Universität Karlsruhe (TH)

Contentmigration in heterogenen Content Management Systemen

Diplomarbeit am Institut für Theoretische Informatik
Fakultät für Informatik
Universität Karlsruhe (TH)

von

cand. inform.
Nicole Schmidt

Erstbetreuer:

Prof. Dr. rer. nat. P. Deussen
Dr.habil. Hartmut Barthelmeß

Zweitbetreuer:

Prof. Dr. P. Sanders

Tag der Anmeldung: 20.03.2006
Tag der Abgabe: 19.09.2006

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 19. September 2006

Für meine Oma.

Ich danke meinen Eltern, die mich meine ganze Studienzeit motiviert und unterstützt haben. Meinen Betreuern bei der Namics AG und an der Universität danke ich für ihre Hilfe, Zeit und guten Ratschläge.
Ich möchte mich auch bei allen Menschen bedanken, die mich wären meiner Studienzeit motiviert, inspiriert und unterstützt haben.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Zielsetzung der Arbeit 1

1.2 Gliederung der Arbeit 3

2 Grundlagen **5**

2.1 World Wide Web 5

2.1.1 URI 6

2.1.2 HTTP 6

2.2 Dokumentenbeschreibungssprache 7

2.2.1 HTML 7

2.2.1.1 HTML-Dokument 8

2.2.1.2 HTML-Head 10

2.2.1.3 HTML-Body 10

2.2.2 XHTML 10

2.2.3 HTML vs. XHTML 12

2.3 Skripte 13

2.4 Content Management System 14

2.4.1 Evolution der Content Management System 16

2.4.2 Vorteile beim Einsatz eines Content Management System . . . 16

2.5 Web - Structure - Mining 17

2.6 Fuzzy-Logik 18

2.6.1 Fuzzy-Filter 21

2.6.1.1 Fuzzifizierer 21

2.6.2 Linguistische Variablen 22

2.6.3 Fuzzy-Operatoren 23

2.6.4 Entscheidungslogik mit Regelbasis 23

2.6.5 Defuzzifizierer 24

2.7 Zusammenfassung 24

3 Analyse **25**

3.1 Begriffserklärung . . . 25

3.2 Contentmigration . . . 26

3.2.1 Migration der Seitenstruktur . . . 26

3.2.2 Migration des Seiteninhalts . . . 27

3.3 Anforderungen an das Gesamtsystem . . . 28

3.4 Verwandte Arbeiten . . . 29

3.4.1 Spider Architektur . . . 29

3.4.2 HTML parsen . . . 30

3.4.3 HTML - Tag Klassifikation . . . 31

3.4.4 Inhaltsidentifikation . . . 32

3.5 Verarbeitung von XML . . . 33

3.6 Zusammenfassung . . . 34

4 Automatisierung der Contentmigration **35**

4.1 Konzept des Gesamtsystems . . . 35

4.1.1 Spider - Modul . . . 36

4.1.2 Mapping - Modul . . . 39

4.1.2.1 Zuordnung mittels einer Fuzzy - Regelbasis . . . 40

4.1.2.2 Aufbau des mapped_content.xml . . . 40

4.1.3 Import - Modul . . . 42

4.1.3.1 Aufbau des target_structure.xml . . . 42

4.2 Zusammenfassung . . . 43

5 Umsetzung **45**

5.1 Spider - Modul . . . 47

5.2 Mapping - Modul . . . 50

5.3 Import - Modul . . . 56

5.4 Zusammenfassung . . . 57

6 Evaluierung **59**

6.1 Seitenmigration . . . 59

6.2 Seiteninhaltsmigration . . . 60

6.3 Import in das Zielsystem . . . 61

6.4 Mögliche Erweiterungen . . . 61

6.5 Zusammenfassung . . . 62

7 Fazit **63**

A Abkürzungen **65**

Literaturverzeichnis **67**

Abbildungsverzeichnis

2.1 Die Entwicklung von HTML und XHTML [Tolk03] 8
2.2 Ausgabe der HTML-Datei in einem Browser 9
2.3 Arbeitsweise clientseitiger Skripte 13
2.4 Arbeitsweise serverseitiger Skripte 14
2.5 Aufbau eines Web Content Management Systems [Oliv01] 15
2.6 Kostenentwicklung einer Website [Oliv01] 17
2.7 Web Mining 18
2.8 Vergleich einer Klassische Menge mit einer Fuzzy-Menge 19
2.9 Fuzzy-Controller 22

3.1 Seitenstruktur 27
3.2 Modul für die Inhaltszuordnung 28
3.3 URL Status nach [Heat02] 30
3.4 Aufbau einer Webseite 33
3.5 Inhaltsbaum mit Schwesterelementen 34

4.1 Schritte der Datenverarbeitung [Usam96] 36
4.2 Gesamtsystem 37
4.3 Spider-Modul 38

5.1 Verwendete Programmiersprachen 45
5.2 Gesamtsystem 46
5.3 Spidermodul 47
5.4 Erstellung des page_structure.xml 49
5.5 Contentatome und Subklassen 52
5.6 Fuzzy-Mengen für Texte 53
5.7 Fuzzy-Mengen für Bilder 54

1. Einleitung

1.1 Zielsetzung der Arbeit

Als 1989 Tim Berners-Lee ein Hypertext-System am CERN in Genf aufbaute um auf einfache Art und Weise Forschungsergebnisse mit seinen Kollegen auszutauschen wurden damit die ersten Schritte in Richtung World Wide Web (WWW) gemacht. Seit dieser Zeit ist das Internet zum ersten und globalen Informationssystem gewachsen. Es ist mittlerweile weltweit zugreifbar. Es werden auch nicht mehr nur Forschungsergebnisse über das WWW ausgetauscht. Im Gegenteil, im Internet finden sich Informationen zu nahezu jedem Thema, seien sie von Privatpersonen veröffentlicht oder von Unternehmen. Menschen pflegen Kontakte über das Internet und führen Diskussionen. E-Mail, Foren und Chats erfreuen sich großer Beliebtheit.

Beinahe jedes Unternehmen unterhält heute eine Webseite. Unternehmen vertreiben die eigenen Produkte über das Internet, bieten Produktinformationen und Kontaktdaten an. Viele Unternehmen betreiben die interne Kommunikation über ein Intranet oder die Kommunikation mit den Außenstellen über ein Extranet, wohinter die gleiche Technologie steckt, wie hinter dem WWW.

Investitionen in das Internet werden immer größer. In den letzten Jahren floss bereits ein großer Teil des für Printmarketingetats in das Internetmarketing. Es sind viele Versandhäuser entstanden, die keine kostspieligen Kataloge mehr drucken und versenden, sondern ihre Waren ausschließlich im Internet anbieten.

Kurzum, das Internet gewann und gewinnt zunehmend an Bedeutung. Durch die schnelleren Datenleitungen, die seit einiger Zeit auch für Privathaushalte erschwinglich sind, nimmt die Popularität von datenintensiven Internetanwendungen zu. Die Internetanwendungen unterscheiden sich im Bezug auf ihren Komfort immer weniger von lokal ablaufenden Desktopanwendungen. So werden Lesezeichen[1] (Bookmarks)

[1] http://www.del.icio.us

oder Bilder[2] oft online abgelegt und mit anderen geteilt.

Im Zuge dieser Entwicklung sind die (Web-) Content Management Systeme entstanden ((W)CMS). Sie erlauben es Internetauftritte auf einfache Weise zu pflegen und zu erweitern und dies ohne die Kenntnisse der Web-Technologien, wie HTML, CSS oder HTTP. Ist ein CMS erst einmal von einem Fachmann konfiguriert, kann jeder, der Grundkenntnisse im Umgang mit einem Computer hat, Webseiten des Internetauftritts editieren oder neu erstellen. Das Ergebnis passt sich Dank des CMS auch vom Aussehen nahtlos in die bereits vorhandene Internetpräsenz ein.

Die meisten Firmen und auch große Institutionen haben mittlerweile den Vorteil eines CMS entdeckt und setzen solche ein. Soll auf ein anderes CMS umgestiegen werden, dies kann aus den unterschiedlichsten Gründen der Fall sein: Das CMS ist veraltet und wird nicht mehr gepflegt, ein anderes CMS bietet Möglichkeiten, die das alte CMS nicht bietet, auf die man allerdings nicht verzichten möchte.

Möchte eine Firma auf ein neues CMS umsteigen, das bisherige Design und den Inhalt beibehalten, so hatte man bisher kaum eine andere Möglichkeit, als die Migration von Hand vorzunehmen. D.h. die Seiten müssen von einem Autor im neuen CMS angelegt und mit den benötigten Attributen versehen werden, daraufhin müssen die Inhalte der alten Webseiten, bestenfalls via Copy-and-Paste in die neuen Seiten kopiert werden. Diese Arbeit ist sehr aufwendig und eintönig.

Dieses Vorgehen bietet die Motivation für diese Arbeit. Es soll eine Möglichkeit finden, die Contentmigration soweit als möglich zu automatisieren um Zeit und Kosten der Migration zu verringern. Das Potential dieser Idee ist enorm, denn die Internetauftritte der Unternehmen werden immer umfangreicher. Internetauftritte bestehen nicht selten aus 10.000 einzelnen Webseiten oder mehr. Um eine Idee davon zu bekommen wie umfangreich die Webseiten wirklich sind bietet sich der YAHOO! Site Explorer[3] an.

Der Internetauftritt des Automobilherstellers Daimler-Chrysler[4] besteht beispielsweise aus 14.129 Seiten, der der Deutschen Post[5] aus 7.491 Seiten und der des Versandhauses Quelle[6] aus nicht weniger als 90.988 Seiten. Bei den angegebenen Seitenzahlen handelt es sich um die Anzahl der verlinkten, eigenständigen HTML-Dokumente.

Soll nun eine Internetpräsenz mit 10.000 Seiten in ein anderes CMS migriert werden, so müssen im neuen System 10.000 Seiten und deren Inhalte angelegt werden. Geht man davon aus, dass das Anlegen einer Seite eine Minute dauert, was einer sehr optimistischen Schätzung entspricht, dann dauert es $10.000 Minuten \approx 20 Manntage$

[2] http://www.flickr.com/
[3] http://sitexplorer.search.yahoo.com
[4] http://www.daimlerchrysler.de
[5] http://www.DeutschePost.de
[6] http://www.quelle.de

nur um die Seiten anzulegen.

