwand individuell gestaltete Layouts verwirklichen. Trotzdem muss das Dokument eine Regelmäßigkeit vorweisen. Weist das Dokument viele unregelmäßige Eigenschaften auf (z.B. Dokumente mit unregelmäßigem Seitenaufbau), wird die Umsetzung in FO etwas aufwändiger. Auch bei Dokumenten die Elemente mit komplizierter Logik haben (z.B. Entscheidungsbäume) stößt man mit der XSLT Transformation zu XSL-FO an seine Grenzen, da XSLT keine vollwertige Programmiersprache ist.
Die typografischen Mittel, die XSL-FO zur Verfügung stellt, sind sehr umfangreich. Über das Definieren von Einrückungen, Unterstreichungen, Überstreichungen, Zeichenabständen, Hoch- und Tiefstellung von Buchstaben und horizontalen Linien gibt es noch viele weitere typografische Mittel (siehe [KP04, Kapitel 6.6]).
Durch die Arbeit mit Blöcken ist die Positionierung der Elemente für den Anwender sehr komfortabel. Der Anwender bekommt eine gute Vorstellung der Raumaufteilung seiner Dokumentenseite. Dadurch lässt sich vor allem die Gestaltung von Kopf- und Fußbereich, in denen die verschiedenen Elemente einen bestimmten Platz haben, leicht umsetzen.
XSL-FO ist durch die Vielfalt der Layoutmöglichkeiten optimal für die Umsetzung umfangreicher und weniger komplexen Dokumente geeignet.

4.2.3. Adobe® FrameMaker®

Adobe® FrameMaker®, im Folgenden FrameMaker, ist ein umfangreiches Tool, welches die Vorteile einer anwenderfreundlichen Textverarbeitung mit den Stärken von XML kombiniert. In FrameMaker ist sowohl die Erstellung von unstrukturierten Dokumenten, wie man sie z.B. auch mit Word erstellen würde, als auch die Erstellung von strukturierten Dokumenten, wie in unserem Falle aus XML-Daten, möglich. Im weiteren wird nur die Anwendung mit strukturierten Dokumenten betrachtet.

Die Arbeit mit FrameMaker®

FrameMaker stellt zwei Ansichten zur Erstellung der Printpublikation zur Verfügung. Eine Layoutansicht und eine Strukturansicht. In der Layoutansicht kann das Layout der Seiten anhand der komfortablen Editierfunktionen, wie sie auch von anderen Textverarbeitungs- oder Zeichenprogrammen bekannt sind, bestimmt werden. In der Strukturansicht ist der strukturelle Aufbau eines Dokuments abgebildet. Dort kann den verschiedenen Elementen des Dokuments Formatierregeln zugeteilt werden.
In der Layoutansicht arbeitet man mit Vorgabeseiten, in denen das grundsätzliche Layout der Dokumentseiten gestaltet wird. Dies funktioniert mit Werkzeugen wie Textblöcken oder Linien, wie sie auch aus anderen Zeichentools bekannt sind (siehe Abbildung 4.11). Die Gestaltung der Vorgabeseiten wird dadurch sehr elegant gelöst und es kann auf einfache Weise ein beliebig individuelles Layout erstellt werden.
Bei Erstellung von Büchern sind standardmäßig zwei Vorgabeseiten, für linke und rechte Seite, vorgegeben, sowie eine Kopfzeile, ein Inhaltsbereich (Body) und eine Fußzeile. Textrahmen und grafische Elemente können beliebig platziert werden.

In den Kopf- und Fußzeilen werden von FrameMaker laufende Kolumnentitel, wie z.B. Seitennummern oder Kapitelüberschriften, unterstützt. Das Definieren von kontextbehafteten Überschriften in der Kopf- oder Fußzeile wird in FrameMaker nicht so ohne weiteres unterstützt. Dadurch ist der Spielraum bei der Inhaltsgestaltung von Kopf- und Fußzeilen etwas eingeschränkt.

Ein Zusatz von FrameMaker ist die vorhandene API, die in der Programmiersprache C zur Verfügung gestellt wird. Sie stellt ein sehr großes Spektrum an Funktionen zur Verfügung. Mit ihrer Anwendung kann theoretisch jedes beliebige Layout erzielt werden. Die Nutzung der API kostet den Anwender etwas mehr Arbeit, macht aber FrameMaker auch für komplexere Layout-Anforderungen einsetzbar.
Ein weiteres Konzept in FrameMaker ist die Arbeit mit Referenzseiten. In den Referenzseiten werden immer wieder auftauchende Seiteninhalte oder einzelne Objekte (Grafiken, Zeichnungen, etc.) abgelegt, die dann im Dokumentenfluss bei Gebrauch referenziert werden können. So können z.B. für die *warning*-, *caution*- und *note*-Elemente der FSM-Dokumente die Grafiken und die horizontalen Trennbalken in den Referenzseiten abgelegt werden, um sie später in der Behandlung der XML-Elemente aufzurufen.
Die Standardfunktionen von FrameMaker für Bücher unterstützen zudem das Generieren von einer geraden Anzahl an Seiten, so dass eventuell am Ende eines Dokuments eine zusätzliche leere Seite generiert wird. Möchte man diese Seite nicht leer lassen sondern auch mit Inhalt

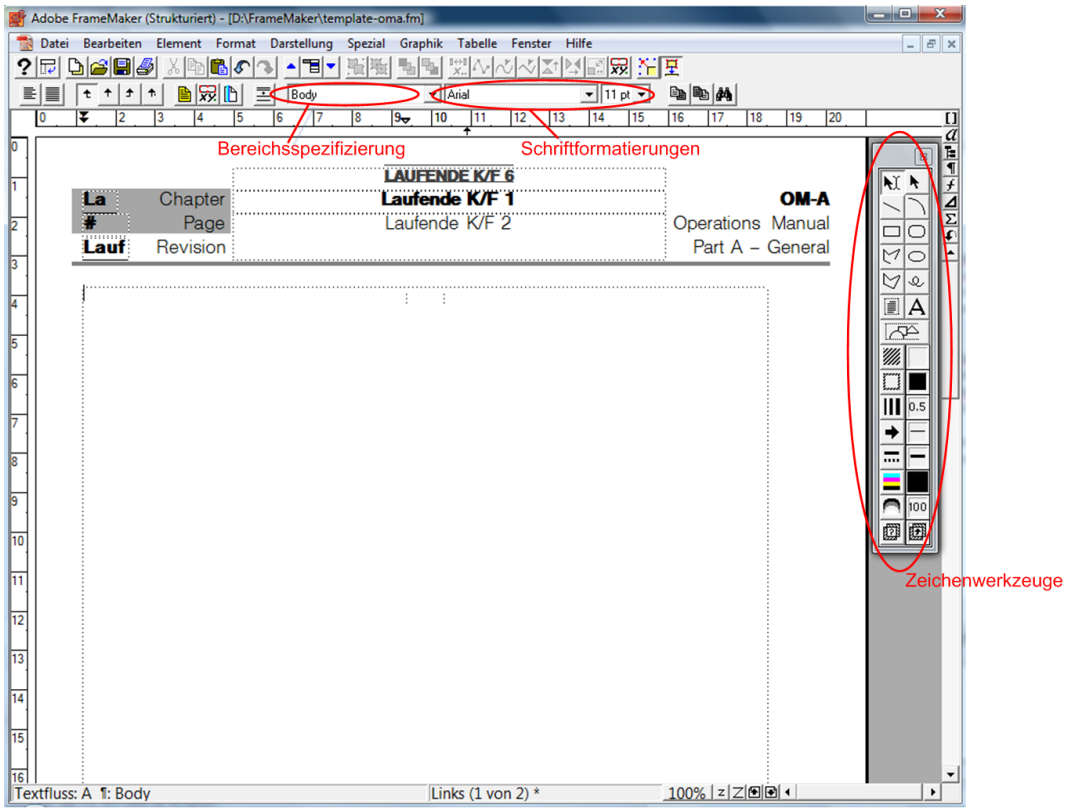


Abbildung 4.11.: Layoutansicht in FrameMaker (Quelle: Eigener Screenshot des FrameMaker Editors)

beschreiben (Stichwort: 'Intentionally Left Blank'), so kann hierfür über die mitgelieferte FrameMaker API eine Referenzseite aufgerufen werden.
Eine weitere Funktion, die FrameMaker bietet, ist die Möglichkeit jeden Zugriff im Editor rückgängig zu machen. Jeder Schritt wird in einer Verlaufsliste (wie es sie z.B. auch in Adobe® Photoshop® gibt) festgehalten und so kann jeder Verarbeitungsschritt nachvollzogen und verworfen werden.

Der strukturierte Aufbau

Erst seit 2002 unterstützt Adobe FrameMaker mit der Version 7.0 die Arbeit mit XML-Daten. Für die Erstellung eines Layouts kann mit Adobe FrameMaker eine DTD geladen werden und (basierend auf dieser DTD) eine Layoutbeschreibung erstellt werden.
Diese Layoutbeschreibung findet bei strukturierten Dokumenten in der sogenannten EDD-Datei (Element Definition Document) statt. Die EDD wird nach Laden einer DTD automatisch erstellt. Darin enthalten ist eine Abbildung aller Elemente und zugehöriger Attribute.
In der EDD kann dann das Verhalten der XML Elemente spezifiziert werden. In der Strukturansicht navigiert man mit der Maus durch die verschiedenen Elemente und kann mit Hilfe des Elementkatalogs Layoutregeln definieren. Abbildung 4.12 zeigt einen Ausschnitt aus einer Strukturansicht und dem zugehörigen Elementkatalog. Je nachdem welchen Punkt man in dieser Baumstruktur markiert, zeigt der Elementkatalog andere Optionen an, die eingefügt werden können. In der Abbildung befindet man sich z.B. im Element *<para>* und steht am Punkt „Absatzabstand“. Hier können im Elementkatalog dementsprechend die Formatierregeln

„AbstandOben“ (hier bereits vergeben), „AbstandUnten“ und „AbstandUntenÄndern“ ausgewählt werden.
Das Navigieren in dieser Strukturansicht ist sehr komfortabel. Elemente können während der Arbeit zugeklappt werden, um die Arbeit übersichtlich zu halten. Durch den Elementkatalog können die grundlegenden Formatierregeln schnell vergeben werden. Bei spezifischeren Regeln muss Anfangs aber oft in den vorhandenen Dokumentationen nach Lösungen gesucht werden, da die Beschreibung von logischen Regeln (z.B.: „wie beschreibt man die erste section eines *chapters*“ oder „wie erkenne ich ein vorhergehendes *<para>*-Element“) eine FrameMaker-eigene Syntax vorweist und somit etwas gewöhnungsbedürftig ist.

Für die Elemente können kontextfreie Regeln sowie kontextabhängige Regeln definiert werden. In Abbildung 4.12 ist eine kontextfreie Regel (in der Strukturansicht: *AlleKontextregeln*) und eine kontextabhängige Regel (in der Strukturansicht: *Kontextregel*) zu sehen. Hier erhält das Element *<para>* standardmäßig die Schriftart ’Arial’ mit der Schriftgröße 9pt. Wenn ein vorhergehendes *<para>*-Element existiert soll das Element zusätzlich einen Abstand nach oben bekommen.

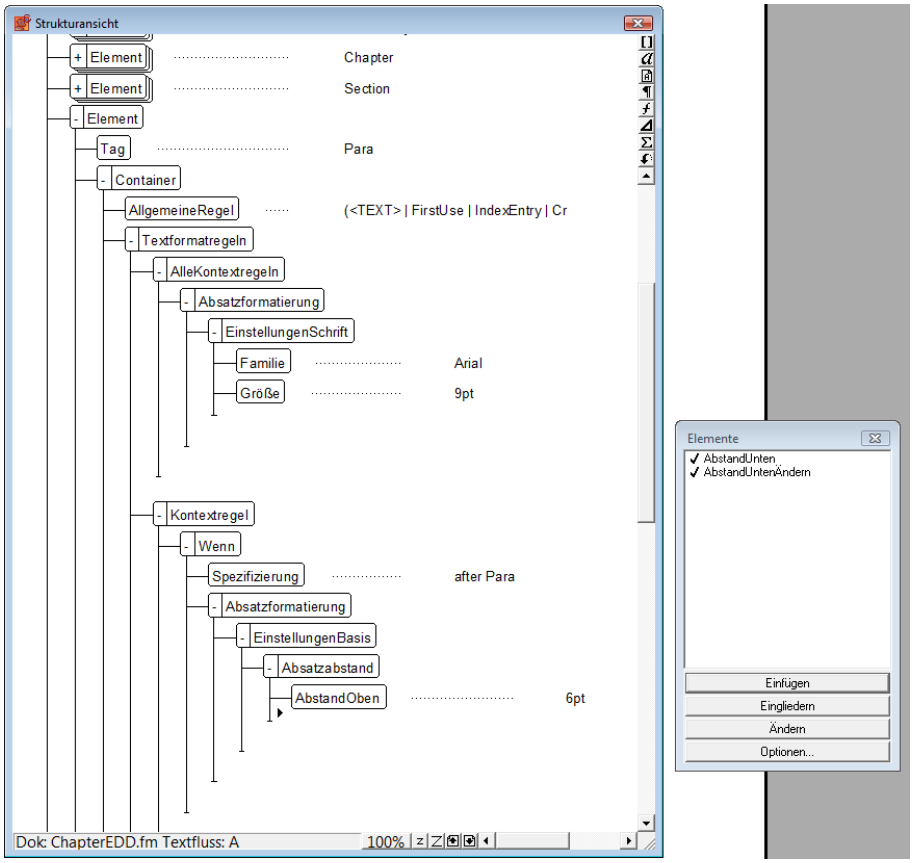


Abbildung 4.12.: Strukturansicht in FrameMaker: Kontextregel (Quelle: Eigener Screenshot des FrameMaker Editors

Der Body der Vorgabeseiten kann auch ein mehrspaltiges Layout bekommen. Um im Dokumentenfluss abwechselnd ein einspaltiges und ein zweispaltiges Layout zu erhalten, kann in der EDD für verschiedene Elemente eine spaltenübergreifende Behandlung definiert werden. Definiert man z.B. in den Vorgabeseiten zwei Spalten im Body, so müsste die Standardregel für alle Elemente spaltenübergreifend sein. Nur der Index würde die Spaltenaufteilung einhalten.
Layoutregeln in der EDD werden vererbt. Wird die spaltenübergreifende Behandlung beispielsweise im *root*-Element vergeben, so gilt sie standardmäßig für alle Elemente, solange sie

nicht überschrieben wird. Durch die Vererbung von Layouteigenschaften kann es vorkommen, dass Abstände mit Addition oder Subtraktion angepasst werden müssen. Dies kann auch zu Unübersichtlichkeit oder Verwirrungen führen, vor allem für außenstehende Entwickler, die erst nach Umsetzung die Layoutbeschreibung einsehen.