Diese Arbeit beschäftigt sich mit den Möglichkeiten der Automatisierung der Contentmigration. Das Ausgangs- sowie das Zielsystem des Internetauftrittes sollen nicht relevant sein. Es werden die Fragen geklärt, wie kann der Vorgang der Contentmigration zerlegt werden und welche der resultierenden Teilaufgaben automatisiert können werden. Es wird eine Vorgehensweise entwickelt und implementiert. Als Zielsystem für die Implementierung wurde das CMS Typo3[7] gewählt, wobei es sich bei Typo3 um ein weitverbreitetes OpenSource CMS handelt.

1.2 Gliederung der Arbeit

Kapitel 2 enthält die theoretischen Grundlagen. In Kapitel 3 finden sich eine Analyse der Aufgabenstellung, sowie relevante vorausgegangene Arbeiten. Der Lösungsansatz wird in Kapitel 4 vorgestellt. In Kapitel 5 wird ein prototypisches System präsentiert, dass die vorgeschlagenen Automatisierungen implementiert. Der Content einer bestimmten Domain wird hierbei in eine vorkonfigurierte Typo3-Instanz importiert. In Kapitel 6 wird der Lösungsansatz bewertet. Kapitel 7 enthält das Fazit.

[7] http://typo3.org/

2. Grundlagen

In diesem Kapitel werden die für die nachfolgende Arbeit relevanten Grundlagen erläutert. Es werden die Dokumentenbschreibungssprachen HTML und XHTML vorgestellt und deren Unterschiede aufgezeigt, sowie der grundsätzliche Aufbau einer HTML Seite erläutert. Zum besseren Verständnis des Zuordnungsmoduls werden die Grundlagen der Fuzzy-Logik beschrieben.

2.1 World Wide Web

Das **W**orld **W**ide **W**eb wurde laut [Tim 94] entwickelt um ein Pool menschlichen Wissens zu sein. Es sollte vor allem den Wissenschaftlern am Kernforschungszentrum in Genf (CERN) erlauben ihre Gedanken, Ideen bezüglich gemeinsamer Forschungsprojekte zu teilen. Doch das WWW hat sich bald über die Grenzen des CERN hinaus verbreitet.

Das WWW ermöglicht es Dokumente auf anderen, entfernten Computern zu betrachten, nach Dokumenten zu suchen oder einfach Verweisen (Links) in Dokumenten zu folgen. Dokumente im WWW sind mit Hilfe der **H**yper**T**ext **M**arkup **L**anguage (HTML) strukturiert. Verweise können nicht nur auf HTML-Dokument zeigen, sondern auf alle möglichen anderen Ressourcen im WWW. Ein großer Vorteil des WWW ist und war, dass der Endbenutzer kein Wissen über die verschiedenen benutzten Protokolle besitzen muss.

Nach [Tim 94] besteht das WWW aus mehreren Komponenten:

- Dem Adresssystem, das es mit Hilfe der URI ermöglicht alle im Netz befindlichen Ressourcen zu adressieren.
- Das Netzwerkprotokoll HTTP, zur Übertragung von Daten (hauptsächlich Webseiten) über das Internet.
- Der Markupsprache HTML, die von jedem Browser interpretiert werden kann.

2.1.1 URI

Zur Identifizierung einer Ressource im WWW dient der **U**niform **R**essource **I**dentifier (URI). Sie werden zur Adressierung von Ressourcen im Internet und dort vor allem im WWW eingesetzt. URIs werden als Zeichenfolge in digitale Dokumente, vor allem in HTML-Dokumenten, eingebunden.

Der Aufbau eines URI:
`<schema> ://[ <username> [: <passwort>] @ ] <hostname>[: <portnummer>] [/ <url-pfad>]`

Der in der Umgangssprache viel bekanntere und gebräuchlichere **U**niform **R**esource **L**ocator (URL) bezeichnet eine Unterart des URIs. URLs identifizieren eine Ressource über ihren primären Zugriffsmechanismus und den Ort der Resource in Computernetzwerken. Der Name des URI-Schemas ist in der Regel vom verwendeten Netzwerkprotokoll abgeleitet. Praxisnaher Beispiele hierfür sind HTTP oder FTP: Beispiel: `http://www.uni-karlsruhe.de/Uni/img/news/personen_1.jpg`

URLs stellen die erste und häufigste Art von URIs dar. Daher werden sie häufig synonym verwendet.

2.1.2 HTTP

Das **H**ypertext **T**ransfer **P**rotocol (HTTP) ist ein Protokoll um Daten über ein Netzwerk zu übertragen. Das Haupteinsatzgebiet von HTTP ist das laden von Webseiten in einen Webbrowser.

HTTP ist ein zustandsloses Protokoll, d. h. dass die Verbindung nach erfolgreicher Übertragung nicht aufrechterhalten werden muss. Sollen anschließend noch weitere Daten übertragen werden, so muss erneut eine Verbindung aufgebaut werden.

HTTP ist nicht auf Hypertext beschränkt, sondern auch zum Austausch beliebiger Daten geeignet. HTTP kennt außerdem noch verschiedene Request-Methoden, die gebräuchlichste ist das GET-Request. Sie dient dazu Daten vom Server anzufordern. Ein anderes gebräuchliches Request ist das HTTP-Post-Request. Das POST-Request ähnelt dem GET-Request, es wird ein zusätzlicher Datenblock übermittelt, der sich aus Name/Wert-Paaren zusammensetzt. Diese Name/Wert-Paare stammen meist aus einem HTML-Formular. Prinzipiell können solche Name/Wert-Paare auch mittels eines GET-Requests übermittelt werden, indem sie an die URL angehängt werden.

2.2 Dokumentenbeschreibungssprache

Bereits 1992 wurde die erste Version von HTML in einem Dokument beschrieben. [Conn92]

Als Basis für die Entwicklung von HTML 2.0 (November 1995) diente die **S**tandard **G**eneralized **M**arkup **L**anguage (SGML). HTML 2.0 wurde mehrmals erweitert. Es entstanden HTML 3.2 (Januar 1997), HTML 4.0 (Dezember 1997) und HTML 4.01 (Dezember 1999). Die Entwicklung und die Beziehungen zwischen den verschiedenen Markupsprachen ist in Abbildung 2.1 dargestellt. Bei der Weiterentwicklung von HTML wurden der Sprache jeweils Tags und Attribute hinzugefügt.

Da die Anwendungen immer vielseitiger und spezieller wurden und damit auch die Anforderungen an HTML, hat man sich dazu entschlossen die 1996 entwickelte **E**xtensible **M**arkup **L**anguage (XML) mit der bereits existierenden Version 4.01 von HTML zu vereinen. Das Ergebnis war **E**xtensible **H**yper**T**ext **M**arkup **L**anguage 1.0 (XHTML) (Januar 2000). [Tolk03] Im Mai 2001 wurde XHTML 1.1 wurde es vom **W**orld **W**ide **W**eb **C**onsortium (W3C) zum Standard erklärt. XHTML 1.1 ist eine strikte Version von XHTML 1.0, die die Varianten Framset und Transitional nicht mehr enthält.

XHTML 2.0 befindet sich in Entwicklungsphase und wurde noch nicht zum Standard erkärt. Es beinhaltet einige neue Tags und sorgt für eine noch bessere Trennung zwischen Inhalt und Darstellung.

2.2.1 HTML

HTML ist die meistgenutzte Sprache des WWW. Sie ist keine Programmiersprache, sondern eine Dokumentenbeschreibungssprache. Sie besitzt weder Schleifen noch Subroutinen, die für Programmiersprachen charakteristisch sind. HTML beschreibt lediglich die Gliederung und die Struktur eines Dokumentes und nicht dessen Darstellung, auch wenn es häufig dazu benutzt wird.

Um ein Dokument zu gliedern wurde bei der Entwicklung von HTML davon ausgegangen, dass die Strukturen der Dokumente sich aus gemeinsamen Elementen, wie unter anderem Titel, Paragraphen, Tabellen, Listen, zusammensetzen. Für alle diese Möglichkeiten existieren Markierungen (Tags), die im Text gesetzt werden können und so einen Textteil als Titel, Tabelle, Liste oder ähnliches kennzeichnen. Alle Markierungen zusammen bilden die Auszeichnungssprache, auch Markup. [Tolk03]

Da HTML nur die Struktur und die Gliederung, nicht aber das Design einer Dokumentes festlegt, sieht HTML nicht auf allen Browsern gleich aus. Dennoch ist HTML für alles Bildschirmauflösungen und Farbtiefen geeignet und sogar Blinde können mit HTML arbeiten.

Um behinderten Menschen den Umgang mit dem Internet zu erleichtern wurden von der Web Accessibility Initiative (WAI) Richtlinien erstellt, wie HTML-Seiten

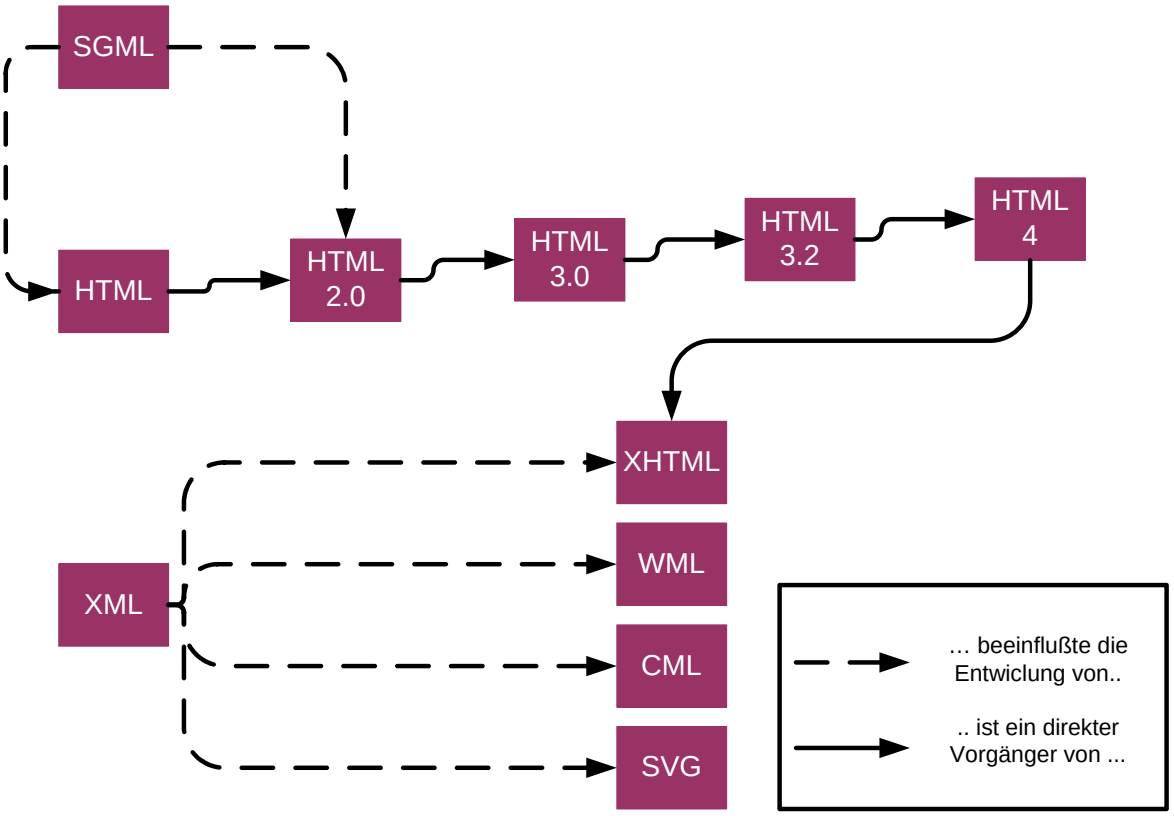

Abbildung 2.1: Die Entwicklung von HTML und XHTML [Tolk03]

strukturiert sein sollen[1]. Gerade für behinderte Menschen ist es wichtig, dass die angebotene Information über mehr als einen Kanal zugänglich ist.

2.2.1.1 HTML-Dokument

Ein HTML-Dokument ist ein Textdokument, das bestimmte Markierungen (Tags) enthält. Die Tags markieren Textteile und weisen ihnen Bedeutungen zu. Ein `<h1>`-Tag beispielsweise kennzeichnet den darin enthaltenen Text, als Überschrift der 1. Ordnung.

Um Textteile zu markieren gibt es öffnende Tags „`<tagname>`“ und schließende Tags „`</tagname>`“, die einen Text oder eine Menge anderer Tags beinhalten und dem Inhalt eine Semantik zuweisen. So kann beispielsweise durch den `<title>`-Tag der eingeschlossene Text als Titel des Dokuments gekennzeichnet werden. Nicht jeder Tag, der geöffnet wurde muss wieder geschlossen werden. Es gibt auch leere Tags, die als Platzhalter für gewisse Elemente fungieren. So z.B. der `<img>`-Tag, der für ein Bild im Dokument steht.

Tags können Attribute enthalten: `<tagname attributname =“attributwert“>`. Ein für diese Arbeit wichtiges Attribut ist beispielsweise, das href-Attribut des `<a>`-Tags (Verweis), es enthält die URL der Webseite auf die verwiesen wird. Ebenfalls von

[1] http://www.w3.org/WAI/

Bedeutung sind die Attribute des `<img>`-Tag, die Auskunft über den Speicherort des Bildes geben, sowie über die Anzeigegröße (Höhe und Breite).

Listing 2.1: Code HTML-Seite

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>Seitentitel</title>
        <meta name="description" content="Dies ist ein Beispiel">
    </head>
    <body>
        <!-- im Body-Element steht der angezeigte Seiteninhalt-->
        <h1>Beispielinhalt</h1>
        <img scr="bilder/bild.jpg">
        <hr>
        <ul>
            <li>Das erste Listenelement
            <li>Das zweite Listenelement
        </ul>
    </body>
</html>
```

In Listing 2.1 ist der Code einer HTML-Seite zu sehen. Daraus geht hervor, dass ein ein HTML-Dokument aus einem HTML-Head und einem HTML-Body besteht. Diese werden im Folgenden noch näher betrachtet.

Die Ausgabe der HTML-Seite im Browser ist in Abbildung 2.2 zu sehen. Auf der Webseite finden sich eine Überschrift erster Ordnung, ein Bild, gefolgt von einer horizontalen Linie und einer Liste, die zwei Elemente enthält.

Abbildung 2.2: Ausgabe der HTML-Datei in einem Browser

2.2.1.2 HTML-Head

Der HTML-Head (HTML-Kopf) wird im HTML-Dokument durch den <head>-Tag gekennzeichnet. Im HTML-Head stehen Daten, die browserintern verarbeitet werden oder von Suchmaschinen genutzt werden. [Tolk03]
Das einzige Element, das im Head enthalten sein muss, ist der Dokumententitel. [Ragg97] Der Titel ist einmalig für jedes HTML-Dokument.
Weitere für die Contentmigration interessante Elemente sind die Meta-Elemente. In den Meta-Elementen werden Metainformationen als Name/Wert-Paare bereitgestellt. [Ragg97]

Listing 2.2: Metadaten

```
<meta name = "description"
        content = "Universität Karlsruhe (TH),
                Homepage der Universität Fridericiana">
<meta name = "keywords"
        content = "Hochschule, Universität, Karlsruhe,
                Uni Karlsruhe, Technische Hochschule,
                University of Karlsruhe, Universite">
```

In Listing 2.2 sind die für die Contentmigration wichtigen Metainformationen dargestellt. Relevant sind vor allem die Metadaten die das Name-Attribut „description“ oder „keyword“ enthalten, da diese seitenspezifische Daten enthalten. Der Meta-Tag mit dem Name-Attribut „description“ enthält eine kurze Beschreibung des Inhalts der Webseite. Der Meta-Tag mit dem Name-Attribute „keyword“ enthält Schlagwörter die als Suchbegriffe für die Webseite dienen sollen.

2.2.1.3 HTML-Body

Im HTML-Body befindet sich der Teil des Dokuments, der im Browser angezeigt wird. Hier können alle HTML-Tags verwendet werden, die für die jeweilige HTML Version spezifiziert wurden. In Listing 2.1 handelt es sich um HTML 4.01. Daher sind die für HTML 4.01 spezifizierten Tags erlaubt.

2.2.2 XHTML

Wie in Abbildung 2.1 zeigt ist XHTML 1.0 eine Neuformulierung von HTML 4 unter Verwendung von XML [Tolk03]. Daher sind keine neuen Tags dazu gekommen oder entfernt worden.

Ein Vorteil von XHTML gegenüber HTML ist, dass die für XML entwickelten Werkzeugen und Programmbibliotheken eingesetzt werden können. Ein weiterer Vorteil von XHTML ist, dass die Dokumente mit Hilfe der zugehörigen **D**okument**t**y**d**definition (DTD) auf Gültigkeit geprüft werden können.

Es müssen syntaktische Grundregeln eingehalten werden, damit eine XHTML-Datei wirklich XHTML enthält. Um gültig zu sein, muss ein XML-Dokument folgende Kriterien erfüllen [Harr02]:

1. Der XHTML-Code muss syntaktisch korrekt sein.
2. Der XHTML-Code muss eine Dokumententyp-Deklaration besitzen.
3. Das Root-Element des XHTML-Codes muss von der Dokumenttyp-Deklaration definiert worden sein.
4. Der XHTML-Code muss alle Einschränkungen beachten, die von der DTD angezeigt und definiert werden.

Die in Listing 2.1 vorgestellte HTML-Seite wurde in XHTML (Listing 2.3) umcodiert.

Listing 2.3: Beispiel einer XHTML-Seite

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="de" xml:lang="de">
    <head>
        <title>Seitentitel</title>
        <meta name="description" content="Dies ist ein Beispiel" />
        <meta name="keyword" content="Beispiel" />
        <meta name="keyword" content="HTML" />
        <meta name="keyword" content="Aufbau" />
    </head>
    <body>
        <!-- im Body-Element steht der angezeigte Seiteninhalt-->
        <h1>Beispielinhalt</h1>
        <img scr="bilder/bild.jpg" />
        <hr />
        <ul>
            <li>Das erste Listenelement</li>
            <li>Das zweite Listenelement</li>
        </ul>
    </body>
</html>
```

Zwischen dem HTML-Code und dem XHTML-Code lassen sich Unterschiede erkennen. Der XHTML-Code enthält eine DTD. Es wird, wie es für jedes XML möglich ist, eine DTD angegeben, mit Hilfe derer das Dokument validiert werden kann. Die Angabe einer DTD ist zwingend erforderlich.

Für XHTML gibt es mehrere mögliche Angaben für die DTD. Die drei wichtigsten sind:

1. strict: Validiert sehr streng nach den XHTML-Kriterien.
2. transitional: Ist kompatibel zu älteren Versionen von HTML.
3. frameset: Erlaubt die Nutzung von Frames[2].

Ein XHTML-Dokument muss wohlgeformt sein. D.h. es muss grundlegende syntaktische Anforderungen erfüllen. Dazu gehört die korrekte durchgeführte Schachtelung der Elemente. Jeder Tag, der sich in einem anderen Tag befindet, muss geschlossen werden, bevor der übergeordnete Tag geschlossen wird (im obigen Beispiel (Listing 2.3) an den Listenelementen (<li>) zu sehen). Geöffnete Tags müssen wieder geschlossen werden, es sei denn, es handelt sich um Leerelement Tags. Leerelementtags, wie das Image-Element (<img>) oder die horizontale Linie (<hr>), müssen ebenfalls geschlossen werden. Tagnamen müssen durchweg klein geschrieben werden, Attribute müssen in Anführungszeichen stehen und dürfen nicht mehr abgekürzt werden. [Tolk03, Harr02]

2.2.3 HTML vs. XHTML

Sowohl HTML als auch XML und somit XHTML können durch eine kontextfreie Grammatik beschrieben werden. [John02] XHTML und HTML sind beide Dokumentenbeschreibungssprachen. Für die maschinelle Weiterverarbeitung ist XHTML besser geeignet als HTML, da es mit den gängigen für XML entwickelten Werkzeuge und Programmbibliotheken möglich ist XHTML zu verarbeiten, nicht aber HTML.