Für viele typische Elemente eines Dokuments, wie z.B. Kapitel, Tabellen, Grafiken etc., sind in FrameMaker bereits Standarddefinitionen vorhanden, so dass nicht für jedes Element von Grund auf das Verhalten und das Layout bestimmt werden muss, was z.B. bei Tabellen sehr umständlich wäre. Um diese Standardelemente anzuwenden, müssen die Elemente aus der DTD 'gemappt' werden. Dies passiert in den 'Lese- und Schreibregeln' einer strukturierten FrameMaker Applikation. Beim 'mapping' von Tabellen sehen die Regeln wie folgt aus:

```
element "table"
{
  is fm table element "Table";
  attribute "colsep" is fm property column ruling;
  attribute "rowsep" is fm property row ruling;
  attribute "tablestyle" is fm property table format;
}

element "tbody" is fm table body element "TableBody";
element "thead" is fm table heading element "TableHead";
element "tfoot" is fm table footing element "TableFoot";
element "row" is fm table row element "TableRow";
element "entry" is fm table cell element "TableCell";
```

Mapping von XML-Elementen zu strukturierten FrameMaker Objekten (Quelle: [Ado07b])

Das Standardverhalten der gemappten Elemente kann im Nachhinein in der EDD verändert werden.
Bei Tabellen arbeitet das Programm standardmäßig mit dem CALS Tabellenmodell. Tabellen, die in der XML-Datei bereits im CALS-Format vorliegen, können direkt übernommen werden und es braucht keiner weiteren Übersetzung.
FrameMaker unterstützt TrueType, OpenType und Type 1 Fonts. TrueType-Fonts, die in Windows installiert sind, stehen im Editor zur Verfügung und können problemlos eingesetzt werden.

Ist die Layoutbeschreibung komplett, kann sie schließlich mit dem Template (den Vorgabeseiten) verknüpft und anschließend die XML-Datei geladen werden. Der Inhalt der XML-Datei verteilt sich dann abwechselnd auf die Vorgabeseiten.
Zu große Datenmengen führen bei FrameMaker zu Problemen. Entweder wird die Arbeit in den FrameMaker Dokumenten sehr langsam oder es kommt ganz und gar zu einem Stillstand. Daher, und auch aus anderen Gründen, sollte bei der Erstellung von FrameMaker-Dokumenten nicht mit dem kompletten Buch gearbeitet werden, sondern mit den einzelnen Kapiteln. Dafür benötigt es einer Vorverarbeitung der XML-Dateien. Sind die einzelnen Kapitel als FrameMaker-Dokumente gespeichert, können sie über die 'Masterbuch'-Funktion zu einem Gesamtbuch zusammengebunden werden. Das Masterbuch definiert welche FrameMaker-Dateien zusammengehören und in welcher Reihenfolge sie im Buch auftauchen. Mit diesen Informationen kann dann eine PDF-Ausgabe für das komplette Buch erfolgen.
Wird eine Ausgabe erzeugt, kann diese im FrameMaker Editor vor dem Speichern in das PDF-Format eingesehen werden. In dieser Ansicht besteht zusätzlich die Möglichkeit das Ausgabedokument noch nachträglich manuell zu bearbeiten.

Schlussfolgerung

Zu Beginn der Arbeit mit FrameMaker bedarf es einer zeitintensiven Einarbeitung, da man sich vorerst mit den Konzepten und Werkzeugen vertraut machen muss, die etwas gewöhnungsbedürftig sind. Die Funktionen die angeboten werden sind aber sehr reichhaltig und die zusätzlichen Programmiermöglichkeiten durch die mitgelieferte API machen FrameMaker zu einem mächtigen Werkzeug, mit dem individuelle und auch kompliziertere Layout-Anforderungen bewältigt werden können.
Durch die vielen manuellen Schritte, die im FrameMaker Prozess getätigt werden müssen, erreicht man aber leider keine vollautomatische Verarbeitung der XML-Daten in das Zielformat PDF.
Insgesamt stellt FrameMaker ein sehr kompaktes Paket mit einem großen Funktionsumfang bereit, mit dem man nach einiger Einarbeitungszeit sehr schnell seine Layout-Anforderungen umsetzen kann.

4.2.4. Bewertung der DTP-Lösungen

Die wichtigsten Anforderungen für das zu erzeugende Layout der beiden Dokumente OM-A und FSM sind mit einer Bewertung der DTP-Lösungen in der Tabelle 4.1 noch einmal als Übersicht dargestellt.

	LaTeX	XSL-FO	Adobe® Framemaker®
individuelle Layoutgestaltung (Setzen von Abständen etc.)	-	++	++
verschiedenes Layout für linke und rechte Seite	+	++	++
Generieren von blank pages	++	++	o
Kopfzeilen- und Fußzeilenbehandlung	o	++	+
Verarbeitung von CALS-Tabellen	- -	++	++
zweispaltiges Layout für *index*	+	+	+
Nutzung von TrueType-Fonts	-	+	++
Nutzung von PNG-Grafiken	++	++	++
Erzeugen grafischer Elemente (horizontale Balken etc.)	++	+	++

++ sehr gute Unterstützung
+ gute Unterstützung
o möglich
- umständlich
- - sehr aufwendig/ nicht möglich

Tabelle 4.1.: Bewertung der DTP-Lösungen nach gegebenen Anforderungen (Quelle: Eigene Darstellung)

Die Auswahl aus den heute gegebenen DTP-Lösungen ist stark von den Anforderungen und Gegebenheiten des vorliegenden Systems abhängig. Jede DTP-Lösung hat seine Vor- und Nachteile und findet in verschiedenen Anwendungsfällen seine Berechtigung. Daher ist die Suche nach dem passenden Tool nicht eine Suche nach dem 'besten' Tool, sondern die Suche nach einem Tool, welches sich für die gegebenen Anforderungen am besten eignet.

Nach Untersuchung der verschiedenen Tools, die sich an den beschriebenen Anforderungen aus der Analyse messen mussten, stechen vor allem die Schwierigkeiten der Umsetzung mit LaTeX heraus. LaTeX wird hauptsächlich an Universitäten und Fachhochschulen eingesetzt, insbesondere im mathematisch-naturwissenschaftlichen Bereich, und bietet für diese Anwendungsfälle eine gute Unterstützung.
Durch die Layoutvorgaben von LaTeX, die sehr statisch sind, fällt das Erstellen von individuellen Layouteigenschaften sehr schwer. Da LaTeX zudem kein Gesamtpaket ist, sondern aus vielen einzelnen Paketen besteht, muss häufig in verschiedenen Dokumentationen recherchiert werden. Ein weiteres Problem bei der Umsetzung der gegebenen Anforderungen ist das Tabellenmodell, welches vom CALS Tabellenmodell komplett abweicht und eine Verarbeitung der Tabellen sehr erschwert.

Die DTP-Lösungen XSL-FO und FrameMaker werfen keine Probleme bei der Umsetzung der Anforderungen auf.
Ein Vorteil von XSL-FO gegenüber Adobe FrameMaker ist, dass bei Fertigstellung der Stylesheets zur Transformation der XML-Daten in die Seitenbeschreibungssprache XSL-FO, ein komplettes Dokument automatisch in das Ausgabeformat PDF gebracht werden kann. Über FrameMaker würde jedes Kapitel einzeln generiert werden und später in einem manuellen Prozess zu einem Gesamtbuch (Masterbuch) zusammengefasst.
Durch den komplett automatischen Vorgang über XSL-FO ergibt sich aber auch ein Nachteil. In FrameMaker gibt es die Möglichkeit der nachträglichen Modifizierung der erzeugten Ausgabedokumente. Es können also Korrekturen am Ausgabedokument auch mit der Ansicht des erzeugten Layouts vorgenommen werden. Eine nachträgliche Bearbeitung des erzeugten Papierlayouts ist beim Einsatz von XSL-FO nicht möglich. Z.B. möchte man aus optischen Gründen einige Stellen auf einer neuen Seite beginnen lassen. Solche Fälle sind aber erst nach Einsicht des Papierlayouts festzustellen. Um solche Probleme zu umgehen wurde die DTD von der LSY um ein zusätzliches Element, mit dem Namen *<newpage>* erweitert. Dieses Element löst einen künstlichen Seitenumbruch aus. Solch ein Element verstößt eigentlich gegen die Regeln von XML, da in XML keine Layoutinformationen hinterlegt werden sollen. Dieses Element war aber von der LH erwünscht, um dem Autor eine gewisse Kontrolle über das Layout zu überlassen.
FrameMaker ist ein Produkt aus einem (seit langer Zeit) erfolgreichem Unternehmen, das auch bei den Anfängen des Desktop-Publishings mitgewirkt hat. Dieses Produkt hat eine lange Entwicklungsphase hinter sich und verspricht auch für die Zukunft ein Produkt zu sein, welches ihre starke Position auf dem Markt halten wird. Die vorhandenen Dokumentationen sind sehr umfangreich. Zusätzlich wird ein umfangreicher Support angeboten. Insgesamt geht man bei diesem Produkt kein großes Risiko ein.
XSL-FO ist ein sehr interessantes Werkzeug, da es, wie auch bei XML, vom W3C spezifiziert ist. XSL-FO wurde direkt für die Anwendung mit XML entwickelt. Durch die internen Kenntnisse der XML-Technologie, kann die Seitenbeschreibungssprache mit großer Effizienz entwickelt werden. Das W3C entwickelt seit 1994 Technologien rund um das World Wide Web und setzt mit einem erfahrenen Team, Partnerunternehmen und öffentlicher Arbeit, Standards, die weltweit eingesetzt werden.
XSL-FO ist einer dieser entwickelten Standards und wird, wie auch andere Spezifikationen aus dem W3C, auch in der Zukunft Bestand haben. Vom W3C stammt aber lediglich die Spezifikation. Die eigentliche Umsetzung geschieht in sogenannten XSL-FO Prozessoren, welche von anderen Unternehmen entwickelt worden sind. Inwieweit diese Produkte eine gute und zukunftssichere Umsetzung anbieten, muss separat untersucht werden.

Auswahl der DTP-Lösung

Da das Verlangen nach einem automatischen Publikationsprozess vorhanden ist, fällt die Wahl auf eine Umsetzung mit XSL-FO. Mit XSL-FO lassen sich die gegebenen Anforderungen gut umsetzen und bietet einen reichhaltigen Umfang an Layoutmöglichkeiten.
Zur Ausgabe der FO Dokumente in PDF muss ein sogenannter XSL-FO Prozessor bereit gestellt werden. Um eine gute Umsetzung und ein zukunftssicheres Produkt auszuwählen, werden im folgenden Abschnitt die marktführenden XSL-FO Prozessoren untersucht.

4.3. XSL-FO Prozessoren

Ein XSL-FO Prozessor ist eine Software, die die Formatierungsanweisungen des XSL-FO Dokuments umsetzt. Man spricht auch von einem Formatierprogramm oder einem Formatierer. Die existierenden Formatierprogramme produzieren in der Regel Zwischenstufen, meist Dokumente in dem von Adobe® entwickelten PDF-Format [TEI07, Kapitel 3.2.5]. Dabei gibt es Unterschiede in der Übersetzung der FO Dokumente. Einige Formatierer stellen Erweiterungen zu den Standard FO-Kommandos bereit. Bei einigen wiederum gibt es auch Einschränkungen im Gebrauch von FO-Kommandos.
Vor der eigentlichen Umsetzung mit XSL-FO muss vorerst ein Blick auf die zur Zeit auf dem Markt vorhandenen XSL-FO Prozessoren geworfen werden. Dabei werden nur die drei bekanntesten untersucht.
Insgesamt gibt es momentan nur wenige Tools, mit denen sich FO Dokumente übersetzen lassen. Die im folgenden beschriebenen XSL-FO Prozessoren sind bereits sehr ausgereift und unterstützen einen Großteil der aktuellen W3C-Recommendation.

4.3.1. Apache FOP

Apache FOP ist ein in Java programmiertes Open-Source-Tool. Im Januar 2007 wurde nach mehreren Beta-Phasen mit der Version 0.93 die erste stabile Version veröffentlicht und erreicht damit eine für den professionellen Einsatz geeignete Umsetzung.
Während man in den Beta-Phasen immer mit vielen kleinen Bugs zu kämpfen hatte wurde mit der Version 0.93 ein neuer Meilenstein gesetzt und somit der Einsatz dieses Tools auch in Unternehmen interessant gemacht.
FOP 0.93 implementiert die Vorschläge des W3C der XSL-Versionen 1.0 und 1.1 zum größten Teil.
Einschränkungen gibt es z.B. bei Tabellen. Hier kann das Element *<fo:table-caption>* nicht umgesetzt werden, welches normalerweise eine Überschrift für die Tabelle erstellt. Der Umgang mit Tabellen-Zellen und -Zeilen hat sich aber mit der neuen Version verbessert, so müssen z.B. wie in früheren Versionen die Spaltenbreiten nicht mehr explizit angegeben werden.
Der Einsatz von *<fo:float>*-Objekten, die das umfließen einer Grafik mit Text beschreibt, wird auch nicht umgesetzt. Eine weitere Einschränkung findet sich in den verschiedenen *writing-modes* (siehe Kap. 2.3.2) die auch nicht unterstützt werden. Insgesamt gibt es eine Vielzahl von Einschränkungen die auch auf der Homepage des Apache Projektes einzusehen sind[8]. Bei den meisten Einschränkungen handelt es sich jedoch um FO-Eigenschaften, die sehr selten eingesetzt werden. Die Umsetzung in Version 0.93 ist soweit vorangeschritten, dass sie ohne große Bedenken in den meisten FO-Umgebungen eingesetzt werden können.

Erweiterungen gibt es bei FOP nur wenige. Eine davon sind die Elemente *<fox:orphan-content-limit>* und *<fox:widow-content-limit>*. Mit diesen Elementen kann für die FO-Objekte *<fo:table>* und *<fo:list-block>* beeinflusst werden, wieviele Zeilen am Anfang oder am Ende einer Seite zusammengehalten werden sollen. Wenn eine Tabelle beispielsweise am Ende einer Seite nur Platz für eine Zeile hat, sollte die Tabelle aus optischen Gründen auf der nächsten Seite beginnen.

[8] einzusehen unter: http://xmlgraphics.apache.org/fop/compliance.html

Da es sich hier um ein Open-Source Projekt handelt hat man leider auch keinen Anspruch auf Support. Es finden sich aber einige FAQ's und Diskussionsforen, die auch von der offiziellen Homepage[9] direkt verlinkt werden.

4.3.2. AntennaHouse XSLFormatter

Der XSLFormatter von AntennaHouse ist mit der Version 4.1 nach eigenen Angaben das erste Produkt, das eine volle Implementierung aller neuen Features der XSL-Version 1.1 des W3C bietet.
Die einzigen Einschränkungen findet man hier bei kaum genutzten FO-Features, wie *<fo:text-shadow>* oder *<fo:color-profile>*.
Insgesamt sind die Umsetzungen der XSL-Version 1.1 sehr umfangreich, was auch die Übersicht auf der eigenen Homepage preisgibt[10].
Der AntennaHouse XSLFormatter V4.1 stellt zudem eine Reihe von Erweiterungen zur Verfügung (über 150, Stand: 30.07.2007). Einige nützliche Erweiterungen sind z.B.:

- *axf:outline-level*:

  Hiermit können Lesezeichen für die PDF-Ausgabe erzeugt werden, um das Navigieren im Dokument zu erleichtern.

- *axf:column-rule-style*:

  Mit diesem Attribut kann für mehrspaltige Seitenlayouts eine vertikale Trennlinie zwischen den Spalten erzeugt werden.