Ein weiteres Problem bei der Verarbeitung von HTML und XHTML ist, dass die gängigen Browser auch fehlerhafte HTML- oder XHTML-Dokumente korrekt darstellen. Viele Webseitenentwickler überprüfen daher ihren Code nicht auf Validität. So befinden sich HTML- und XHTML-Seiten im Netz, die nicht valide sind. Diese nicht validen HTML-Dokumente zu parsen ist schwierig, da ihre Struktur nur zum Teil vorhersehbar ist.

Daher müssen Parser, die HTML verarbeiten sollen, wie Browser, sehr fehlertolerant sein. Die Standard-HTML-Parser-Bibliotheken der verschiedenen Programmiersprachen sind das häufig nicht. Das bedeutet, dass sie für die Verarbeitung von HTML-Dokumenten aus dem Internet wenig geeignet sind.

[2]Frames sind HTML-Seiten in HTML-Seiten. Eine Technik, die vor einigen Jahren häufig verwendet wurde, heute aber nicht mehr häufig zu finden ist.

2.3 Skripte

Da sich mit HTML nur statische Webseiten erstellen lassen, kommen heute die unterschiedlichsten Skripsprachen im WWW zum Einsatz. Sie haben das Internet stark beeinflusst und dynamisiert. Ohne Skriptsprachen würde das Internet auch heute hauptsächlich aus statischen Seiten bestehen und viele der populären Internetanwendungen und -services wären nicht umsetzbar. Skriptsprachen sind Programmiersprachen und vor allem dazu gedacht kleine und überschaubare Programme zu schreiben. Sie verzichten auf komplexere Sprachelemente, wie die Deklaration von Variablen und Vererbung. Bei Skriptsprachen muss zwischen client- und serverseitigen Sprachen unterschieden werden.

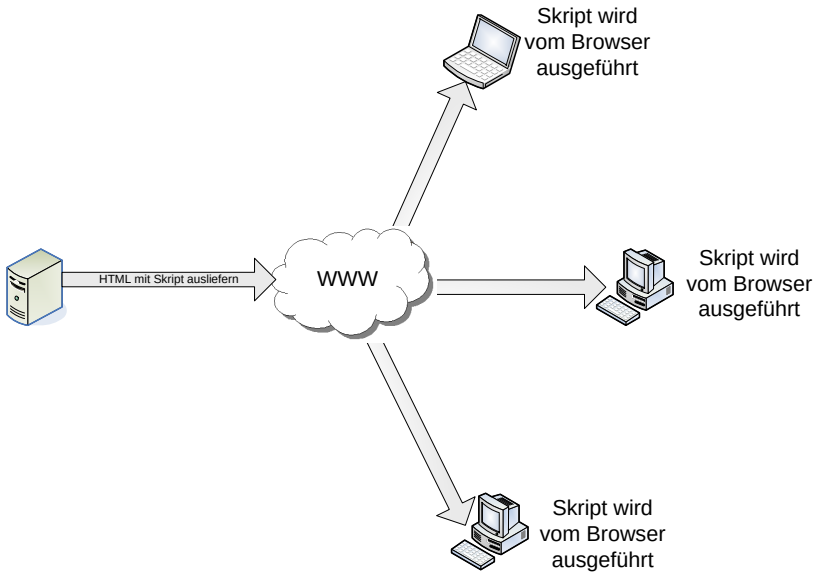

Abbildung 2.3: Arbeitsweise clientseitiger Skripte

Clientseitige Skriptsprachen werden in die auszuliefernden HTML-Seiten eingebunden und vom jeweilig benutzten Browser ausgeführt (siehe Abbildung 2.3). Im HTML-Code sind Skripte durch spezielle Skript-Tags gekennzeichnet.

Eine clientseitige Skriptsprache ist ECMAScript, deren Ausprägung „JavaScript“ wird von allen gängigen Browsern unterstütz außer dem Internet Explorer unterstützt. Der Internet Explorer unterstützt stattdessen JScript, eine clientseitige Scriptsprache, die JavaScript so ähnlich ist, dass sie in der Praxis nicht unterschieden werden. Der Internet Explorer unterstützt außerdem noch VBScript.

Die Auswahl der serverseitigen Sprachen ist weit größer. Heute hauptsächlich im Einsatz sind die Cold Fusion Markup Language, Perl, PHP, Python, Ruby und ASP.

Serverseitige Sprachen werden vom Webserver ausgeführt. Für den Client, der die entsprechende Seite angefordert hat, wird eine HTML-konforme Datei generiert und

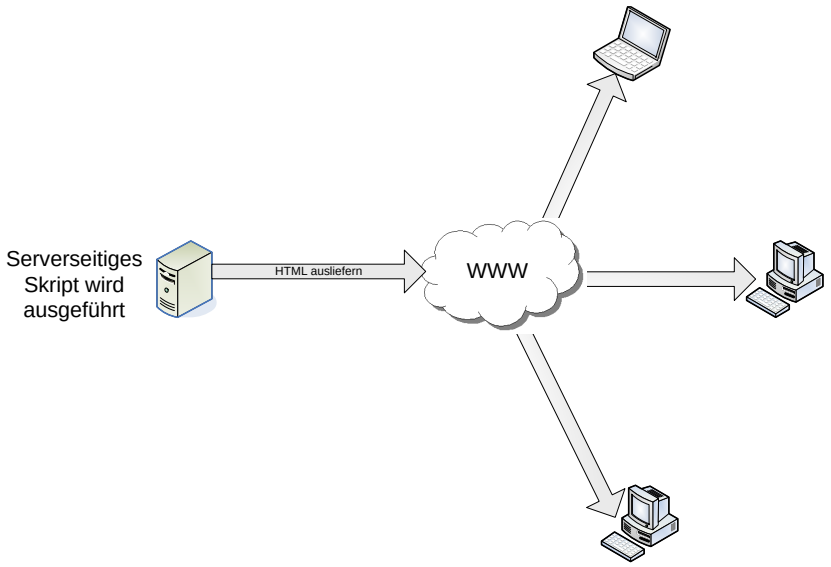

Abbildung 2.4: Arbeitsweise serverseitiger Skripte

ausgeliefert (siehe Abbildung 2.4). Häufig kann der Client nicht zwischen einer statischen und einer serverseitig, mit Hilfe von Skripten generierten Sprache, unterscheiden, da er in beiden Fällen eine HTML-Seite ausgeliefert bekommt.

2.4 Content Management System

Content Management Systeme (CMS), oder in der Literatur auch häufig Web Content Management Systeme (WCMS) genannt, sind Systeme zur Verwaltung und Publikation des digitalen Assets. Beim WCMS liegt der Fokus bezüglich der Publikation auf dem Web. Das ist auch der Aspekt, der für diese Arbeit interessant ist. Der Begriff Web Content Management System besteht aus vier Blöcken:

- Web
- Content
- Management
- System

Durch das Betrachten der einzelnen Teile wird auch klar, was ein WCMS ist.

Wie in Abbildung 2.5 zu sehen besteht das System aus den drei Blöcken, dem Web, dem Content und dem Management, die durch einen weiteren Block, das System, verbunden werden.

Der erste Block ist das Web (Netz), sie steht für das Intranet, Extranet und Internet. Im Internet angebotene Inhalte sind meist für die Öffentlichkeit gedacht, während ein Intranet meist eine streng definierte, häufig unternehmensweite Zielgruppe

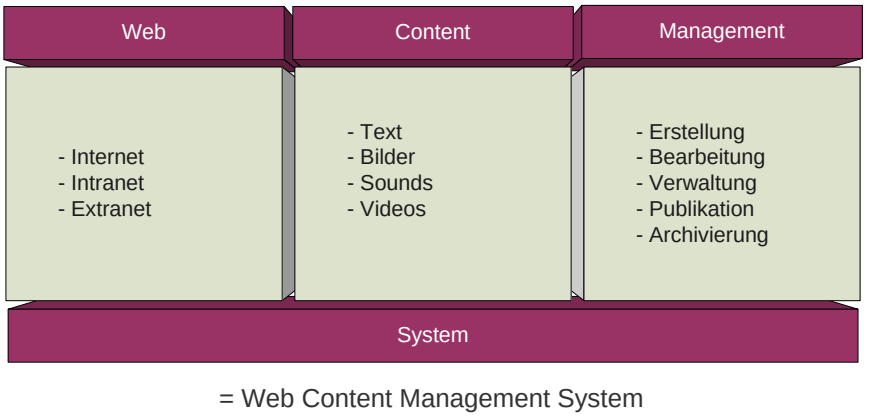

Abbildung 2.5: Aufbau eines Web Content Management Systems [Oliv01]

hat. Ein Extranet kann als ein, über das Internet zugreifbares, Intranet beschrieben werden. Dieser Zugriff ist in der Regel geschützt. Ein Extranet dient häufig zur Kommunikation der unterschiedlichen Standorte eines Unternehmens untereinander.

Der nächste Block ist der Content (Inhalt). Sie repräsentiert die zu publizierenden Inhalte, wie Texte, Bilder, Sounds, Videos und vieles mehr. Da ein WCMS zur Publikation und Darstellung der Inhalte Schablonen (Templates) benutzt, können die einmal strukturierten Inhalte im Zuge des Cross-Media-Publishings weiterverwendet werden. So ist eine Publikation außerhalb des Webs im Printbereich oder auf mobilen Endgeräten, wie Handys oder Palm Pilots mit geringerem Aufwand möglich.

Der dritte Block steht für das Management, hierunter fällt die gesamte Verwaltung des Inhalts, sowie die Abbildung von Workflows.

Diese drei Blöcke werden durch den darunter liegenden System-Block verbunden. Der System-Block ist für die Benutzerverwaltung und die damit verbundene Personalisierung, die Export-Import-Schnittstellen und die Programmierschnittstellen (API) zuständig. [Oliv01]

2.4.1 Evolution der Content Management System

Die erste Generation von CMSen propagierte den Einsatz von Datenbanken um Inhalte persistent zu machen. Aus diesen Datenbanken heraus wurden dynamisch HTML-Seiten generiert. Ebenfalls in der ersten Generation vertreten sind Systeme, die ein Dokumentenmanagement auf HTML-Dokumenten anbieten. Mit diesem System war es erstmals auch Nicht-Experten möglich Inhalte zu erstellen und zu pflegen. Workflows waren zu dieser Zeit noch nicht in die Systeme integriert.

Die zweite Generation praktizierte bereits eine konsequente Trennung von Darstellung und Inhalten. Inhaltsobjekte konnten mit Metadaten versehen werden. Publikationszeiträume konnten definiert und Navigationen automatisiert werden. Als Suche über die Inhalte war weiterhin die Volltextsuche dominierend.

Heutige Systeme entsprechen der dritten Generation. Die Schnittstellen der Systeme, wurden so erweitert, dass sie sich in vorhandene IT-Strukturen integrieren lassen. Bereits programmierte Komponenten lassen sich in das CMS integrieren. Inhalte können individuell kontrolliert werden und sind auch für einzelne Benutzer der Zielgruppe personalisierbar. Personalisierung ist ein wichtiges Attribut um die Kundenbindung zu fördern. [Oliv01] Immer mehr Hersteller bieten ein Content Repository zur Ablage der Daten an, auf das externe Applikationen über eine Schnittstelle zugreifen können.

Erst vor einigen Monaten wurde JSR170[3] verabschiedet, eine standardisierte Schnittstelle um auf den Content in einem CMS zuzugreifen. Dies könnte der Schritt zur nächsten Generation sein.

Eine Auflistung der Entwicklungsschritte von einer Generation der CMSe zur nächsten befindet sich in Tabelle 2.1.

2.4.2 Vorteile beim Einsatz eines Content Management System

Einer der Hauptgründe, die für den Einsatz eines CMS sprechen, ist die Kostenentwicklung eines Internetauftritts. Zwar sind beim Einsatz eines CMS die Kosten zu Anfang höher, da das CMS erst an die Bedürfnisse des Unternehmens angepasst werden muss. Es müssen Applikationen geschrieben und Templates im passenden Design erstellet werden. Wenn es allerdings um die Pflege des Systems geht, das Hinzufügen von neuen Seiten, oder die Aktualisierung bestehender Seiten, dann ist der Einsatz eines CMS günstiger. Vor allem dadurch, dass Templates und Code einfacher weiterverwendet werden können.

[3] http://developers.sun.com/learning/javaoneonline/2006/coreenterprise/TS-4474.html

Erste Generation	Zweite Generation	Dritte Generation
Management der Website durch Experten	Einbeziehung technisch nicht versierter Autoren	WCMS als Standard für alle Mitarbeiter
keine Meta-Informationen	Meta-Informationen zu Objekten	frei definierbare Meta-Informationen
manuelle Inhaltsstrukturierung	automatische Navigationen	kontextabhängige Navigationen
keine interne Zugriffsverwaltung	eigene Benutzerverwaltung	offene, standardisierte Benutzerverwaltung
manueller Workflow	fester Workflow mit Freigabezyklus	frei definierbare Workflows
keine dynamischen Inhalte	rudimentäre Dynamisierung	freie Dynamisierung des Contents
keine Personalisierung	einfache Personalisierung	individuelle Personalisier ung intern oder extern
keine Integration in bestehende Systeme	Applikationsintegration über APIs	Konrektoren zu wichtigen Webapplikationen
manuelle Aufbereitung des Contents	automatische Publikation des Contents	Content als Quelle für individuelle Webapplikationen

Tabelle 2.1: Die drei WCMS-Generationen [Oliv01]

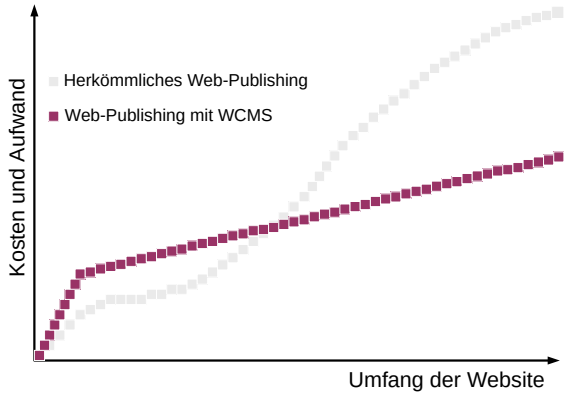

Abbildung 2.6: Kostenentwicklung einer Website [Oliv01]

Die Kostenentwicklung ist in Abbildung 2.6 dargestellt. Es ist zu sehen, dass sich gerade bei umfangreichen oder schnell wachsenden Internetauftritten der Einsatz eines CMSes lohnt.

2.5 Web - Structure - Mining

Beim Web-Mining wird versucht die Techniken des Data-Mining auf das Web zu übertragen, um so Informationen aus dem WWW extrahieren zur können. Diese Extraktion soll so weit als möglich automatisch oder zumindest teilautomatisch ablaufen. Beim Web Mining können drei Unterarten unterschieden werden (siehe: Abbildung 2.7):

1. Web-Content-Mining: Mining der Inhalte, zum Beispiel mit Techniken des Information Retrieval.

2. Web-Structure-Mining: Mining der (Verweis-)Struktur.

3. Web-Usage-Mining: Mining des Benutzerverhaltens, beispielsweise über die Auswertung von Logfiles.

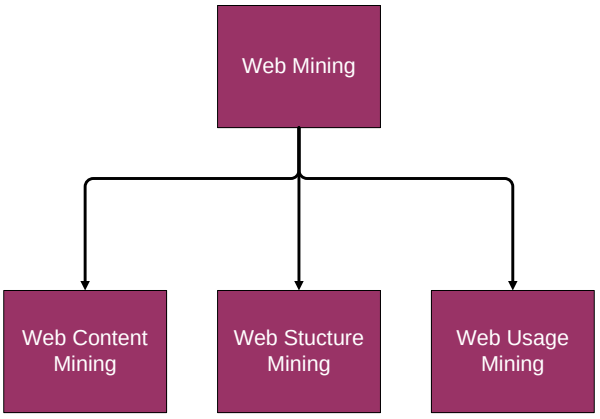

Abbildung 2.7: Web Mining

Die meisten Web-Mining-Projekte befassen sich damit, wie möglichst treffend Information im Internet gesucht werden kann. Damit lassen sich die meisten Arbeiten unter dem Begriff Web-Content-Mining zusammen fassen.

Die Contentextraktion, die als einer der Hauptaufgaben der Contentmigration angesehen werden kann, lässt sich dem Web-Structure-Mining zuordnen. Denn bei der Contentmigration spielt die Verlinkungshierarchie der Webseiten eine große Rolle, wobei nur die Seiten einer Domäne betrachtet werden. Weiter müssen Strukturen in den Webseiten gefunden und zugeordnet werden. Die bisherigen Arbeiten, die sich mit Web-Structrue-Mining befassen, versuchen hauptsächlich, „relevante Seiten“ von „weniger relevanten Seiten“ zu unterscheiden, um so eine Rangliste erstellen zu können.

2.6 Fuzzy-Logik

Die Grundidee, die hinter der Fuzzy-Logik (Unscharfe Logik) steht, geht bereits auf Platon zurück. Schon er hat die Meinung vertreten, dass die Logik auch die Graustufen zwischen wahr und falsch berücksichtigen sollte. Sein Schüler Aristoteles teilte die Meinung seines Lehrers jedoch nicht und so wurde die zweiwertige Logik bestimmend für die kommenden Jahrhunderte.

Der Begriff „Fuzzy-Logik“ wurde von L. Zadeh, einem Elektrotechnikprofessor an der Universität in Berkeley, Kalifornien 1965 eingeführt. Die Fuzzy-Logik stellt eine Erweiterung zu klassischen Logik dar und kennt im Gegensatz zur klassischen Logik

nicht nur die Werte „Wahr“ und „Falsch“, „0“ und „1“, sondern auch Werte dazwischen. Soll beispielsweise entschieden werden, ob ein Text „lang“ oder „kurz“ ist, so wird in der klassischen Logik ein Schwellwert angegeben. Alle Werte kleiner dem Schwellwert bekommen das Attribut „kurz“ zugewiesen, alle, die den Schwellwert überschreiten bekommen das Attribut „lang“ zugewiesen. Die Fuzzy-Logik erlaubt es in solchen Fällen mit stetigen Übergängen zu arbeiten. Abbildung 2.8 verdeutlicht diesen Unterschied.