Das Produkt von Antenna House kommt zusätzlich mit einem .NET Interface, COM Interface, Java Interface und C/C++ Interface. Dadurch lässt sich dieses Produkt z.B. gut in Dokumentenmanagementsysteme integrieren und ein vollautomatischer Ablauf erzeugen.
Ein weiterer Zusatz von Antenna House ist der integrierte XSLT-Prozessor, der XML-Dokumente mit XSL-Stylesheets in XSL Formatting Object (XSL-FO) übersetzen kann. Damit hat man alle Schritte in einem Produkt.

Antenna House bietet für dieses Produkt einen umfangreichen Support und stellt zudem sehr hilfreiche Dokumentationen (unter anderem eine FAQ) bereit.
Bei einem Preis von ca. 5.000 US-$ pro Server-Lizenz eignet sich der Einsatz dieses Tool in Unternehmen nur dann, wenn ein hoher Bedarf an PDFs besteht. Durch den stolzen Preis von 1.250 US-$ für eine Einzelplatz-Lizenz ist dieses Produkt in privaten Haushalten wohl eher selten aufzufinden.

4.3.3. RenderX XEP

RenderX XEP ist im Mai 2007 mit der Version 4.10 erschienen. Das Produkt von RenderX ist schon sehr lange auf dem Markt und hat eine lange Entwicklung hinter sich. RenderX XEP erfüllt die W3C-Vorlagen der XSL-Version 1.0.
Die XSL-Version 1.1 ist aber nicht komplett vergessen worden. Für XSL 1.1 stellt XEP eigene Erweiterungen bereit. Werden Elemente oder Attribute aus XSL 1.1 verwendet, konvertiert XEP

[9] http://xmlgraphics.apache.org/fop
[10] http://www.antennahouse.com/xslfo/axf4fo.htm

sie automatisch in die von RenderX bereits umgesetzten Erweiterungen. Trotzdem fehlen viele der neuen Features aus XSL 1.1.
Eines davon ist das Element *<fo:flow-map>*, welches genutzt werden kann um einen Inhaltsbereich einer Seite in mehrere Flow-Elemente aufzuteilen, womit dann z.B. ein zweispaltiger Seitenfluss erstellt werden kann.

Auch bei diesem Produkt ist die Umsetzung sehr umfangreich, so dass sie ohne große Bedenken überall eingesetzt werden kann[11].
Einige erwähnenswerte Erweiterungen von XEP sind:

- *rx:bookmark*:

  Vergleichbar mit Antenna House können hiermit Lesezeichen für die PDF-Ausgabe erzeugt werden, um das Navigieren im Dokument zu erleichtern.

- *rx:index-item*:

  XEP bietet weitreichende Unterstützung bei der Erstellung von einem Index für den Buchdruck. Es können hier beispielsweise doppelt vorkommende Seitenzahlen für einen Index-Eintrag unterdrückt werden.

Auch bei diesem Produkt wird ein technischer Support angeboten. Bei Problemen mit dem Produkt gibt es zudem eine Suchfunktion auf der Homepage für Tipps & Tricks. Auf der Homepage wird zusätzlich ein Live-Chat mit dem Support-Team angeboten (nicht getestet).
Bei einem Preis von etwa 4.000 US-$ pro Server Lizenz ist dieses Tool wohl auch eher für Unternehmen gedacht, in denen PDF-Dokumente eine große Verwendung finden. Eine Einzelplatz Lizenz ist für ca. 300 US-$ zu erwerben. Bei diesem Preis kann sich eine Investition bei häufigem Einsatz lohnen.

4.3.4. Auswahl des XSL-FO Prozessors

Für einen Einsatz in Unternehmen, in denen die Zuverlässigkeit und die Verfügbarkeit der Produkte eine große Rolle spielen, ist es nicht empfehlenswert auf Open-Source Produkte zurückzugreifen, bei denen es immer wieder zu kleinen Problemen kommt. Außerdem ist ein sehr wichtiger Aspekt bei der Auswahl eines Tools der Support, der vorhanden sein muss.
Der Funktionsumfang und der Automatisierungsgrad, den man mit den Produkten XEP von RenderX und XSLFormatter von Antenna House erreicht, machen eine Investition in diese Produkte gerecht. Der Funktionsumfang ist dabei, vor allem bei den hauseigenen Erweiterungen, beim Produkt von Antenna House größer.
Ein weiterer wichtiger Aspekt bei der Auswahl ist der Zugang zu den Informationen rund um das Produkt. Die Übersicht der Antenna House Homepage ist im Vergleich zum RenderX Produkt etwas besser. Die Informationen, die gesucht werden, sind schneller zu erreichen, die nebenbei auch umfangreicher sind als auf der RenderX Homepage.

Für die Realisierung in XSL-FO wird im Folgenden das auf jahrelanger Erfahrung basierende Produkt von Antenna House eingesetzt (Antenna House XSL Formatter V4.1).

[11] Übersicht: http://www.renderx.com/reference.html#XslFoConformance

5. Entwurf

Bevor es an die Realisierung geht wird vorerst ein Entwurf entwickelt, um einen strukturierten Aufbau der Umsetzung zu erreichen. Weiterhin soll somit eine mögliche Wiederverwendung des Quellcodes in folgenden Projekten ermöglicht werden.

5.1. Der Prozessablauf

Die Umsetzung eines Dektop-Publishings über XSL-FO beinhaltet zwei Verarbeitungsschritte (siehe Kap. 2.3.2, Seite 13).
Im ersten Schritt werden die XML-Dokumente über die Transformationssprache XSLT in ein XSL-FO Dokument überführt. Im zweiten Schritt wird das XSL-FO Dokument mit Hilfe eines XSL-FO Prozessors in das Ausgabeformat PDF gebracht. Für den ersten Schritt wird der XSLT-Prozessor 'SAXON'[1] eingesetzt. Die Transformation des XSL-FO Dokuments nach PDF geschieht mit dem Produkt 'Antenna House XSL Formatter' (siehe: Kap. 4.3.4).
Der Prozessablauf ist in Abbildung 5.1 zur Veranschaulichung dargestellt.

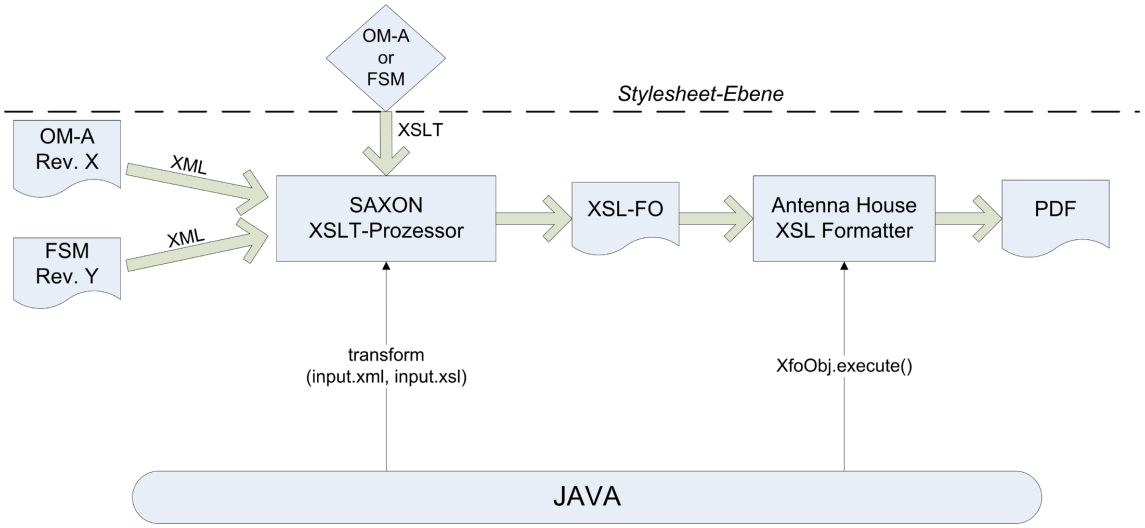


Abbildung 5.1.: Prozessablauf: Verarbeitungsschritte bis zur Ausgabe in PDF (Quelle: Eigene Darstellung)

Für die XSLT-Transformation wird ein XML-Dokument des OM-A oder des FSM in einer beliebigen Revision geladen. Um die Transformation zu starten müssen zusätzlich die zugehörigen Stylesheets, FSM-Stylesheets oder OM-A-Stylesheets, geladen werden. Sind beide Inputquellen geladen, führt der XSLT-Prozessor 'SAXON' seine Arbeit durch und liefert ein XSL-FO Dokument.
Das XSL-FO Dokument aus dem ersten Schritt wird vom 'Antenna House XSL Formatter' interpretiert und in ein PDF-Dokument überführt.
Die eingesetzten XSLT- und XSL-FO-Prozessoren stellen eine JAVA API zur Verfügung. Die Aufrufe der zwei Verarbeitungsschritte sollen, mit Hilfe dieser APIs, über eine JAVA-Schnittstelle,

[1] siehe: http://saxon.sourceforge.net/

die zusätzlich realisiert werden muss, durchgeführt werden.
Die Stylesheet-Ebene beinhaltet eine erweiterte Logik, welche im nächsten Abschnitt beschrieben wird.

5.2. Aufteilung in Module

Wie bereits in der Analyse festgestellt, gibt es mehrere Elemente, welche die gleiche Behandlung im Layout erhalten. Bei einigen Elementen weicht das Verhalten zwischen OM-A und FSM ab.
Für die Realisierung werden aus diesem Grund die umzusetzenden Elemente in allgemeingültige und spezifische Module aufgeteilt. Das Ziel dabei ist, ähnliche Beschreibungen nicht doppelt zu definieren und typische Elemente dieser Dokumente allgemein zu halten, so dass sie von weiteren Dokumenten, die in der Zukunft hinzu kommen können, genutzt werden können.
Um eine geeignete Übersicht über den Code zu erhalten, werden die Module zusätzlich in mehrere kleine Stylesheets aufgeteilt. Elemente die ein ähnliches Verhalten aufweisen oder zusammengehören (wie z.B. bei *table* mit ihren vielen Unterelementen), werden in einem Stylesheet zusammengefasst. Dazu folgt ein Beispiel:
inline-Elemente (z.B. fettgedruckte, kursive Abschnitte) kommen in das Stylesheet-Modul 'inline-Elemente' und werden allgemeingültig behandelt, da sie in beiden Dokumenten dasselbe Verhalten aufweisen.

Nach Untersuchung der DTD, der Layoutanforderungen und den Grundlagen von XSL-FO ergibt sich folgende Aufteilung der Module:

- **OM-A**
  - **Schnittstelle** für die Transformation von OM-A Dokumenten. (hier werden unter anderem alle Stylesheets eingebunden, die für das OM-A relevant sind. (*om-a.xsl*)
  - **Variablen** zur Beschreibung spezifischer Eigenschaften. (*vars.xsl*)
  - **<fo:root>** mit Seitenvorlagen. (*root.xsl*)
  - **static-content** der Seitenvorlagen, also Header und Footer. (*static.xsl*)
  - **Warning-/Caution-/Note-Elemente**. (*w-c-n.xsl*)
  - **Inhaltsverzeichnis**. (*toc.xsl*)
  - **Index** (*index.xsl*)
- **FSM**
  - **Schnittstelle** für die Transformation von FSM Dokumenten. (hier werden unter anderem alle Stylesheets eingebunden, die für das FSM relevant sind. (*fsm.xsl*)
  - **Variablen** zur Beschreibung spezifischer Eigenschaften. (*vars.xsl*)
  - **<fo:root>** mit Seitenvorlagen. (*root.xsl*)
  - **static-content** der Seitenvorlagen, also Header und Footer. (*static.xsl*)
  - **Warning-/Caution-/Note-Elemente**. (*w-c-n.xsl*)
  - **Inhaltsverzeichnis**. (*toc.xsl*)
  - **Index** (index.xsl)
- **common**

- **Block-Elemente** wie <para> (Absätze), <txtgrphc> (Textgrafiken). (*block.xsl*)
- **Grafische Elemente**, z.B. <graphic> und <inlpgrh> (*graphix.xsl*)
- **Inline-Elemente**, wie z.B. Schriftformatierungen innerhalb einer Zeile (fettdruck etc.). (*inline.xsl*)
- **Nummerierte Listen**. (*numlist.xsl*)
- **Unnummerierte Listen** (*unumlist.xsl*)
- **Referenzen** mit dem ID-Attribut, z.B. <refint> und <refext>. (*reference.xsl*)
- **Tabellen** (*table.xsl*)
- **Top-Level Elemente**, wie z.B. <chapter>, <section> etc. (*top-level.xsl*)
- **Sonstige Elemente**, die nicht zugeordnet werden können. (*misc.xsl*)
- **nicht-definierte Elemente**. Diese sollen rot dargestellt werden, um zu erkennen welche Elemente vergessen wurden. (*default.xsl*)

Die Variablen die für die OM-A- und FSM-Stylesheets eingeführt werden dienen dazu, kleine Abweichungen im Layout zu beschreiben. Elemente, die bei beiden Dokumenten ein ähnliches Verhalten haben und somit allgemeingültig behandelt werden, die aber trotzdem kleine Abweichungen im Layout enthalten, wie z.B. Schriftgrößen von Überschriften (*titles*), verwenden diese Variablen. Durch den Einsatz solcher Variablen muss bei geringfügigen Abweichungen im Layout keine spezifische Behandlung der Module umgesetzt werden. Beispielsweise müssen bei verschiedenen Labels in unnummerierten Listen die Umsetzung dieses Elements nicht für beide Dokumente geschrieben werden, sondern es kann in einer allgemeingültigen Beschreibung das Symbol durch eine Variable geladen werden. Der Variablenname muss dazu in beiden Stylesheets denselben Namen tragen. Diese Regel, dass die Variablen in FSM und OM-A dieselben Namen tragen müssen, gilt für alle Variablen die eingeführt werden.

Die Aufteilung der Module ist in Abbildung 5.2 bildlich dargestellt.