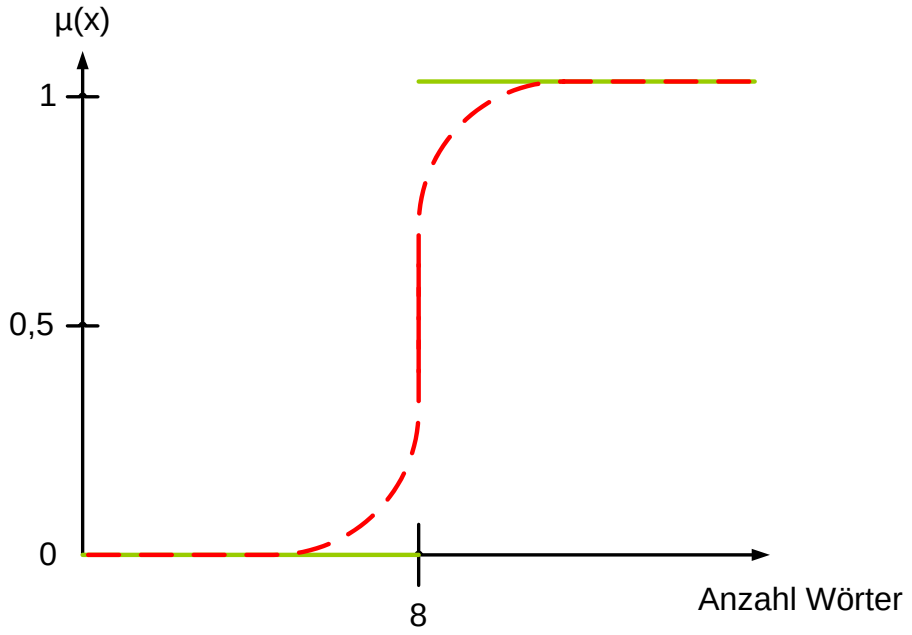

Abbildung 2.8: Vergleich einer Klassische Menge mit einer Fuzzy-Menge

Die klassische Menge kann also für jedes Element entscheiden, ob es zu Menge M gehört. Wobei die Menge M in Listenform gegeben sein kann

$$M = \{1, 2, 3, 5, 8, 13\}$$

wenn es sich um eine endliche Menge handelt. Auch unendliche Mengen können in dieser Schreibweise angegeben werden, wenn die Bindungsgesetze leicht zu erkennen sind

$$M = \{1, 2, 3, 5, 8, 13, 21, \ldots\}$$

wie beispielsweise die der Fibonacci-Reihe. Beliebige und vor allem unendliche Mengen werden allerdings meist mit Hilfe von Prädikaten (Eigenschaften) angegeben.

$$M = \{x \in G | P(x)\}$$

Eine weitere Möglichkeit eine Menge auf der Grundmenge G zu beschreiben besteht in der Angabe der charakteristischen Funktion. Diese Darstellungsform ist für die Fuzzy-Logik besonders relevant.

Definition 1 (Charakteristische Funktion [Lipp06]) *Sei G eine Grundmenge und M eine Teilmenge von G. Dann heißt die Funktion*

$$X_M : G \to \{0, 1\}$$

mit

$$X_M(x) = \begin{cases} 1 : x \in M \\ 0 : x \notin M \end{cases}$$

die Identifikator- oder charakterischtische Funktion der Menge M.

Der Unterschied zur Fuzzy-Menge ergibt sich aus folgender Definition:

Definition 2 (Fuzzy-Menge [Lipp06]) *Es sei G eine Grundmenge und $\mu_{\tilde{A}}$ eine Funktion der Grundmenge G in das Einheistintervall $[0,1]$ der reelen Achse, also $\mu_{\tilde{A}}\, G \to [0,1]$. Dann heißt die Menge $\tilde{A}$ aller Paare $(x, \mu_{\tilde{A}}(x))$*

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \,|x \in G\}$$

eine Fuzzy-Menge (unscharfe Menge, fuzzy set) über G. Die Funktion von $\mu_{\tilde{A}}$ wird als Mitgliedsgrad- bzw. Zugehörigkeitsfunktion (auch membership function) von $\tilde{A}$ bezeichnet. Für ein $x \in G$ wird der Wert $\mu_{\tilde{A}}(x)$ Zugehörigkeitsgrad, Erfüllungsgrad oder auch Mitgliedsgrad von x zu $\tilde{A}$ genannt.

Eine Fuzzy-Menge ordnet jedem Element der Grundmenge einen Zugehörigkeitsgrad zu.

Dieses Vorgehen ist der Praxis sehr sinnvoll, da häufig keine exakten Modelle erstellt werden können oder vorhandenen Daten unvollständig, verrauscht oder fehlerhaft sind.

Die Fuzzy-Logik kann somit mit Unsicherheit oder Unwissenheit umgehen. Wobei man unter Unsicherheit die Frage versteht, ob ein bestimmtes Ereignis überhaupt auftritt und unter Unschärfe die Frage, wie ein bestimmtes Ereignis charakterisiert wird. [Biew97]

Die Mitgliedsfunktion einer Fuzzy-Menge hat folgenden Verlauf: μ steigt mit zunehmendem x an und erreicht an der Stelle $x = m_1$ das Maximum. Dann bleibt μ bis zu einem Wert m_2 konstant und fällt wieder monoton ab. Drei Fuzzy-Mengen mit solchen Mitgliedsfunktionen treten besonders häufig auf; die Trapezmengen, Dreiecksmengen und die Gaußmengen.

Definition 3 (Trapezmenge [Lipp06]) *Die Trapezmenge $\tilde{T}(l, h_1, h_2, r)$ charakterisiert sich durch ihre linke Grenze l, ihre Hochpunkt 1 h_1, ihren Hochpunkt 2 h_2 und ihre rechte Grenze r. Die Mitgliedsfunktion $\mu_{\tilde{T}}$ der Trapezmenge hat folgende Gestalt:*

$$\mu_{\tilde{T}} = \left\{ \begin{array}{rcl} 0 & : & x \leq l \\ \frac{x-r}{m_1-l} & : & l < x < h_1 \\ 1 & : & h_1 \leq x \leq h_2 \\ \frac{h_2-x}{r-h_2} + 1 & : & h_2 \leq x \leq r \\ 0 & : & r \leq x \end{array} \right\}$$

Definition 4 (Dreiecksmenge [Lipp06]) *Die Dreiecksmenge $\tilde{D}(l, m, r)$ charakterisiert sich durch ihre linke Grenze l, ihr Maximum m und ihre rechte Grenze r. Die Mitgliedsfunktion $\mu_{\tilde{T}}$ der Dreiecksmenge hat folgende Gestalt:*

$$\mu_{\tilde{D}} = \left\{ \begin{array}{rcl} 0 & : & x \leq l \\ \frac{x-r}{m-l} & : & l < x < m \\ 1 & : & x = m \\ \frac{m-x}{r-m} + 1 & : & m \leq x \leq r \\ 0 & : & r \leq x \end{array} \right\}$$

Definition 5 (Gaußmengen [Lipp06]) *Die Gaußmenge $\tilde{G}(m, w)$ wird mit Hilfe einer Exponentialfunktion, der Mitte m und dem Weitenparameter w beschrieben. Die Mitgliedsfunktion $\mu_{\tilde{T}}$ der Gaußmenge hat folgende Gestalt:*

$$\mu_{\tilde{G}} = \exp\left(-\left(\frac{x-m}{w}\right)^2\right)$$

2.6.1 Fuzzy-Filter

Der Fuzzy-Filter ist eine spezielle Anwendung eines regelbasierten Fuzzy-Systems ohne Rückkopplung. Als Basis dient eine Menge von `IF-Then`-Regeln. 1972 wurde das Konzept eines regelbasierten Fuzzy-Systems von L. Zadeh eingeführt. Die ersten konkreten Fuzzy-Controller wurden von F. Mamdani un S. Assilian 1975 entwickelt. [Lipp06] Ein Fuzzy-Controller nach [Mada] besteht aus den Kernkomponenten:

1. Fuzzyfizierer
2. Entscheidungslogik mit Regelbasis
3. Defuzzifizierer

Das Zusammenspiel der Kernkomponenten im Fuzzy-Controller ist in Abbildung 2.9 zu sehen. Der Fuzzyfizierer wandelt die crispen (scharfen) Eingabedaten in Fuzzy-Mengen um. Die Regelbasis hält eine Menge von Regeln fest. Sie besteht aus einer endlichen Menge von Regeln, der Form:

$$R : IF(x_1 IS\ \tilde{A}_1) AND \ldots AND (x_n IS\ \tilde{A}_n) THEN (y_k IS\ \tilde{B}_k)$$

Die Entscheidungslogik (Inferenz Maschine) wendet die linguistischen Regeln auf die fuzzifizierten Eingaben an und gibt Fuzzy-Werte aus. Der Defuzzifizierer wandelt die Fuzzy-Werte der Entscheidungslogik wieder in crispe Werte um.

Eine Variante dieses Fuzzy-Controllers ist der Suego-Controller. Er unterscheidet sich durch die fehlende Defuzzifizierung vom Controller nach Madami. Dies wird erreicht, indem die Regelbasis direkt crispe Werte liefert. Eine Motivation hierfür liefert der unter Umständen hohe Aufwand der Defuzzifizierung.

2.6.1.1 Fuzzifizierer

Der Fuzzifizerer muss die ankommenden crispen Daten (Eingabevektor) in Fuzzy-Werte umwandeln, da die Enscheidungslogik Fuzzy-Zahlen als Eingabe benötigt. Die Schwierigkeit besteht darin eine gute Abschätzung für die Unschärfen zu finden. [Lipp06]

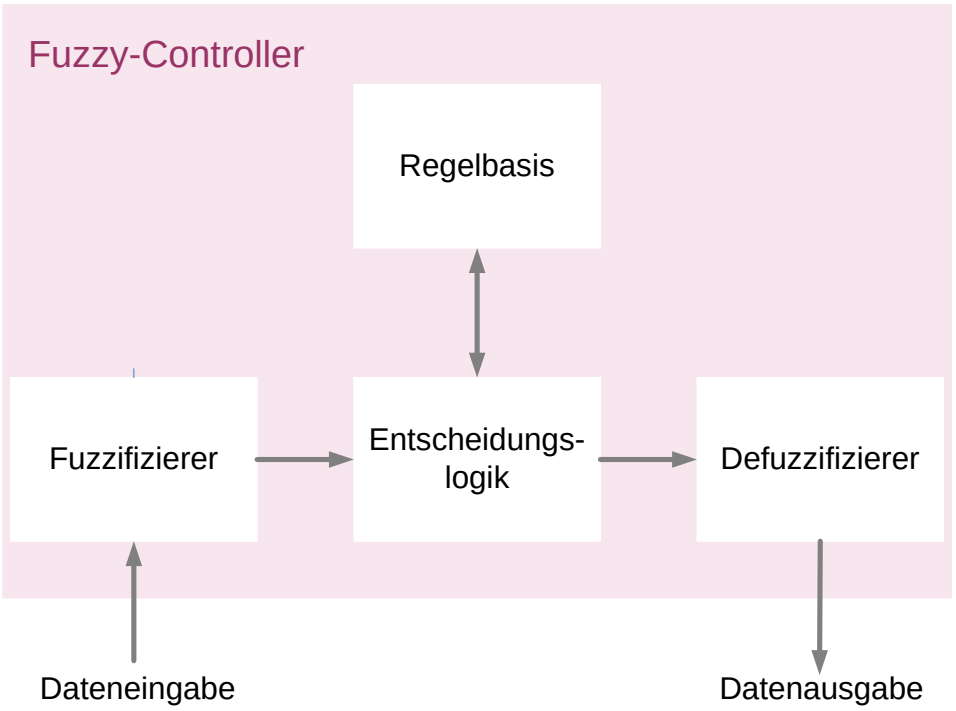

Abbildung 2.9: Fuzzy-Controller

2.6.2 Linguistische Variablen

Definition 6 (Linguistische Variable [Lipp06]) *Das Tupel* $< X, A_x, G, M_x >$ *wird als linguistische Variable bezeichnet, wobei X der Name der linguistischen Variablen ist, G die Grundmenge,* A_x *die Menge der linguistischen Werte (Ausprägungen, linguistische Terme), die die linguistische Variable annehmen kann und* M_x *die Funktion, die einen linguistischen Wert aus* A_x *einer Fuzzy-Menge über der Grundmenge G zuordnet.*

Soll beispielsweise für die Grundmenge $G = [0, \infty]$ die linguistische Variable X für die Textlänge (in Wörtern) definiert werden. Mögliche Ausprägungen der linguistischen Variablen sind „kurz“, „mittel“, „lang“ und „sehr lang“. So ist $A_x = \{kurz, mittel, lang, sehrlang\}$. Die Funktion M_x ordnet den Ausprägungen der linguistischen Variable die entsprechende Bedeutung durch eine restringierende Fuzzy-Menge über G zu, $M_x : A_x \rightarrow \tilde{A}_x$, wie

$$mittel \rightarrow MITTEL = \tilde{T}(7, 20, 80, 120)$$

eine Trapezmenge, für eine mittlere Textlänge.

2.6.3 Fuzzy-Operatoren

Fuzzy-Mengen werden berechnet, indem man sie mit Fuzzy-Operatoren verknüpft. Das Ergebnis sind wieder Fuzzy-Mengen. Es gibt zwei wesentliche Klassen von Operatoren auf Fuzzy-Mengen: Die t-Normen und die s-Normen (auch als t-Konorm bezeichnet). Die t-Norm berechnet den Durchschnitt zweier Fuzzy-Mengen. Die Vereinigungsmenge zweier Fuzzy-Mengen wird durch eine s-Norm gebildet. Eine Funktion $F : [0,1] \times [0,1] \rightarrow [0,1]$ muss folgende Voraussetzungen erfüllen, damit sie eine t-Norm bzw. eine s-Norm ist.

$$
\begin{array}{ll}
F(a,b) = F(b,a) & (Kommutativität) \\
F(a,b,c) = F(a,F(b,c)) = F(F(a,b),c) & (Assoziativität) \\
a \leq c \wedge b \leq d \Rightarrow F(a,b) \leq F(c,d) & (Monotonie) \\
F(0,a) = 0, F(a,1) = a & (Normierung\ der\ t-Norm) \\
F(0,a) = a, F(a,1) = 1 & (Normierung\ der\ s-Norm)
\end{array}
$$

Häufig in der Praxis eingesetzten Operatoren sind der Minimum-Operator als t-Norm und der Maximum-Operator als s-Norm.
Der Modus Ponens ist eine Operation aus der klassischen Logik, wird aber auch in der Regelbasis des Fuzzy-Controllers gebraucht.

Definition 7 (Modus Ponens) *Der Modus Ponens besagt, dass wenn es eine Regel gibt, die besagt, dass wenn a gilt auch b gilt und a in einem konkreten Fall gilt, dann gilt auch b.*

$$
\begin{array}{ll}
a & \rightarrow \quad b \\
a & \\
\hline
 & \quad b
\end{array}
$$

2.6.4 Entscheidungslogik mit Regelbasis

Die Regelbasis bildet den Kern des Controllers. Hier werden endlich viele `IF-Then`-Regeln gespeichert, die das Wissen eines Operators repräsentieren. So muss für jede Dimension des Eingaberaums $X_1 \times \ldots \times X_n$ und jede Dimension des Ausgaberaums $Y_1 \times \ldots \times Y_n$ eine linguistische Variable X_i definiert werden. Weiter müssen die Ausprägungen der linguistischen Variablen und die zugrunde liegenden Fuzzy-Mengen bereit gestellt werden. In der Prämisse sind ausschließlich Konjunktionen als Verknüpfungen erlaubt. Die Konklusion jeder Regel enthält genau eine Fuzzy-Ausgabemenge. Nach dem Prinzip des verallgemeinerten Modus Ponens liefert die Entscheidungslogik aus den Regeln der Regelbasis $\tilde{A}_r$ die Ausgabe-Fuzzy-Menge $\tilde{B}_j$ für jede Dimension des Ausgaberaumes.
Um den Erfüllungsgrad der Prämisse zu berechnen wird jede Regel einzeln angewendet, was auch gleichzeitig ein Maß für die Qualität der Regelbasis bezüglich der Eingabe ist. Die Berechnung erfolgt in zwei Schritten

1. Zuerst werden die Zugehörigkeiten der Eingabe-Werte zu jeder Fuzzy-Menge der Regel-Prämisse bestimmt.

2. Anschließend werden die Werte mit einer t-Norm verknüpft.

Hat eine Regel die Form

$$IF\left(x_1 = \tilde{A}_1\right) AND \cdots AND\left(x_n = \tilde{A}_n\right) THEN\left(y_k = \tilde{B}_k\right)$$

werden zuerst die Teilkomponenten $\left(x_1 = \tilde{A}_1\right)$ berechnet und daraufhin die Ergebnisse mittels t-Norm verknüpft. Das enspricht der AND-Verknüpfung.

2.6.5 Defuzzifizierer

Da als Ausgabewerte crispe Werte benötigt werden, müssen die Fuzzy-Werte in crispe Werte umgewandelt werden. Das ist die Aufgabe des Defuzzifizierers. Es muss also aus Ausgaben der Regelbasis $\tilde{D}_j$ eine reelle Zahl y_j berechnet werden. Die reelen y_j bilden den Ausgabevektor aus Y, wobei $1 \leq j \leq m$ gilt.

2.7 Zusammenfassung

In diesem Kapitel wurde auf die für das Verständis der Arbeit relevatnen Grundlagen eingeganen. Es wurden Begriffe des WWW erläutert und auf die relevanten Dokumentenbeschreibungssprachen eingegangen. Die Grundlangen der Fuzzy-Logik und der Aufbau eines Fuzzy-Controlers wurden erläutert.

3. Analyse

In diesem Kapitel werden die anfallenden Arbeitsschritte analysiert und geprüft, welche sich automatisieren lassen.

Um zu umfangreiche Anpassungen der importierten Seiten zu vermeiden, soll die Migration, auf der sich bereits online befindenden Präsenz basierend, durchgeführt werden. D.h. der zu migrierende Inhalt soll aus den bereits online geschalteten HTML-Seiten extrahiert werden. So ist das System unabhängig vom zuvor benutzen CMS.

3.1 Begriffserklärung

In diesem Abschnitt sollen verschiedene im Folgenden immer wieder auftauchende und für das Verständnis wichtige Begriffe geklärt werden.

- **Contentpage/ Inhaltsseite**: Eine Inhaltsseite ist eine Seite, die zu migrierenden Inhalt enthält.
- **HTML-Head/ HTML-Kopf**: Der Head ist der Teil einer Webseite, der durch den head-Tag (`<head> ... </head>`) begrenzt wird.
- **HTML-Body/ HTMl-Körper**: Der Body ist der Teil einer Webseite, der durch den body-Tag (`<body> ... </body>`) begrenzt wird.
- **Content/ Inhalt**: Der Content/ Inhalt einer Webseite ist der Teil der Seite, der in das neue System übernommen werden soll.
- **Contentelement/ Inhaltselement**: Im Zielsystem sind Inhaltselemente als auf den Contentseiten erstellbare Einheiten zu finden.
  Im Inhalt sind Contentelemente Strukturen, die in Aufbau und Komplexität den Inhaltselementen des Zielsystems entsprechen.
- **Contentatom/ Inhaltsatom**: Ein Content-/ Inhaltsatom ist ein elementarer Teil des Inhalts. Es handelt sich um einen Textknoten im Contentbaum, der allerdings noch weitere Tags enthalten kann.

- **Vaterseite/ Parentpage**: Die Definition einer Vaterseite setzt sich aus zwei Komponenten mit unterschiedlicher Priorität zusammen.
  1. Die Vaterseite einer Webseite ist die im Pfad angegebe, übergeordnete Seite.
  2. Ist keine übergeordneter Webseite im Pfad vorhanden, so ist die Vaterseite, die Seite, die im bisher aufgebauten Seitenbaum bei einer Breitensuche den ersten Verweis auf die Seiten enthält.
- **Schwesterseiten**: Seiten, die dieselbe Vaterseite haben.
- **Schwesterelemente**: Contentelemente auf der gleichen Hierarchiestufe.

Um die Relation Vaterseite zu verdeutlichen, hier ein Beispiel. Auf der Hompage der Namics AG ist die Seite „**http://www.namics.com/wissen.html**“ die Vaterseite von „**http://www.namics.com/wissen/whitepapers.html**“, da der Dateiname der Vaterseite „**wissen**“ im Pfad direkt vor der Kindseite enthalten ist. Es greift also die erste Bedingung.

Unter einer anderen Domäne ist die Webseite „**http://www.nicoleschmidt.info/index.php?id=2**“ die Vaterseite von „**http://www.nicoleschmidt.info/index.php?id=8**“, da es die erste Seite ist, die darauf verweist. Im Pfad ist keine Hierarchie enthalten, da die Seiten über ein HTTP-Get-Request mit Parametern (id=8) aufgerufen werden. Die erste Bedingung kommt hier nicht zum tragen, so greift die zweite Bedingung.