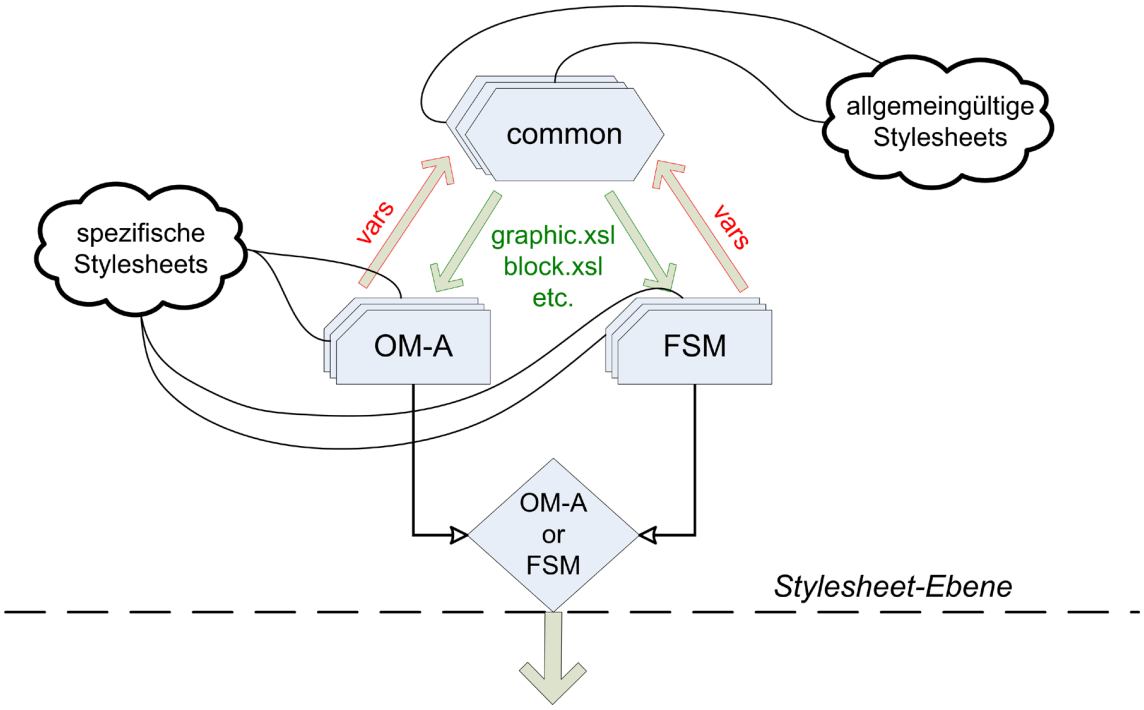


Abbildung 5.2.: Zusammenspiel der Stylesheets (Quelle: Eigene Darstellung)

Das Laden der verschiedenen Stylesheets soll für den Aufruf der Transformation transparent bleiben. Wird eine XSLT-Transformation von der JAVA-Anwendung gestartet, wird lediglich die Schnittstelle des jeweiligen Moduls (OM-A oder FSM) aufgerufen. Das Zusammenspiel der verschiedenen Stylesheets geschieht dabei intern auf der Stylesheet-Ebene.

6. Realisierung

Ausgehend von technischen Dokumentationen, die im XML-Format vorliegen, soll ein möglichst automatischer Prozess diese Dokumente in ein Ausgabeformat für den Druck auf Din A4 übersetzen. Dies geschieht mit der Seitenbeschreibungssprache XSL-FO und dem Ausgabeformat PDF. Die XML-Daten müssen über XSL-Stylesheets in das FO-Format übersetzt werden. Die Grundlage dafür ist die 'ops-doc' DTD in der Version 4.2.
Aufbauend auf dem Entwurf der im vorangegangenen Kapitel entwickelt wurde, wird nun die Realisierung beschrieben.
Dieses Kapitel enthält Ausschnitte aus dem entwickelten Quellcode. Der komplette Quellcode ist auf der beigelegten CD enthalten.

6.1. Allgemeine Umsetzung

6.1.1. Seitenlayout und Abfolge der Seiten

Beide Dokumente, das OM-A und das FSM, haben einen ähnlichen Seitenaufbau. Beide enthalten einen Kopfbereich (Header), einen Inhaltsbereich (Body) und einen Fußbereich (Footer). Im Folgenden wird die Beschreibung des Seitenlayouts in XSL-FO, am Beispiel des Dokuments FSM, erarbeitet.
Wie das Seitenlayout und die Abfolge der Seiten in FO beschrieben wird, ist bereits in Kapitel 4.2.2: weitgehend erläutert. Bei den drei Bereichen einer FSM-Seite handelt es sich um eine *<fo:region-before>* (Kopfbereich), eine *<fo:region-body>* (Inhaltsbereich) und eine *<fo:region-after>* (Fußbereich). Um die Seitenvorlagen zu erstellen, müssen folgende Parameter gegeben sein. Die Zugehörigkeit der Parameter ist in Abbildung 4.9 dargestellt.

Für Seitenmaße und Seitenränder (*<fo:simple-page-master>*-Attribute):

- *page-height, page-width*
- *margin-top, margin-bottom, margin-left, margin-right*

Für *region-before*:

- *extent*

Für *region-body*:

- *margin-top, margin-bottom, margin-left, margin-right*

Für *region-after*:

- *extent*

Diese Parameter müssen jeweils für die linke und die rechte Seite gesammelt werden. Nach der Sammlung der Werte[1] kann die Umsetzung in FO erfolgen, welche in Listing A.1 einzusehen ist.

An dem Code ist zu erkennen, dass die *body-region* der *blank page* leer bleibt. Dies geschieht, da die body-region keinen statischen Inhalt enthalten kann, sondern nur einen dynamischen

[1] siehe layout-definition: *CD:<//LSY/Doku/layout-definition.xls>*

Inhalt, der aus der XML-Quelle kommt. Der statische Text „Aus redaktionellen Gründen frei“ (bzw. „Intentionally Left Blank“) wird daher mit in den Kopfbereich geschrieben und bekommt einen Abstand nach oben, damit er in den *body* rutscht.
Die Regionen erhalten auch einen eindeutigen Namen mit dem Attribut *region-name*. Dadurch können sie später referenziert und mit Inhalt gefüllt werden. Ist der Name nicht vergeben, wie beim *<fo:region-body>*, wird ein Default-Name nach dem Muster „xsl-Bereichsname“ vergeben. So erhalten in diesem Fall alle *body-regions* denselben Namen. Dies sollte auch der Fall sein, um den Inhalt später in den *body* der verschiedenen Seiten fließen lassen zu können, da eine *body-region* nur einmal referenziert wird. Bei der *blank page* bekommt der Header einen anderen Namen, weil der Inhalt der *blank page* eine separate Behandlung verlangt.
Die logische Abfolge der Seiten kann nach der XSL-FO Analyse problemlos definiert werden. Die Umsetzung ist in Listing A.2 zu finden.

Rechte Seitenvorlagen werden bei ungeraden Seitenzahlen verwendet und linke Seitenvorlagen bei geraden Seiten. Somit startet ein Kapitel immer mit einer rechten Seite mit der Seitennummer 1.
Um die Funktion der *blank page* zu nutzen, bekommt das Tag *<fo:page-sequence>*, welches diesen *<fo:page-sequence-master>* referenziert, das Attribut *force-page-count=“end-on-even“*. Dieses Attribut erzwingt, dass eine page-sequence (ein Kapitel) mit einer geraden, linken Seite endet.

6.1.2. *static*-Content und *flow*-Content

Im nächsten Schritt wird der zuvor beschriebene *<fo:page-sequence-master>* referenziert und die Seiten mit Inhalt gefüllt. Der Body bekommt einen sogenannten *flow*-Content, in den der Inhalt der XML-Daten hineinfließen kann. Die anderen Bereiche (Kopf- und Fußbereich) bekommen mit Hilfe der Tags *<fo:static-content>* einen statischen Inhalt. Dieser statische Inhalt wird auf allen erzeugten Seiten wiederholt. ’Statisch’ bedeutet, dass diese Informationen einmal definiert werden und dann für die komplette *page-sequence* (ein Kapitel) gelten. Dabei kann es sich auch um logische Parameter, wie Seitennummern, handeln.
Eine Sequenz von Seiten beginnt mit dem Tag *<fo:page-sequence>*. Um die Funktion der *blank page* zu nutzen und jedes Kapitel mit der Seitennummer 1 beginnen zu lassen, ist jedes Kapitel eine eigene *page-sequence*.
Innerhalb einer *<fo:page-sequence>* kommt die Beschreibung von static- und flow-Content. Der Aufbau solch einer *page-sequence* wird in Listing A.3 gezeigt.

Direkt unter dem Element *<fo:page-sequence>* müssen die Tags *<fo:static-content>* und *<fo:flow>* definiert werden. Innerhalb der Tags *<fo:static-content>* und *<fo:flow>* wird mit den sogenannten „Block-Areas“ von XSL-FO, wie z.B. *<fo:block>* oder *<fo:table>*, gearbeitet, um den eigentlich darzustellenden Inhalt zu beschreiben.

Der *flow*-Content erhält in dem beschriebenen Listing zunächst einen Block für die Daten die nun folgen, um den Inhalt zu füllen. Dieser Block beschreibt Metadaten für die Inhalte. Dazu gehören die Attribute *hyphenate*, zur Nutzung der automatischen Worttrennung bei einem erzwungenen Zeilenumbruch, dem dazugehörigen Attribut *language*, sowie die *font*-Attribute zur Definition der Standardschriftart und Größe.

Der *static*-Content ist im vorangegangen Code nicht näher beschrieben. Zur inhaltlichen Gestaltung des Kopfbereichs bedarf es einer genaueren Untersuchung.
Die Informationen im Header werden in eine Tabellenstruktur gefüllt. Es werden 4 Zeilen und 4 Spalten genutzt. Die Aufteilung wird in Abbildung 6.1 dargestellt.

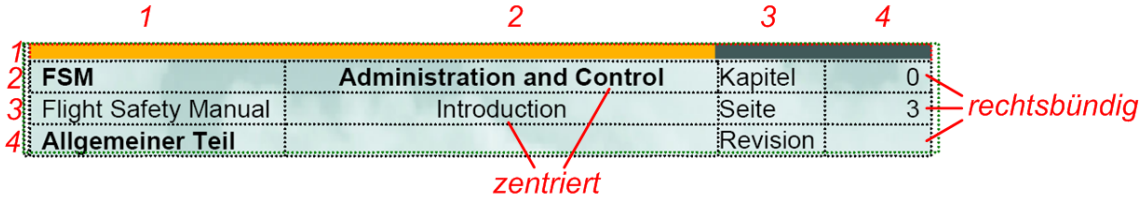


Abbildung 6.1.: Kopfbereich des FSM in einer Tabelle aufgeteilt (Quelle: Eigene Darstellung)

Das Hintergrundbild wird in einer darunter liegenden Tabelle als *background-image* definiert.
Die lebenden Kolumnentitel (Seitennummer, Kapitelüberschrift, etc.) werden in der folgenden Übersicht beschrieben:

- **Kapitelüberschrift:**

  Die Kapitelüberschriften ergeben sich aus dem *<title>*-Element, das direkt unter einem *<chapter>*- bzw. *<general-info>*-Element liegt. An diese Information kommt man über folgenden XSLT-Ausdruck:

```
<xsl:value-of select="ancestor-or-self::general-info/title
                    | ancestor-or-self::chapter/title"/>
```

- **Kapitelnummer:**

  Die Kapitelnummer kommt aus dem *fixnum*-Attribut des jeweiligen Kapitels:

```
<xsl:value-of select="@fixnum"/>
```

- **section-Überschrift:**

  Die Überschrift der aktuellen *section* kann mit dem FO-Element *<fo:retrieve-marker>* gesucht werden. Dafür muss jedes *<section>*-Element einen *<fo:marker>* zur Kennzeichnung bekommen. Folgender Code sucht, ausgehend von der aktuellen Position im XML-Strom, das *<fo:marker>*-Objekt für ein *<section>*-Element, das unmittelbar vorausgeht.

```
<fo:retrieve-marker retrieve-class-name="sectionTitle"
                    retrieve-position="first-including-carryover"
                    retrieve-boundary="page-sequence"/>
```

- **Seitennummer:**

  Das Generieren von Seitennummern kann in FO folgendermaßen gelöst werden:

```
<fo:page-number/>
```

- **Revisionsnummer:**

  Die Revisionsnummer steht im *root*-Element der XML-Datei im Attribut *revnbr*:

```
<xsl:value-of select="ancestor::ops-vol/@revnbr"/>
```

Mit den vorliegenden Informationen kann nun der *static*-Content für die Header erstellt werden.

Für die *blank page* kommt folgender Block unter die Tabelle des Headers:

```
<fo:block space-before="122mm" text-align-last="center"
          font-family="{$normal.font-family}" font-size="14pt">
  <xsl:text>Aus redaktionellen Gründen frei</xsl:text>
</fo:block>
```

FO-Block für Notiz der blank page (Quelle: Eigene Beschreibung)

Mit dem Attribut *space-before* kann ein Block in Relation zum oberen Seitenrand nach unten geschoben werden.
Der Fußbereich wurde ebenso mit Hilfe des Tabellenmodells gelöst und gestaltet sich problemlos. Die Umsetzung kann in Listing A.4 eingesehen werden.

Nun, da das Grundgerüst der Seiten steht, kann mit der Beschreibung der Inhalte begonnen werden.

6.1.3. Anwendung von XSLT-Schablonen (<xsl:template>)

Bevor die inhaltlichen Elemente erläutert werden, werden zunächst einige verwendete Konzepte und Konstrukte beschrieben. Diese werden eingesetzt, um die Effizienz und die Wartbarkeit des Quellcodes zu erhöhen.

Konstrukte, die häufig auftauchen, werden in sogenannte XSLT-Schablonen geschrieben und mit einem Namen versehen. Dadurch können sie jederzeit im Quellcode aufgerufen werden. Zu den entwickelten XSLT-Schablonen gehören unter anderem:

- fixnum (Generieren der Abschnittsnummer aus den verschiedenen fixnum-Attributen)
- title (Generieren der Abschnittsüberschriften)
- getid (zum Auflösen eines id-Attributs)
- setid (zum Setzen eines id-Attributs)
- tocentry (erstellt einen Eintrag im Inhaltsverzeichnis)
- generierte Texte im Header (wie z.B. aktueller Kapitelname)

Die XSLT-Schablonen sind ebenfalls dazu geeignet, die Übersicht des entwickelten Codes zu verbessern. So wird das Füllen der Seiten mit dem *static-Content* in eine eigene aufrufbare Schablone gepackt und innerhalb dieser Schablone wieder in seine Einzelteile zerlegt. Dadurch wird für die Elemente *<chapter>* und *<general-info>*, die eine *<fo:page-sequence>* erstellen und somit *static*- und *flow*-Content enthalten, die eigentliche Beschreibung der Inhalte verlagert. Mit Hilfe dieses Vorgangs, bleibt die Struktur der FO Code-Elemente überschaubar.
Der Quellcode für die Schablonen *<chapter>* und *<general-info>* wird in Listing A.5 beschrieben. Darin wird die eigentliche Beschreibung in XSL-FO wieder in eine andere Schablone verlagert. Diese Schablone, mit dem Namen 'fill-static-content', wird in der Datei 'static.xsl' definiert und ist in Listing A.6 beschrieben.
Durch die Aufteilung von Struktur und inhaltlicher Beschreibung wird die Wartbarkeit des Quellcodes erhöht.

6.1.4. Die Arbeit mit Variablen

Wie schon im Abschnitt 5.2 erwähnt, werden Variablen eingesetzt, um Abweichungen in allgemeingültigen Modulen zu steuern. In diesem Abschnitt soll die Arbeit mit Variablen an einigen Beispielen betrachtet werden. Folgende Module bei denen Variablen eingesetzt werden, werden nun erläutert:

- *unumlist*
- *title*
- *table*

unumlist

Unnummerierte Listen haben bei beiden Dokumenten denselben Aufbau. Sie unterscheiden sich lediglich im Symbol des Labels (und einem zusätzlichen Abstand nach unten). Um die Beschreibung für unnumerierte Listen nicht für beide Dokumente einzeln behandeln zu müssen, werden hier folgende Variablen verwendet:

```
<xsl:variable name="unumlist.label.character">&#x2D;</xsl:variable>
<xsl:variable name="unumlist.label.font-family">Helvetica</xsl:variable>
<xsl:variable name="unumlist.margin-bottom">2pt</xsl:variable>
```

FSM Variablen für *<unumlist>* (Quelle: Eigene Beschreibung)

Diese Variablen können dann in der Umsetzung für *<unumlist>* geladen werden. Variablen werden in XSLT mit einem vorangestellten Dollar-Zeichen ('$') aufgerufen.
Die Umsetzung ist in Listing A.7 einzusehen.
Für das Erstellen von Listen gibt es in FO das Konstrukt *<fo:list-item>*. Dieses Konstrukt wird hier verwendet. Es besteht aus einem Label und einem Body-Bereich für den Inhalt. Der Abstand zwischen Label und Inhalt kann mit diesem Konstrukt genau definiert werden.
Für die Beschreibung des Labels wurden hier die zuvor definierten Variablen eingesetzt. Durch diese Beschreibung kann der Code für beide Dokumente eingesetzt werden und muss nicht doppelt definiert werden.

title

Das Generieren der Abschnittsüberschriften (*<title>*) verbirgt eine ganze Menge an Parametern, die sich in beiden Dokumenten FSM und OM-A unterscheiden. Dazu zählen die Schriftgrößen, der Abstand zur Abschnittsnummer (*fixnum*-Attribut), sowie die Zeilenhöhe. Im Folgenden ein Ausschnitt der Variablen für das Element *<title>*:

```
<xsl:variable name="chapter.title.font-size">12pt</xsl:variable>
<xsl:variable name="chapter.title.line-height">22pt</xsl:variable>
<xsl:variable name="chapter.title.baseline-shift">5pt</xsl:variable>
<xsl:variable name="section.title.font-size">12pt</xsl:variable>
<xsl:variable name="section.title.line-height">22pt</xsl:variable>
<xsl:variable name="section.title.baseline-shift">5pt</xsl:variable>
<xsl:variable name="l1.title.font-size">12pt</xsl:variable>
<xsl:variable name="l1.title.line-height">19pt</xsl:variable>
<xsl:variable name="l1.title.baseline-shift">3.5pt</xsl:variable>
```

OM-A Variablen für *<title>* (Quelle: Eigene Beschreibung)

Parameter können nach Definition dieser Variablen dynamisch im Stylesheet geladen werden. Da Überschriften aus dem Element *<title>* immer wieder erstellt werden, wird die Umsetzung in eine eigene aufrufbare Schablone gesetzt. Diese Schablone erhält den Namen 'createTLtitle'. Ein Kapitel ruft diese Schablone beispielsweise wie folgt auf:

```
<xsl:call-template name="createTLtitle">
  <xsl:with-param name="start-distance" select="'15mm'"/>
  <xsl:with-param name="margin-bottom" select="'14pt'"/>
  <xsl:with-param name="font-size" select="$chapter.title.font-size"/>
  <xsl:with-param name="font-weight" select="'bold'"/>
  <xsl:with-param name="line-height" select="$chapter.title.line-height"/>
  <xsl:with-param name="baseline-shift"
                  select="$chapter.title.baseline-shift"/>
</xsl:call-template>
```

Aufruf von 'createTLtitle' (Quelle: Eigene Beschreibung)

Der aufgerufenen Schablone werden Parameter übergeben. Einige Parameter sind fest vorgegeben, da sie sich bei FSM und OM-A nicht unterscheiden. Einige Parameter enthalten die bereits besprochenen Variablen, die ein unterschiedliches Verhalten der verschiedenen Dokumente erreichen sollen.
Die Umsetzung von 'createTLtitle' wird in Listing A.8 aufgezeigt.
In der Umsetzung wird eine Vielzahl von Variablen eingesetzt, die dynamisch die verschiedenen Schriftarten, Abstände, etc. laden. Im Template 'createTLtitle' bekommen die Parameter mit der *select*-Anweisung initiale Werte, die aber bei Aufruf dieser Schablone überschrieben werden können.
Auch hier wurde das *<fo-list-item>* Konstrukt von FO eingesetzt. Die Abschnittsnummer (*fixnum*) wird hier als Label und der Titel als Inhalt definiert.

table

Die Arbeit mit Variablen bei Tabellen soll nun anhand einiger Beispiele kurz erläutert werden. Beide Dokumente, das OM-A und das FSM, haben für die Tabellenrahmen eine unterschiedliche Farbe. Um dies für beide Dokumente zu beschreiben, wird folgende Variable eingeführt:

```
<xsl:variable name="table.border-color"/>
```

Im Stylesheet kann eine FO-Tabelle diese Variable nutzen, um die Farbe der Tabellenrahmen zu spezifizieren:

```
<xsl:attribute name="border-color">
  <xsl:value-of select="$table.border-color"/>
</xsl:attribute>
```

Stylesheet: Aufruf der Variable 'table.border-color' (Quelle: Eigene Beschreibung)

Zusätzlich haben die Tabellen unterschiedliche Hintergrundfarben für Header und Footer. Folgende Variable erreicht diese Unterscheidung:

```
<xsl:variable name="table.background-color"/>
```

Auch die Dicke der Tabellenrahmen unterscheidet sich in beiden Dokumenten. Innere und äußere Tabellenrahmen müssen ebenfalls unterschieden werden. Dadurch ergeben sich letztendlich folgende Variablen:

```
<xsl:variable name="table.inner-border-width"/>
<xsl:variable name="table.outer-border-width"/>
```

Mit all diesen Variablen können somit die Unterschiede zwischen OM-A und FSM auf einfache Weise gesteuert werden.

6.1.5. Umsetzung der grundlegenden Module

Die Umsetzung einiger Module/Elemente wurde bereits in den vorangegangenen Abschnitten erläutert. In diesem Abschnitt soll die Umsetzung weiterer grundlegender Module erläutert werden. Im Vordergrund der Beschreibung steht hier das Zusammenspiel von XSLT und XSL-FO. Folgende Module/Elemente werden in diesem Abschnitt behandelt:

- *<para>*
- Inhaltsverzeichnis (Table Of Contents)
- Referenzen (ID Vergabe und Auflösung)

Im nächsten Kapitel (Kap 6.2) werden darüber hinaus die Module/Elemente behandelt, die für die Umsetzung einen höheren Aufwand erfordern.

<para>

Die *<para>*-Elemente sind in den Dokumenten am häufigsten vertreten und in der Umsetzung sehr einfach. Ein *<para>*-Element beschreibt einen Textabsatz und wird in FO mit dem Element *<fo:block>* beschrieben. Die einzige Regel, die hier beachtet werden muss, ist die Notwendigkeit jeweils einen Abstand von 6pt zwischen zwei *<para>*-Elementen zu setzen. Die Umsetzung sieht wie folgt aus:

```
<xsl:template match="para">
  <fo:block>
    <!-- Wenn Oberer Nachbar auch para dann Abstand von 6pt -->
    <xsl:if test="preceding-sibling::*[1][self::para]">
      <xsl:attribute name="margin-top">6pt</xsl:attribute>
    </xsl:if>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>
```

Stylesheet zur Umsetzung von <para> (Quelle: Eigene Beschreibung)

Die Schriftart, die Schriftgröße und weitere Eigenschaften sind bereits in höher liegenden Elementen beschrieben (*chapter, general-info*). Diese Eigenschaften werden vererbt und müssen somit nicht erneut definiert werden.

Inhaltsverzeichnis

Die Inhaltsverzeichnisse werden in beiden Dokumenten an verschiedenen Stellen erstellt. Im OM-A steht am Anfang jedes Kapitels ein Inhaltsverzeichnis. Beim FSM gibt es am Anfang des Dokuments ein Inhaltsverzeichnis für das gesamte Dokument.
Für das OM-A geschieht die Erstellung des Inhaltsverzeichnisses, wenn im XML-Dokument ein *general-info* oder *chapter*-Element auftaucht (*<xsl:template match="general-info | chapter“>*). Da diese Schablone bereits im *common*-Verzeichnis (das Verzeichnis mit den allgemeingültigen Regeln) definiert wurde, muss sie hier überschrieben werden. Dies erfolgt mit der Vergabe einer Priorität für diese Schablone:

```
<xsl:template match="chapter/title" priority="1">
```

Das FSM generiert nach der Überschrift von Abschnitt 0.1 das Inhaltsverzeichnis. Hier muss ebenfalls eine Priorität vergeben werden:

```
<xsl:template match="general-info/section[@fixnum='1']/title" priority="1">
```

Der Aufbau der Inhaltsverzeichnisse ist bei beiden Dokumenten sehr ähnlich. Des Weiteren wird nur der Aufbau des Inhaltsverzeichnisses für das Dokument OM-A beschrieben.
Das Inhaltsverzeichnis enthält drei Informationen: die Abschnittsnummer, den Titel des Abschnitts, sowie die zugehörige Seitennummer. Um diese Informationen untereinander mit gleichmäßigen Abständen anzuzeigen, wird das FO Tabellenkonstrukt verwendet. In eine Tabelle mit drei Spalten wird in die erste Spalte die Information der Abschnittsnummer gesetzt, in der mittleren Spalte der Titel des Abschnitts und in der letzten Spalte rechtsbündig die Seitennummer.
Im Inhaltsverzeichnis finden sich Informationen zu den *sections* und der *l1item*-Ebene. Mit dem XSLT-Ausdruck:

```
<xsl:for-each select="$act_chapter/section">
```

werden alle *sections* markiert und nacheinander durchlaufen. Innerhalb dieses Ausdrucks müssen nun die *section*-Informationen, sowie die Informationen der zugehörigen *sub-sections* generiert werden. Die zugehörigen *sub-sections* der aktuellen *section* werden mit folgendem Ausdruck markiert:

```
<xsl:for-each select="//chapter[@fixnum = $chapNr]/
                      section[@fixnum = $act_section_nr]/*/l1item">
```

Der Quellcode für das Erstellen des Inhaltsverzeichnisses ist in Listing A.9 zu finden.

Die Informationen, die ausgedruckt werden sollen, sind bei *section*- und *l1item*-Elementen die selben. Daher kann diese Beschreibung erneut in einer aufrufbaren Schablone erstellt werden (Name: 'tocentry'). Die Informationen der einzelnen Einträge sind recht einfach zu generieren. Die Abschnittsnummer ergibt sich aus den *fixnum*-Attributen. Eine Umsetzung in XSLT zur Zusammensetzung der fixnum-Attribute ist in Listing A.10 beschrieben.

Den Titel des Abschnitts erhält man mit folgendem Ausdruck:

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

Um die Seitennummer herauszufinden, muss eine Referenz zum jeweiligen Abschnitt hergestellt werden. Dafür stellt FO die Funktion *<fo:page-number-citation>* bereit. Hiermit kann über die Angabe einer eindeutigen ID, die für jeden Abschnitt erstellt werden muss, die Seitennummer aufgelöst werden. Die Auflösung der Referenzen wird im nächsten Abschnitt beschrieben.

Referenzen

Innerhalb der Dokumente sollen mit ID-Attributen Referenzen hergestellt werden können. Nach der DTD haben viele Elemente ein ID-Attribut. Diese ID-Nummern müssen innerhalb des gesamten Dokuments eindeutig sein. Ein Problem ist, dass nach der DTD diese ID-Attribute nicht alle den Namen 'id' tragen. Bei einigen Elementen tragen sie den Namen 'key' und einmal taucht der Name 'ftnoteid' auf. Alle diese Attribute sollen später in der Verarbeitung zu FO übernommen werden. Folgende Elemente sind betroffen:

- *<ftnote>* (mit Attribut 'ftnoteid')
- *<table>* (mit Attribut 'id')
- *<list(N)>* (mit Attribut 'key') N=1,...,4
- *<l(N)item>* (mit Attribut 'key') N=1,...,4
- *<numlist>* (mit Attribut 'key')
- *<numlitm>* (mit Attribut 'key')
- *<unumlist>* (mit Attribut 'key')
- *<unumlitm>* (mit Attribut 'key')
- *<w-c-n>* (mit Attribut 'key')
- *<itemdesc>* (mit Attribut 'key')
- *<general-info>* (mit Attribut 'key')
- *<chapter>* (mit Attribut 'key')
- *<section>* (mit Attribut 'key')
- *<glossary>* (mit Attribut 'key')
- *<inlgrph>* (mit Attribut 'key')
- *<graphics>* (mit Attribut 'key')
- *<sheet>* (mit Attribut 'key')

Diese Attribute werden in das FO-Attribut 'id' umgesetzt. Ist das Attribut in der XML-Quelle gesetzt, wird diese ID-Nummer übernommen. Ist das Attribut nicht gesetzt, wird sie in FO explizit gesetzt. Die Generierung von ID's geschieht in FO über den Aufruf:

```
{generate-id()}
```

Das Auflösen von IDs, die über diesen Ausdruck generiert worden sind, geschieht ebenfalls durch denselben Aufruf. Die Verbindung stellt FO automatisch her.
Es muss bei der Realisierung darauf geachtet werden, ob eine ID aus der XML-Quelle entnommen werden kann oder ob sie von FO erstellt werden muss.
Es soll eine Funktion zur Generierung der IDs (<xsl:template name="setid">), sowie eine Funktion zur Auflösung einer ID (<xsl:template name="getid">) bereitgestellt werden .

```
<xsl:template name="setid">
  <xsl:param name="attribute-name" select="'key'"/>
  <xsl:attribute name="id">
    <xsl:call-template name="getid">
      <xsl:with-param name="attribute-name">
        <xsl:value-of select="$attribute-name"/>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:attribute>
</xsl:template>

<xsl:template name="getid">
  <xsl:param name="attribute-name" select="'key'"/>
  <xsl:choose>
    <!-- Wenn ein Attribut id|key|ftnoteid bereits im XML-Dokument
      -- vorhanden -->
    <xsl:when test="@*[name()= $attribute-name]">
      <xsl:value-of select="@*[name()= $attribute-name]"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="generate-id()"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Stylesheet: 'get-id' und 'set-id' (Quelle: Eigene Beschreibung)

Der Parameter 'attribute-name' löst das Problem der verschiedenen ID-Attributnamen aus der DTD. Der jeweilige Attributname ('id', 'key' oder 'ftnoteid') kann hiermit übergeben werden.
Im Template 'setid' wird das template 'getid' aufgerufen, da das Setzen einer ID genauso aufgebaut ist wie das Auflösen einer ID (*generate-id()*, oder Übernahme aus XML-Quelle). Im Template 'getid' erfolgt dann die Abfrage, ob in der XML-Quelle bereits ein entsprechendes ID-Attribut vorhanden ist. Ist keines vorhanden, wird die Funktion von FO genutzt.

6.2. Besonderheiten

Bei der Umsetzung in XSL-FO stößt man gelgentlich an seine Grenzen. Auch bei diesem Projekt tauchen Elemente auf, bei denen die Umsetzung nicht ohne erhöhten Aufwand möglich ist. Diese komplexeren Elemente sollen in diesem Abschnitt untersucht und gelöst werden. Zu den in diesem Abschnitt besprochenen Elementen gehören:

- Index (Register für Schlagwörter)
- Tabellen (CALS Tabellenmodell in FO Tabellenmodell)

6.2.1. Index

Das Register taucht nur im Dokument OM-A auf und befindet sich in Abschnitt 0.1.7. Es hat ein abweichendes Inhaltslayout, da die Informationen zweispaltig angezeigt werden (siehe Kap. 4.1.2, Seite 24). Um diesen zweispaltigen Body-Bereich zu erhalten, gibt es in FO mehrere Lösungswege.
Ein Lösungsweg ist der, neue Seitenvorlagen zu erstellen, die zwei *body-regions* enthalten. Das Erstellen von mehreren *body-regions*, in denen der Inhalt dann nacheinander hineinfließen kann, ist erst seit der XSL Version 1.1 möglich (Dezember 2006).
Sind diese *body-regions* definiert, muss zusätzlich eine sogenannte *flow-map* angelegt werden. Diese definiert dann die Reihenfolge der *body-regions*, die für den *flow*-Content gilt. Wurden zwei *body-regions* definiert, eine mit dem Namen 'L' und eine mit dem Namen 'R', so sieht die *flow-map* folgendermaßen aus:

```
<fo:flow-map flow-map-name="E1">
  <fo:flow-assignment>
    <fo:flow-source-list>
      <fo:flow-name-specifier flow-name-reference="A"/>
    </fo:flow-source-list>
    <fo:flow-target-list>
      <fo:region-name-specifier region-name-reference="L"/>
      <fo:region-name-specifier region-name-reference="R"/>
    </fo:flow-target-list>
  </fo:flow-assignment>
</fo:flow-map>
```

Stylesheet: *flow-map* für zweispaltigen Index (Quelle: Eigene Beschreibung)

Zur Anwendung dieser *flow-map* wird bei der Definition der *page-sequence* und des *flow*-Contents eine Referenz hergestellt:

```
<fo:page-sequence master-reference="om-a-index" flow-map-reference="E1">
...
  <fo:flow flow-name="A">
```

Die Informationen, die im Register zu finden sind, sind Schlagwörter die im XML-Dokument mit dem Attribut *entry* des Elements *<index>* markiert wurden. Diese müssen alphabetisch sortiert werden. Wie es auch aus anderen Büchern bekannt ist wird im Register jeder Abschnitt, der mit einem neuen Buchstaben beginnt, mit dem Anfangsbuchstaben gekennzeichnet (siehe Abbildung 4.5). Rechtsbündig der Schlagwörter sind die Abschnittsnummern aufgelistet, in

denen das Wort zu finden ist. Dabei muss darauf geachtet werden, dass keine Nummer doppelt vorkommt. Die beiden Informationen werden durch eine Punktierung getrennt, welche ihre Größe dynamisch, je nach Abstand zwischen Schlagwort und Abschnittsnummer, anpassen muss.

Der erste Schritt zur Erstellung der Index-Informationen ist die Vergabe von 'Schlüssel' für die Index-Einträge ('index-entry') und die Anfangsbuchstaben ('initial-letter'). 'Schlüssel' sind in XSL mit der *key()* Funktion zugänglich und erleichtern das Suchen nach bestimmten Elementen. Die definierten 'Schlüssel' sind:

```
<!-- key fuer jeden index@entry Anfangsbuchstaben ********************* -->
<xsl:key name="initial-letter" match="index"
         use="translate(substring(@entry,1,1),
                        'abcdefghijklmnopqrstuvwxyz',
                        'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
<!-- key fuer jeden index@entry ************************************** -->
<xsl:key name="index-entry" match="index"
         use="@entry"/>
```

Durch die *translate()* Funktion werden alle Kleinbuchstaben zu Großbuchstaben.
Im nächsten Schritt werden zwei XSLT-Schablonen für das Index-Element erstellt. Eine Schablone, die sich um die Anfangsbuchstaben kümmert, und eine, welche die einzelnen Einträge erstellt. Um diese zwei XSLT-Schablonen zu unterscheiden, bekommen sie das Attribut *mode*, das beim Aufruf verwendet wird.

```
<xsl:template match="index" mode="index-letter">
<xsl:template match="index" mode="index-entry">
```

Ist im XML-Dokument das Kapitel 0.1.7 erreicht, erfolgt der Aufruf für die Erstellung der Anfangsbuchstaben:

```
<xsl:apply-templates select="//index[count(.|
                                 key('initial-letter',
                                   translate(substring(@entry,1,1),
                                     'abcdefghijklmnopqrstuvwxyz',
                                     'ABCDEFGHIJKLMNOPQRSTUVWXYZ'))[1])
                                 = 1]"
                     mode="index-letter">
  <xsl:sort select="translate(@entry, 'abcdefghijklmnopqrstuvwxyz',
                                      'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
```

Stylesheet: Aufruf von 'index' für die Anfangsbuchstaben (Quelle: Eigene Beschreibung mit Einsatz des Muenchian Algorithmus[2])

Die Anweisung *<xsl:sort>* sortiert alle Einträge alphabetisch. Die Aufrufe der angesprochenen Schablonen werden dadurch in alphabetischer Reihenfolge durchgeführt. Durch die *count() = 1* Anweisung wird sichergestellt, dass Aufrufe nur für Buchstaben erfolgen, die auch existieren. Die Existenz eines Buchstaben wird im *xsl:key* 'index-letter' festgestellt. Die zusätzliche Bedingung am Ende der *select*-Anweisung ([1]) stellt sicher, dass für jeden Buchstabe der Aufruf nur einmal erfolgt.
Nach der Ausgabe eines Anfangsbuchstabens, wird in der XSLT-Schablone mit dem *mode* 'index-letter' die zweite XSLT-Schablone aufgerufen (*mode* = „index-entry“), um die Einträge zum jeweiligen Buchstaben auszudrucken.

```
<xsl:apply-templates select="//index[count(.|
                                  key('index-entry', @entry)[1])=1]"
                                  mode="index-entry">
  <xsl:sort select="@entry"/>
  <xsl:with-param name="initial-letter" select="$key"/>
</xsl:apply-templates>
```

Stylesheet: Aufruf von 'index' für die Einträge (Quelle: Eigene Beschreibung)

Dieser Aufruf geschieht durch die Bedingung die in den eckigen Klammern steht ([1]) für jeden *index*-Eintrag nur einmal. Dadurch muss die aufgerufene Schablone nicht erneut auf doppelte Einträge prüfen.
Der Parameter 'initial-letter' wird bei diesem Aufruf übergeben. Dieser Parameter wird verwendet, um die zugehörigen Einträge zu finden.
Die Punktierung wird mit der Funktion *<fo:leader>* erstellt. Damit sich diese Punktierung in der Ausdehnung dynamisch verändert, muss darauf geachtet werden, dass im darüber liegenden *<fo:block>* Element das Attribut *text-align-last* mit dem Wert 'justify' gesetzt ist. Dieses Attribut sorgt dafür, dass der vorhandene Freiraum gefüllt wird.
Die Suche nach den zugehörigen Einträgen kann nun mit der Information des 'initial-letter' und der XSLT-Funktion *<xsl:for-each>* angestoßen werden. Für die Erstellung der *index*-Einträge ist der Quellcode in Listing A.11 hinterlegt.
In der Umsetzung sind zwei *<xsl:for-each>* Anweisungen zu finden. Mit der ersten *for-each* Anweisung werden zum aktuellen *index*-Element alle weiteren *index*-Elemente, die denselben Namen tragen, markiert. Danach werden alle Abschnittsnummern, in denen das aktuelle *index*-Element auftaucht, durch Kommata getrennt ausgegeben. Um doppelte Nummern zu vermeiden, wird in einer inneren Schleife das jeweils nächste Element gesucht und die Abschnittsnummern verglichen.

6.2.2. Tabellen (*<table>*)

CALS und FO Tabellenkonzepte[3] werden in der Literatur häufig besprochen[4]. Die Konvertierung von CALS Tabellen nach FO ist grundlegend unproblematisch. Die meisten Elemente aus dem CALS Tabellenmodell lassen sich direkt auf die Elemente der FO Tabellen abbilden.
Bei der Umsetzung stößt man jedoch mit XSLT auf ein großes Problem, welches in diesem Abschnitt besprochen werden soll.

Problembeschreibung

Die äußeren Tabellenrahmen einer Tabelle werden durch das im Element *<table>* enthaltene Attribut *frame* beschrieben. Wird hier ein entsprechender Wert hinterlegt, werden in der Ausgabe die äußeren Rahmen dargestellt. Für die inneren Tabellenrahmen dienen die Attribute *colsep* und *rowsep*, die in den einzelnen Einträgen (*<entry>*) oder in den Elternelementen (bis hin zu *<table>*) gesetzt werden können. *colsep* und *rowsep* haben die Werte 0 oder 1. Eine 1 steht für dargestellte Rahmen; 0 für nicht dargestellte Rahmen. Die Umsetzung von *colsep* und *rowsep* ist wie folgt definiert:

- *colsep:*
  colsep beschreibt die Trennlinie für die Spaltenabgrenzung. Es soll bei der Umsetzung nur die jeweils rechte Abgrenzung gezeichnet werden, damit es am linken Tabellenrand und bei nacheinander folgenden Zellen nicht zu doppelten Begrenzungslinien kommt. Dabei muss diese Behandlung jeweils bei der letzten Spalte einer Zeile ignoriert werden, da in diesen Fällen der äußere Tabellenrahmen erreicht ist, welcher an anderer Stelle (nämlich in *<table>*) bereits definiert ist.
- *rowsep*
  rowsep beschreibt die Trennlinie für die Zeilenabgrenzung. Es soll bei der Umsetzung nur die jeweils untere Abgrenzung gezeichnet werden, damit es am oberen Tabellenrand und bei untereinander folgenden Zellen nicht zu doppelten Begrenzungslinien kommt. Dabei muss diese Behandlung jeweils bei der letzten Zeile der Tabelle ignoriert werden, da in diesem Fall wieder der äußere Tabellenrahmen erreicht ist.

Um die Separierung der Spalten (*colsep*) umzusetzen ist es also nötig herauszufinden, wann der letzte Eintrag einer Zeile erreicht ist. Dies verursacht Probleme, wenn mit Zeilenüberspannungen (vertikales Verbinden mehrerer Zellen) gearbeitet wird. Hierzu ein kleines Beispiel:

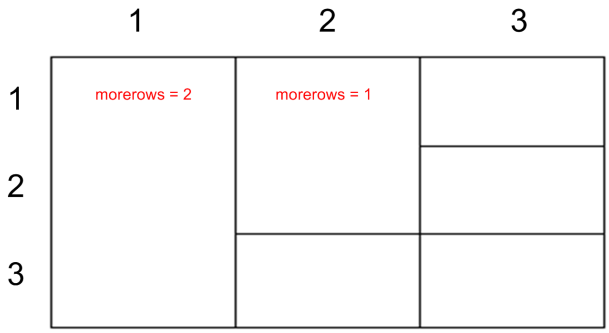


Abbildung 6.2.: Beispiel einer Tabelle mit Attribut *morerows* (Quelle: Eigene Darstellung)

[3] siehe Kap. 4.1.1 und Kap. 4.2.2, Seite 38
[4] siehe z.B. [KP04, Kap. 10.13]

In der ersten Zeile ist bei den ersten beiden Zellen das Attribut *morerows* gesetzt. Dadurch besteht die 2. Zeile nur aus einem Eintrag (**ein** *<entry>* Element) und die 3. Zeile aus zwei, statt drei Einträgen. D.h., dass bei Zeile 2 bereits das erste *<entry>*-Element der letzte Eintrag ist. Arbeitet man nun ohne Zeilenspezifikationen (*<colspec>*), in denen die Spaltennummern hinterlegt sind, müssen alle darüber liegenden Zeilen auf das Attribut *morerows* untersucht werden, um zu erkennen, dass dies der letzte Eintrag der Zelle ist und somit die Spaltenseparierung ignoriert werden muss.
Diese Untersuchung der Zellenverteilung ist über XSLT nicht, bzw. nur mit einem sehr komplexen rekursiven Algorithmus lösbar, der das Programm bei großen Tabellen höchstwahrscheinlich überlasten würde. Das Problem ist, dass bei XSLT keine Daten zwischengespeichert werden können und somit die darüber liegenden Tabellenzeilen rekursiv aufgerufen werden müssten, um ihnen Informationen über Parameter mitzugeben. Eine Lösung dieses Problems über XSLT ist bis dato nicht auffindbar.

Lösung über Java

Ein Lösungsweg des im vorherigen Abschnitt erwähnten Problems wird im Folgenden behandelt. Über eine Vorverarbeitung der XML-Quelldaten mit der Programmiersprache Java, wird für jedes *<entry>*-Element die zugehörige Spaltennummer in einem zusätlichen Attribut definiert. Ist in jedem Eintrag der Tabelle die Spaltennummer hinterlegt, kann in XSLT ohne Probleme die jeweils letzte Zelle einer Zeile bestimmt werden.
In Java gibt es 2 Möglichkeiten bei der Arbeit mit XML-Daten. Die Verarbeitung mit DOM oder mit SAX. Beide Modelle seien kurz erläutert:

- DOM (Document Objet Model):
  Eine vom W3C definierte Programmierschnittstelle für den Zugriff auf HTML- oder XML-Dokumente. Der XML-Zugriff erfolgt über einen Objektbaum, was komfortabler, aber für große Dokumente ungeeignet ist.
- SAX (Simple API for XML):
  Ebenfalls eine vom W3C definierte Programmierschnittstelle. Ein XML-Dokument wird sequentiell durchlaufen. Dadurch eignet es sich auch für große Dokumente. Ereignisgesteuert werden Callbackfunktionen aufgerufen, mit welchen der Aufbau der XML-Dokumente verändert werden kann.

Zusätzlich zum Hinzufügen der Spaltennummern sollen mit der Java Vorverarbeitung fehlende *<entry>*-Elemente aus der XML-Quelle mit leeren Einträgen gefüllt werden. FO bricht sonst bei Tabellen mit einer ungültigen Anzahl an Zellen das Layout ab und macht die Tabelle unbrauchbar.
Da zur Untersuchung der Zellenverteilung mehrfach auf Elemente und Attribute in verschiedenen Hierarchieebenen zugegriffen werden muss, wird für das Modifizieren der Tabellen die Programmierschnittstelle DOM eingesetzt. Um die Arbeitsgeschwindigkeit zu erhöhen, soll die Java Vorverarbeitung folgenden Aufbau haben:

1. XML-Dokument einlesen
2. Tabellen suchen und in einer Liste speichern
3. 1. Tabelle markieren
4. aktuelle Tabelle kopieren (mit der kompletten Struktur)
5. kopierte Tabelle bearbeiten
6. bearbeitete Tabelle zurückschreiben

7. nächste Tabelle markieren
8. weiter bei 4. bis letzte Tabelle erreicht

Um die Zellenverteilung der Tabellen darzustellen, wird eine Matrix mit *x* Zeilen und *y* Spalten erstellt. *x* entspricht der Anzahl der Tabellenzeilen; *y* der Anzahl der Tabellenspalten. In dieser Matrix werden gesetzte Zellen der Tabelle nach Untersuchung der *<entry>*-Elemente aus der XML-Quelle markiert.
Für die Umsetzung in Java wurde ein Pseudo-Code entwickelt. Dieser Code wird in Listing A.12 beschrieben. Der komplette Java Quellcode ist auf der beigelegten CD zu finden.

7. Zusammenfassung und Bewertung

In der vorliegenden Arbeit wird der Einsatz von Desktop-Publishing-Systemen in Unternehmen zur automatisierten Erzeugung eines Papierlayouts aus XML-Daten untersucht.
Zunächst werden die Grundlagen von XML und des automatisierten Desktop-Publishings erläutert.
Im 3. Kapitel wird das Umfeld beschrieben, in dem das Desktop-Publishing eingesetzt werden soll. Das zu entwickelnde DTP-System findet seine Verwendung im Unternehmen der Lufthansa Systems AG, das bereits auf der Grundlage von XML ein umfassendes Dokumentenmanagementsystem entwickelt hat. Der Ablauf des Dokumentenmanagementsystems spielt für die Umsetzung dieser Arbeit eine untergeordnete Rolle. Der Fokus liegt bei den XML-Dokumenten, die aus dem Dokumentenmanagementsystem der LSY kommen. Diese XML-Dokumente weisen die typische Struktur von Büchern auf.
Nach einer umfassenden Analyse, in der die Anforderungen gesammelt, sowie die drei DTP Lösungen LaTeX, XSL-FO und Adobe® FrameMaker® untersucht werden, fällt die Wahl zur Umsetzung des DTP-Systems aufgrund des größten Automatisierungsgrades auf XSL-FO. Außerdem ergeben sich aus der Analyse mit den vorliegenden Anforderungen keine Einschränkungen für den Einsatz von XSL-FO.
Für die Verarbeitung der FO Dokumente in das Ausgabeformat PDF wird nach einer weiteren Analyse das Produkt Antenna House XSL Formatter ausgewählt.
Das im Kapitel 5 entworfene System hat das Ziel, den Einsatz dieses Systems auch für zukünftige Anforderungen zu ermöglichen. Der vorliegende Entwurf ermöglicht die Entwicklung eines Codes, der an vielen Stellen allgemein gehalten und somit für die Erstellung verschiedener Dokumente genutzt werden kann.
Die Umsetzung des DTP-Systems zeigt, dass der Einsatz von XSL-FO und der Transformationssprache XSLT für das Publizieren von Fließtextdokumenten, auch größeren Umfangs, geeignet ist. Defizite wurden dabei nur bei der Erstellung des Indexes, sowie bei der Verarbeitung von Tabellen festgestellt. Die entstandenen Probleme wurden in dieser Arbeit behandelt und eine Lösung des Problems erarbeitet. Bei der Umsetzung der Tabellen eignete sich das CALS-Tabellenmodell nur bedingt, da dieses Modell nicht sauber objektorientiert arbeitet (wichtige Informationen sind nicht im Objekt hinterlegt, sondern müssen durch Algorithmen bestimmt werden). Die Realisierung dieses Elements wurde durch eine Vorverarbeitung über die Programmiersprache Java ermöglicht.

Dokumente im Buchformat, die hauptsächlich aus Fließtext ohne sonderlich kompliziertem Layout bestehen, lassen sich mit XSL-FO problemlos umsetzen. Müssen Elemente umgesetzt werden, die in Büchern standardmäßig nicht eingesetzt werden, sollten die Anforderungen neu gesammelt und auf eine Realisierbarkeit mit XSL-FO untersucht werden. Das Erzeugen von grafischen Elementen lässt XSL-FO beispielsweise an seine Grenzen stoßen, da XSL-FO nur das Erzeugen horizontaler Linien anbietet.

8. Ausblick

Bei der Realisierung des XML-basierten Desktop-Publishing-Systems wurde auf eine mögliche Übertragbarkeit der Lösung auf zukünftige Dokumente, mit ähnlichen Anforderungen, geachtet. Insbesondere durch die Aufteilung der Umsetzung in die verschiedenen Module (siehe Kapitel 5.2) können weitere Dokumente, die derselben DTD folgen, mit wenig Aufwand in das System eingebunden werden. Kleine Abweichungen im Layout der verschiedenen Dokumente können durch Variablen gesteuert werden. Größere Abweichungen können in eigenen Stylesheets die allgemeingültigen Beschreibungen überschreiben. Die Dokumente können zusätzlich durch die Beschreibung neuer Seitenvorlagen ein komplett neues Aussehen erhalten.
Die strenge Aufteilung der Module bewirkt eine leichte Erweiterbarkeit des Systems, was z.B. durch Erweiterungen der DTD notwendig wird. So können weitere Module zusätzlich hinzugefügt und, je nach Einsatzwunsch, eingebunden werden.
Weiterhin kann durch die Bereitstellung der Programmierschnittstellen des Antenna House XSL Formatter's (Java, .NET, etc.) das Desktop-Publishing-System an das Ende eines Dokumentenmanagementsystems integriert werden. Ist die Generierung der XML-Dokumente durch das Dokumentenmanagementsystem abgeschlossen, kann automatisch der entsprechende Prozess zur Layout-Erzeugung angestoßen werden. Dies ermöglicht einen komplett automatisierten Ablauf von der Erstellung der XML-Dokumente bis hin zum Ausgabedokument im Papierlayout.

A. Quellcodes

```
1  <!-- Layout der rechten Seite ***************************-->
2  <fo:simple-page-master master-name="right-page"
3       page-height="297mm" page-width="210mm"
4       margin-top="6mm" margin-bottom="5mm"
5       margin-left="22mm" margin-right="8mm"
6       writing-mode="lr-tb">
7  <fo:region-body
8       margin-top="27mm" margin-bottom="9mm"
9       margin-left="3mm" margin-right="4mm"/>
10 <fo:region-before region-name="header"
11      extent="22mm"/>
12 <fo:region-after region-name="footer"
13      extent="8mm"/>
14 </fo:simple-page-master>
15
16 <!-- Layout der linken Seite ***************************-->
17 <fo:simple-page-master master-name="left-page"
18      page-height="297mm" page-width="210mm
19      margin-top="6mm" margin-bottom="5mm"
20      margin-left="9mm" margin-right="21mm"
21      writing-mode="lr-tb">
22 <fo:region-body
23      margin-top="27mm" margin-bottom="9mm"
24      margin-left="3mm" margin-right="4mm"/>
25 <fo:region-before region-name="header"
26      extent="22mm"/>
27 <fo:region-after region-name="footer"
28      extent="8mm"/>
29 </fo:simple-page-master>
30
31 <!-- Layout der left blank-page ***************************-->
32 <fo:simple-page-master master-name="left-blank-page"
33      page-height="297mm" page-width="210mm"
34      margin-top="6mm" margin-bottom="5mm"
35      margin-left="9mm" margin-right="21mm"
36      writing-mode="lr-tb">
37 <fo:region-body/>
38 <fo:region-before region-name="header-blankpage"
39      extent="22mm"/>
40 <fo:region-after region-name="footer"
41      extent="8mm"/>
42 </fo:simple-page-master>
```

Listing A.1: FO Aufbau der FSM Seitenvorlagen (Quelle: Eigene Beschreibung)

```
<fo:page-sequence-master master-name="chapter">
    <fo:repeatable-page-master-alternatives>
        <fo:conditional-page-master-reference
            master-reference="right-page"
            odd-or-even="odd"
            blank-or-not-blank="not-blank"/>
        <fo:conditional-page-master-reference
            master-reference="left-page"
            odd-or-even="even"
            blank-or-not-blank="not-blank"/>
        <fo:conditional-page-master-reference
            master-reference="left-blank-page"
            blank-or-not-blank="blank"/>
    </fo:repeatable-page-master-alternatives>
</fo:page-sequence-master>
```

Listing A.2: FO Aufbau der FSM Seitenabfolge (Quelle: Eigene Beschreibung)

```
<xsl:template match="general-info | chapter">
  <fo:page-sequence master-reference="chapter"
      force-page-count="end-on-even" initial-page-number="1">

    <!-- Kopfzeile mit Inhalt fuellen ********************-->
    <fo:static-content flow-name="header">
        ...
    </fo:static-content>

    <!-- Fusszeile mit Inhalt fuellen ********************-->
    <fo:static-content flow-name="footer">
        ...
    </fo:static-content>

    <!-- Kopfzeile fuer blank page mit Inhalt fuellen ******-->
    <fo:static-content flow-name="header-blankpage">
        ...
    </fo:static-content>

    <!-- Body mit Inhalt fuellen ************************ -->
    <fo:flow flow-name="xsl-region-body">
        <fo:block hyphenate="true" language="{@language}"
            font-family="{$normal.font-family}"
            font-size="{$normal.font-size}">

            ...

            <xsl:apply-templates/>
        </fo:block>

    </fo:flow>
  </fo:page-sequence>
</xsl:template>
```

Listing A.3: *static*- und *flow*-content (Quelle: Eigene Beschreibung)

```
<fo:table width="180mm">
  <fo:table-body>
    <!-- 1. Zeile: Begrenzungslinie -->
    <fo:table-row>
      <fo:table-cell number-columns-spanned="2">
        <fo:block padding-bottom="1.1mm" font-size="0pt">
          <fo:leader leader-length="100%" leader-pattern="rule"
                     rule-style="solid" rule-thickness="1pt"
                     color="black" />
        </fo:block>
      </fo:table-cell>
    </fo:table-row>

    <fo:table-row>
      <fo:table-cell padding-left="3mm">
        <fo:block font-family="{$normal.font-family}" font-size="11pt">
          &#xA9; Lufthansa Flight Training GmbH
        </fo:block>
      </fo:table-cell>

      <fo:table-cell text-align="right" padding-right="3mm">
        <fo:block font-family="Lufthansa" font-size="14pt" color="#FFB300">
          a
        </fo:block>
      </fo:table-cell>
    </fo:table-row>
  </fo:table-body>
</fo:table>
```

Listing A.4: Umsetzung des Footers für das FSM (Quelle: Eigene Beschreibung)

```
1  <xsl:template match="general-info | chapter">
2
3    <fo:page-sequence master-reference="chapter"
4            force-page-count="end-on-even" initial-page-number="1">
5
6      <!-- static-content fuellen ****************************** -->
7      <xsl:call-template name="fill-static-content">
8        <xsl:with-param name="chapterNr" select="@fixnum"/>
9        <xsl:with-param name="caption3" select="section[1]/title"/>
10     </xsl:call-template>
11
12     <!-- Body mit Inhalt fuellen ****************************** -->
13     <fo:flow flow-name="xsl-region-body">
14       <fo:block ...>
15           ...
16         <xsl:apply-templates/>
17       </fo:block>
18     </fo:flow>
19   </fo:page-sequence>
20 </xsl:template>
```

Listing A.5: Nutzung von XSLT-Schablonen am Beispiel von 'fill-static-content' (Teil 1) (Quelle: Eigene Beschreibung)

```
1  <xsl:template name="fill-static-content">
2    <xsl:param name="chapterNr"/>
3    <xsl:param name="caption3"/>
4
5    <!-- Kopfzeile mit Inhalt fuellen ************************** -->
6    <fo:static-content flow-name="header">
7      <xsl:call-template name="header">
8        <xsl:with-param name="chapterNr" select="$chapterNr"/>
9        <xsl:with-param name="caption3" select="$caption3"/>
10     </xsl:call-template>
11   </fo:static-content>
12
13   <!-- Fusszeile mit Inhalt fuellen ************************** -->
14   <fo:static-content flow-name="footer">
15     <xsl:call-template name="footer"/>
16   </fo:static-content>
17
18   <!-- Kopfzeile fuer blank page mit Inhalt fuellen ********** -->
19   <fo:static-content flow-name="header-blankpage">
20     <xsl:call-template name="header-blankpage">
21       <xsl:with-param name="chapterNr" select="$chapterNr"/>
22       <xsl:with-param name="caption3" select="$caption3"/>
23     </xsl:call-template>
24   </fo:static-content>
25 </xsl:template>
```

Listing A.6: Nutzung von XSLT-Schablonen am Beispiel von 'fill-static-content' (Teil 2) (Quelle: Eigene Beschreibung)

```
1  <xsl:template match="unumlist">
2    <fo:list-block provisional-distance-between-starts="6mm"
3                   provisional-label-separation="0mm">
4      <xsl:choose>
5      <!-- wenn unumlist nicht Kind einer unumlist ist(also List auf
6        -- Level 1), dann Abstand nach unten setzen ******************* -->
7        <xsl:when test="not(parent::unumlitm)">
8          <xsl:attribute name="margin-bottom">
9            <xsl:value-of select="$unumlist.margin-bottom"/>
10         </xsl:attribute>
11       </xsl:when>
12       <!-- otherwise Abstaende oben und unten 0pt! ****************** -->
13       <xsl:otherwise>
14         <xsl:attribute name="margin-bottom">0pt</xsl:attribute>
15       </xsl:otherwise>
16     </xsl:choose>
17     <xsl:apply-templates/>
18   </fo:list-block>
19 </xsl:template>
20
21 <xsl:template match="unumlitm">
22   <fo:list-item>
23     <!-- LABEL DER UNUMLIST ***************************************** -->
24     <fo:list-item-label end-indent="label-end()">
25       <fo:block font-family="{$unumlist.label.font-family}">
26         <xsl:value-of select="$unumlist.label.character"/>
27       </fo:block>
28     </fo:list-item-label>
29
30     <!-- INHALT DER UNUMLIST **************************************** -->
31     <fo:list-item-body start-indent="body-start()">
32       <xsl:apply-templates/>
33     </fo:list-item-body>
34   </fo:list-item>
35 </xsl:template>
```

Listing A.7: Stylesheet zur Beschreibung von *<unumlist>* (Quelle: Eigene Beschreibung)

```
1  <xsl:template name="createTLtitle">
2    <xsl:param name="start-distance" select="'15mm'"/>
3    <xsl:param name="margin-bottom" select="'0mm'"/>
4    <xsl:param name="margin-top" select="'0mm'"/>
5    <xsl:param name="font-size" select="$normal.font-size"/>
6    <xsl:param name="font-weight" select="'normal'"/>
7    <xsl:param name="line-height" select="$normal.font-size"/>
8    <xsl:param name="baseline-shift" select="'0pt'"/>
9
10   <fo:list-block provisional-distance-between-starts="$start-distance"
11                  provisional-label-separation="0mm"
12                  margin-bottom="$margin-bottom"
13                  margin-top="{$margin-top}">
14     <fo:list-item>
15     <!-- LABEL DES TITLES ************************************ -->
16     <fo:list-item-label end-indent="label-end()" >
17       <fo:block font-size="{$font-size}" font-weight="{$font-weight}"
18                 line-height="{$line-height}"
19                 line-height-shift-adjustment="disregard-shifts">
20         <fo:inline baseline-shift="{$baseline-shift}">
21           <xsl:call-template name="fixnum"/>
22         </fo:inline>
23       </fo:block>
24     </fo:list-item-label>
25
26     <!-- INHALT DES TITLES ************************************ -->
27     <fo:list-item-body start-indent="body-start()" >
28       <fo:block font-size="{$font-size}" font-weight="{$font-weight}"
29                 line-height="{$line-height}"
30                 line-height-shift-adjustment="disregard-shifts">
31         <fo:inline baseline-shift="{$baseline-shift}">
32           <xsl:apply-templates/>
33         </fo:inline>
34       </fo:block>
35     </fo:list-item-body>
36     </fo:list-item>
37   </fo:list-block>
38 </xsl:template>
```

Listing A.8: Stylesheet zur Beschreibung von *<title>* in FO (Quelle: Eigene Beschreibung)

```
<xsl:template name="toc">
  <xsl:param name="act_chapter" select="."/>
  <xsl:variable name="chapNr" select="$act_chapter/@fixnum"/>

  <fo:table width="175mm" margin-top="11pt">
    <fo:table-column column-number="1" column-width="21mm"/>
    <fo:table-column column-number="2" column-width="144mm"/>
    <fo:table-column column-number="3" column-width="10mm"/>
    <fo:table-body>
      <xsl:for-each select="$act_chapter/section | $act_chapter/glossary">
        <xsl:variable name="act_section_nr" select="@fixnum"/>

        <!-- Wenn nicht erste section dann leerzeile einfuegen -->
        <xsl:if test="position()!=1">
          <fo:table-row>
            <fo:table-cell number-columns-spanned="3" height="11pt"/>
          </fo:table-row>
        </xsl:if>

        <xsl:call-template name="toc-entry">
          <xsl:with-param name="font-weight">
            <xsl:value-of select="'bold'"/>
          </xsl:with-param>
        </xsl:call-template>

        <!-- Eine section ausgedruckt. l1items zur zugehoerigen
          -- section ausdrucken -->
        <xsl:choose>
          <xsl:when test="$chapNr = '0'">
            <xsl:for-each select="//general-info/
                                  section[@fixnum = $act_section_nr]/*/l1item">
              <xsl:call-template name="toc-entry"/>
            </xsl:for-each>
          </xsl:when>
          <xsl:otherwise>
            <xsl:for-each select="//chapter[@fixnum = $chapNr]/
                                  section[@fixnum = $act_section_nr]/*/l1item">
              <xsl:call-template name="toc-entry"/>
            </xsl:for-each>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
    </fo:table-body>
  </fo:table>
</xsl:template>
```

Listing A.9: Stylesheet: Inhaltsverzeichnis (Quelle: Eigene Beschreibung)

```
<xsl:template name="fixnum">
  <xsl:if test="ancestor-or-self::*[self::general-info or self::chapter]">
    <xsl:value-of select="ancestor-or-self::*[self::general-info
                                              or self::chapter]/@fixnum"/>
  </xsl:if>
  <xsl:if test="ancestor-or-self::section">
    <xsl:text>.</xsl:text>
    <xsl:value-of select="ancestor-or-self::section/@fixnum"/>
  </xsl:if>
  <xsl:if test="ancestor-or-self::l1item">
    <xsl:text>.</xsl:text>
    <xsl:value-of select="ancestor-or-self::l1item/@fixnum"/>
  </xsl:if>
  <xsl:if test="ancestor-or-self::l2item[@fixnum]">
    <xsl:text>.</xsl:text>
    <xsl:value-of select="ancestor-or-self::l2item/@fixnum"/>
    <xsl:if test="ancestor-or-self::l3item[@fixnum]">
      <xsl:text>.</xsl:text>
      <xsl:value-of select="ancestor-or-self::l3item/@fixnum"/>
      <xsl:if test="ancestor-or-self::l4item[@fixnum]">
        <xsl:text>.</xsl:text>
        <xsl:value-of select="ancestor-or-self::l4item/@fixnum"/>
      </xsl:if>
    </xsl:if>
  </xsl:if>

  <!-- glossary -->
  <xsl:if test="ancestor-or-self::glossary">
    <xsl:text>.</xsl:text>
    <xsl:value-of select="ancestor-or-self::glossary/@fixnum"/>
  </xsl:if>
</xsl:template>
```

Listing A.10: XSLT: Erstellen der Abschnittsnummer (Quelle: Eigene Beschreibung)

```
<xsl:template match="index" mode="index-entry">
  <xsl:param name="initial-letter"/>
  <xsl:variable name="entry-name" select="@entry"/>

  <xsl:if test="starts-with($entry-name,$initial-letter)">
    <fo:block text-align-last="justify">
      <xsl:value-of select="$entry-name"/>
      <fo:inline keep-together="always">
        <fo:leader leader-pattern="dots" leader-pattern-width="6pt"/>

        <xsl:for-each select="//index[@entry=$entry-name]">
          <xsl:variable name="position" select="position()"/>
          <xsl:variable name="pre-fixnum">
            <xsl:call-template name="fixnum"/>
          </xsl:variable>

          <xsl:if test="position()=1">
            <xsl:call-template name="fixnum"/>
          </xsl:if>

          <!-- innere Schleife. prueft ob Vorgaenger dieselbe fixnum hat -->
          <xsl:for-each select="//index[@entry=$entry-name]">
            <xsl:if test="position() = $position+1">
              <xsl:variable name="act-fixnum">
                <xsl:call-template name="fixnum"/>
              </xsl:variable>
              <xsl:if test="$act-fixnum != $pre-fixnum">
                <xsl:text>, </xsl:text>
                <xsl:call-template name="fixnum"/>
              </xsl:if>
            </xsl:if>
          </xsl:for-each>
        </xsl:for-each>
      </fo:inline>
    </fo:block>
  </xsl:if>
</xsl:template>
```

Listing A.11: Stylesheet: Generieren der Index-Einträge (Quelle: Eigene Beschreibung)

```
1  Dokument einlesen;
2  NodeList nltable = alle <table>-Elemente im Dokument;
3  for every table{
4    colcount = <tgroup/@cols>;
5    rowcount = <tbody>.countrows;
6    modify(new Table(rowcount, colcount), tbody);
7    //mit new Table(int, int) wird eine Matrix mit 'rowcount' rows und
8    //'colcount' columns erstellt.
9  }
10
11 modify(Table table, Element tbody){
12   //alle rows der Tabelle durchlaufen
13   for(int actrow = 0; actrow < tbody.countrows; actrow++){
14     NodeList nlEntries = alle <entry>-Elemente in <row>
15     for(int actcell = 0; actcell < nlEntries.getlength(); actcell++){
16       actElement = nlEntries.item(actcell);
17
18       1. if( actElement.getAttribute("spanname") != ""){
19            //Suche zugehörige <spanspec>
20            Element colspecst = //definiere colspec wo span beginnt
21            Element colspecend = //definiere colspec wo span endet
22
23            int colnumst = getColnum(colspecst);
24            int colnumend = getColnum(colspecend);
25
26            //Ausnahme: colnum ragt aus Tabelle heraus:
27            if(colnumst > table.getRow(actrow).size(){
28              //vergrößere 'Table'
29            }
30            actElement.setAttribute("column-number", colnumst);
31            for(int i = colnumst; i < colnumend; i++){
32              //setze Tabelleneinträge auf 'true'
33              table.getRow(actrow).getCell(i-1).setReserved();
34            }
35         }
36       2. else if(@namest is set){
37            //suche zugehörige colspec
38            int colnumst = getColnum(colspecst);
39            int colnumend = getColnum(colspecend);
40            //siehe oben
41         }
42       3. else if(@colname is set){
43            //suche colspec
44            int colnum = getColnum(colspec);
45            //Ausnahmebehandlung (s.o.)
46            actElement.setAttribute("column-number", colnum);
47            table.getRow(actrow).getCell(colnum-1).setReserved();
48         }
49       4. else{  //no Attribute is set
50            //Liste auf ersten freien Platz ab actcell suchen und die
51            //gefundene colnum setzen.
52            boolean freeEntryFound = false;
53            int colnum = 0;
54            for(int i = actcell; i < table.getRow(actrow).size
55                                        && !freeEntryFound; i++){
56              if(table.getRow(actrow).getCell(i).isReserved == false){
```

```
57                freeEntryFound = true;
58                colnum = i+1;
59              }
60            }
61            if(!freeEntryFound){
62              colnum = table.getRow(actrow).size()+1;
63              //Neue Zelle in der Table-List hinten anfügen
64            }
65            actElement.setAttribute("column-number",colnum);
66            table.getRow(actrow).getCell(colnum-1).setReserved();
67          }
68      }
69    }
70
71  ##########################################################################
72    In einem 2ten Durchlauf werden nun auf Grundlage der gefüllten
73    Table-Liste (Matrix) die Tabelle mit fehlenden <entry>-Elementen gefüllt.
74
75    for(actrow = 0; actrow < table.size(); actrow++){
76      NodeList nlEntries = alle <entry>-Elemente der aktuellen row.
77      for(int actcell = 0; actcell < table.getRow(actrow).size(); actcell++){
78        if(table.getRow(actrow).getCell(actcell).isReserved() == false){
79          Node nextnode = Suche Node der ab actcell am nächsten liegt.
80          if(nextnode found){
81            insertEntryBefore(nextnode, actcell+1);
82            table.getRow(actrow).getCell(actcell).setReserved();
83          }
84          else{
85            appendEntry(rowElement, actcell+1);
86            table.getRow(actrow).getCell(actcell).setReserved();
87          }
88        }
89      }
90    }
91  }
```

Listing A.12: Pseudocode der Java Tabellenvorverarbeitung (Quelle: Eigene Beschreibung)

B. CD-Inhalt

Dieser Bachelorarbeit ist eine CD beigelegt auf der die Quelltextdateien, eine Definition der Layout-Vorgaben, alle genutzten Internetquellen, sowie die Diplomarbeit als PDF-Datei zu finden sind.
Die Quelltextdateien die sich auf der CD befinden sind Eigentum der Lufthansa Systems und daher streng vertraulich zu behandeln. Sie dürfen in keinster Weise für den kommerziellen Einsatz reproduziert oder weitergegeben werden.

- Verzeichnis: <//>

  Im root-Verzeichnis liegt die Bachelorarbeit als PDF-Datei (“BA_kglugla_2007-08.pdf“).

- Verzeichnis: <//Quellen/>

  In diesem Verzeichnis befinden sich alle Internetquellen, die für die Anfertigung dieser Bachelorarbeit verwendet wurden.

- Verzeichnis: <//LSY/Doku/>

  Dieses Verzeichnis enthält die *layout-definition* mit den Layout-Vorgaben (“layout-definition.xls“).

- Verzeichnis: <//LSY/Code/>

  Dieses Verzeichnis enthält die im Rahmen der Bachelorarbeit entwickelten Quelltextdateien. In der Datei “readme.txt“ findet sich eine Anleitung zum Einsatz des entwickelten Systems.

Literaturverzeichnis

[Ado07a] ADOBE: *The Adobe FrameMaker 7.0 XML Cookbook*. User Guide. Stand: 20. Juli 2007. – http://www.adobe.com/devnet/framemaker/onlinemanuals.html

[Ado07b] ADOBE: *Adobe FrameMaker*. Webseite. Stand: 30. Juli 2007. – http://www.adobe.com/de/products/framemaker/

[Bra00a] BRADLEY, Neil: *The XML Companion*. Addison-Wesley, 2000. – ISBN 0–201–67486–6

[Bra00b] BRADLEY, Neil: *The XSL Companion*. Addison-Wesley, 2000. – ISBN 0–201–67487–4

[Fuh96] FUHRMANN, Beate: *Typographische Texte im World Wide Web*. Diplomarbeit. Februar 1996. – http://www11.informatik.tu-muenchen.de/publications/da/fuhrmann96/beschreib.html#TeX

[Hou07] HOUSE, Antenna: *Antenna House XSL Formatter*. Webseite. Stand: 30. Juli 2007. – http://www.antennahouse.com/

[Jür95] JÜRGENS, Manuela: *LaTeX - Fortgeschrittene Anwendungen*. Online-Dokumentation. 1995. – ftp://ftp.fernuni-hagen.de/pub/pdf/urz-broschueren/broschueren/a0279510.pdf

[Jür00] JÜRGENS, Manuela: *LaTeX - eine Einführung und ein bisschen mehr*. Online-Dokumentation. 2000. – ftp://ftp.fernuni-hagen.de/pub/pdf/urz-broschueren/broschueren/a0260003.pdf

[KP04] KRÜGER, Manfred ; PINEDA, Manuel M.: *XSL-FO in der Praxis*. dpunkt.verlag, 2004. – ISBN 3–89864–249–6

[Krü97] KRÜGER, Frank: *Einführung in das Desktop Publishing*. Webseite. Stand: 20. April 1997. – http://www.fask.uni-mainz.de/cafl/kurse/dtp/dtp-1.html

[LSY06] LSY: *Fachlicher Kick-off EFOM-2*. Präsentation. Stand: 1. Februar 2006. – CD:<//Quellen/Lufthansa Systems/EFOM1vs EFOM2.ppt>

[LSY07] LSY: *OPS-DOC EDD*. Stand: 12. Juni 2007. – CD:<//Quellen/Lufthansa Systems/opsdoc-edd.doc>

[MRU07] *mrunix.de, das Forum für Entwickler*. Internetforum. Stand: 5. Juli 2007. – http://www.mrunix.de/forums/archive/index.php/t-41209.html

[Oos04] VAN OOSTRUM, Piet: *Page layout in LATEX*. Artikel. Stand: 2. März 2004. – ftp://ctan.tug.org/tex-archive/macros/latex/contrib/fancyhdr/fancyhdr.pdf

[Paw02] PAWSON, Dave: *XSL-FO*. O'Reilly, 2002. – ISBN 0–596–00355–2

[Pro07] PROJECT, The Apache XML G.: *Apache FOP*. Webseite. Stand: 30. Juli 2007. – http://xmlgraphics.apache.org/fop/

[Rad07] RADÜNZ, Daniel: *Wie bringt man XML und TeX zusammen?* Vorlesungsskript. Stand: 30. Juni 2007. – www2.informatik.hu-berlin.de/~piefel/LaTeX-PS/V11-XML.pdf

[Ren07] RENDERX: *RenderX XEP*. Webseite. Stand: 30. Juli 2007. – http://www.renderx.com/reference.html

[SB00] SCHRÖDER, Oliver ; BOCK, Jutta: *PDF und Acrobat: Neue Wege in der Druckvorstufe*. Galileo Press, 2000. – ISBN 3–934358–12–8

[Sch07] SCHÖN, Dr. E.: *XML für Medientechnologien*. Vorlesungsskript. Stand: 12. August 2007. – http://www.tu-ilmenau.de/fakei/fileadmin/template/i/mt/XML/XML-Folien_Kap4.pdf

[tea07] PROJECT TEAM, LaTeX: *LaTeX project*. Webseite. Stand: 29. Juli 2007. – http://www.latex-project.org/

[TEC07] TECCHANNEL: *PDF aus XML und XSL:FO*. Webseite. Stand: 18. Juni 2007. – http://www.tecchannel.de/index.cfm?webcode=401792

[TEI07] *Kurs: XML Ausgabeerzeugung mit XSL*. Webseite. Stand: 12. Juni 2007. – http://www.teialehrbuch.de/Kostenlose-Kurse/XML-XSL

[Vis07] VISTAIR: *XSLT Table Code*. Webseite. Stand: 4. Juli 2007. – http://www.vistair.com/newtbl/XSLTableCode.html

[W3C07] W3C: *World Wide Web Consortium*. Webseite. 2007. – http://www.w3.org

[WHB01] WEITZEL, Tim ; HARDER, Thomas ; BUXMANN, Peter: *Electronic Business und EDI mit XML*. dpunkt.verlag, 2001. – ISBN 3–932588–98–3

[WIK07] WIKIPEDIA: *Wikipedia*. Webseite. Stand: Mitte 2007. – http://de.wikipedia.org/wiki/<Begriff>

Tabellenverzeichnis

Abbildungsverzeichnis

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 27. August 2007	
Ort, Datum	Unterschrift