3.2 Contentmigration

Die Contentmigration lässt sich in zwei große Teilprobleme einteilen, die getrennt voneinander behandelt werden können: Die Migration der Seitenstruktur und die Migration des Seiteninhalts. Das heißt, zum einen müssen die Seiten im Ziel-CMS angelegt werden und mit den benötigten Attributen versehen werden. Zum anderen müssen die angelegten Seiten mit Inhalten gefüllt werden, wie Bilder, Text, Listen, und mehr. In den folgenden Unterkapiteln wird genauer auf die Teilprobleme eingegangen.

3.2.1 Migration der Seitenstruktur

Um die Seitenstruktur zu migrieren, muss sie nachgebaut werden. Daher müssen beim Anlegen einer Seite bestimmte Informationen zu den Seiten in das CMS eingetragen werden. Zu jeder Seite muss die Vaterseite (Parent) bekannt sein, um die Baumstruktur im Zielsystem korrekt nachbilden zu können. In vielen CMSen können der Titel der Seite, die Metadaten, wie Schlüsselwörter und Beschreibung beim Anlegen der Seite editiert werden. Dies sind Daten, die sich beim spidern der Internetpräsenz ermitteln lassen.

Des Weiteren ist es je nach CMS wichtig, das Seitentemplate der anzulegenden Seite zu ermitteln, da dies in manchen Systemen beim Anlegen der Seite angegeben werden muss und im Nachhinein nicht mehr änderbar ist.

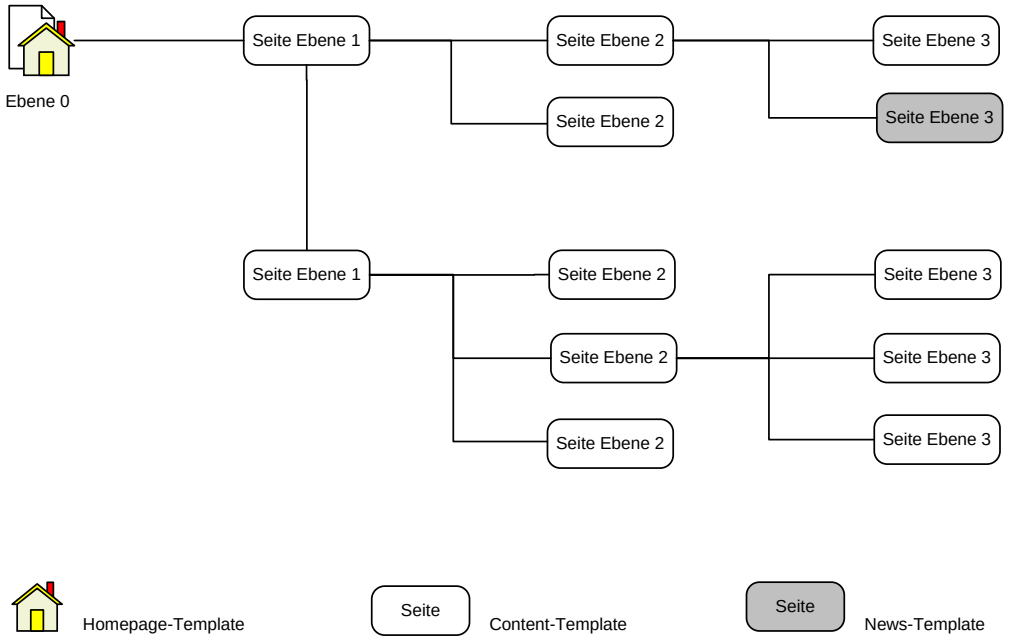

Abbildung 3.1: Seitenstruktur

In Abbildung 3.1 ist exemplarisch eine solche Seitenstruktur dargestellt. Auf Ebene 0 ist die Homepage der Internetpräsenz zu erkennen. Die Homepage dient als Einstiegspunkt. Von dort aus wird auf Unterseiten der jeweils nächsten Ebene verwiesen. Diese Unterseiten enthalten ebenfalls Verweise zu Unterseiten und so weiter. Zu sehen ist auch, dass die einzelnen Seiten unabhängig von ihrer Position Ausprägungen verschiedener Templates sein können.

3.2.2 Migration des Seiteninhalts

Um den Inhalt der einzelnen Webseiten zu migrieren ist es notwendig diesen auf den ursprünglichen Webseiten zu identifizieren. Es muss eine Möglichkeit gefunden werden zwischen Seiteninhalt, Seitenheader, Navigation, Fußzeile und anderen Elementen zu unterscheiden, die nicht direkt zum Seiteninhalt gehören.

Um die identifizierten Inhalte den im Zielsystem vorhandenen Inhaltselementen zuzuordnen, muss der Aufbau des Zielsystems bekannt sein. Es muss ein Modul entstehen, welches die Zuordnung der gefundenen Inhalte zu den Strukturen des Zielsystems übernimmt.

Von Bedeutung ist ebenfalls die Spezifizierung einer Schnittstelle, die die Konfiguration dieses Moduls erlaubt und über die die Strukturen der Inhaltselemente des Zielsystems in das Modul importiert werden können.

Die benötigten Ein- und Ausgaben sind in Abbildung 3.2 dargestellt.

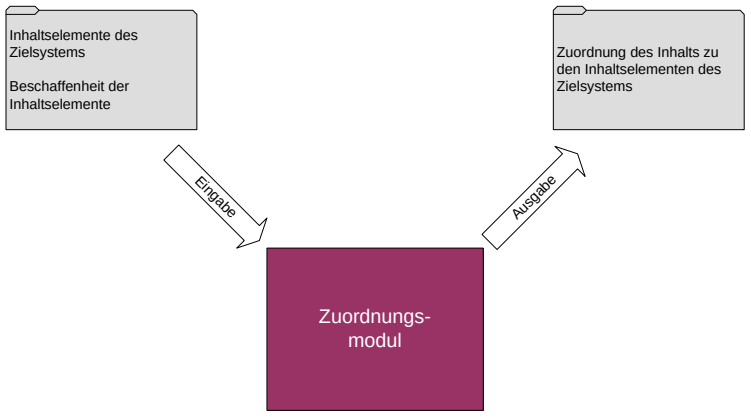

Abbildung 3.2: Modul für die Inhaltszuordnung

3.3 Anforderungen an das Gesamtsystem

Es ergeben sich eine Reihe von Anforderungen beim Design eines solchen System. Das Verfahren soll unabhängig vom Ausgangsystem sein. Es soll keine Rolle spielen, ob für den alten Internetauftritt ein CMS eingesetzt wurde oder, wenn dem so war, welches.

Sowohl die Seitenstruktur, als auch die Seiteninhalte sollen migriert werden können.

Bei der Migration soll die Seitenstruktur erkannt und nachgebildet werden, unabhängig von der Art der Seitenadressierung. Als Seitenadressierungstechniken werden zwei Arten unterschieden. Erstens eine direkte Adressierung der HTML-Seite über den kompletten Pfad zur HTML Datei und zweitens über ein HTTP-Get Request. HTTP-Post Request können nicht berücksichtigt werden, da diese im Header der HTTP-Protokolls stecken und in der URL nicht ersichtlich sind. Für die Migration der Seiteninhalte sind die Inhaltstypen des Zielsystems bekannt. Auf diese soll sich die Zuordnung je nach Zielsystem einstellen können.

Nicht migriert werden sämtliche Arten von Skripten, client- sowie serverseitig, da Skripte nicht zum Inhalt gehören, sondern Funktionalität darstellen. Sie zu portieren wäre Teil der Konfiguration des CMS, nicht der Contentmigration. Ebenso nicht migriert werden Applets und alle anderen Elemente die nicht dem Inhalt, sondern der Funktionalität zugeordnet werden.

3.4 Verwandte Arbeiten

Das bisherige Vorgehen bei der Contentmigration besteht darin, den Content des alten Internetauftritts von Hand, bestenfalls via Copy-and-Paste, in das neue CMS einzupflegen. Es gibt Unternehmen, zu deren Leistungskatalog auch die Contentmigration gehört. Welches Verfahren diese für die Migration einsetzen ist nicht bekannt.

Wird nur auf eine neuere Version eines bereits benutzen CMS migriert, so bieten die meisten Systeme eine Möglichkeit den Inhalt zu überführen.

3.4.1 Spider Architektur

Definition 1 (Spiderprogramm [Heat02]) *Ein Spiderprogramm bewegt sich entlang der Verweise (Links) durch das Internet.*

Während sich das Spiderprogramm durch das Netz bewegt kann es die aufgerufenen Seiten nach Informationen durchsuchen, bestimmte Informationen extrahieren oder die komplette HTML-Seite lokal abspeichern. Die gängigen Suchmaschinen arbeiten mit Spiderprogrammen.

In [Heat02] werden zwei Architekturen für die Implementierung eines Spiderprogramms vorgestellt. Die eine verfolgt einen rekursiven Ansatz. Die andere arbeitet mit verschiedenen Warteschlangen und lässt eine multithreadingfähige Implementierung zu.

Die rekursive Lösung ist in Prozedur 1 dargestellt.

Data : String URL
Result : void
URL herunterladen;
URL parsen;
für *jede gefundene URL* **tue**
  rufe rekursiveSpider (mit gefundener URL) auf;
heruntergeladene Seite verarbeiten;

Prozedur `Rekursives Spiderprogramm` [Heat02]

Im zweiten Ansatz existieren je eine Warteschlange für die noch nicht abgearbeiteten URLs, die aktuell bearbeiteten URLs, die kompletten URLs und die URLs bei deren Bearbeitung ein Fehler aufgetreten ist.

Abbildung 3.3 zeigt den Weg einer URL durch die verschiedenen Warteschlangen. Eine neu gefundene URL wird in die Warteschlange geschrieben. Sobald sie an der Reihe ist wird sie verarbeitet. War die Verarbeitung erfolgreich, wird sie zu den kompletten URLs hinzugefügt, andernfalls zu den fehlerhaften URLs. Befindet sich eine URL in der Liste der kompletten URL oder der fehlerhaften URLs ist ihre Verarbeitung abgeschlossen.

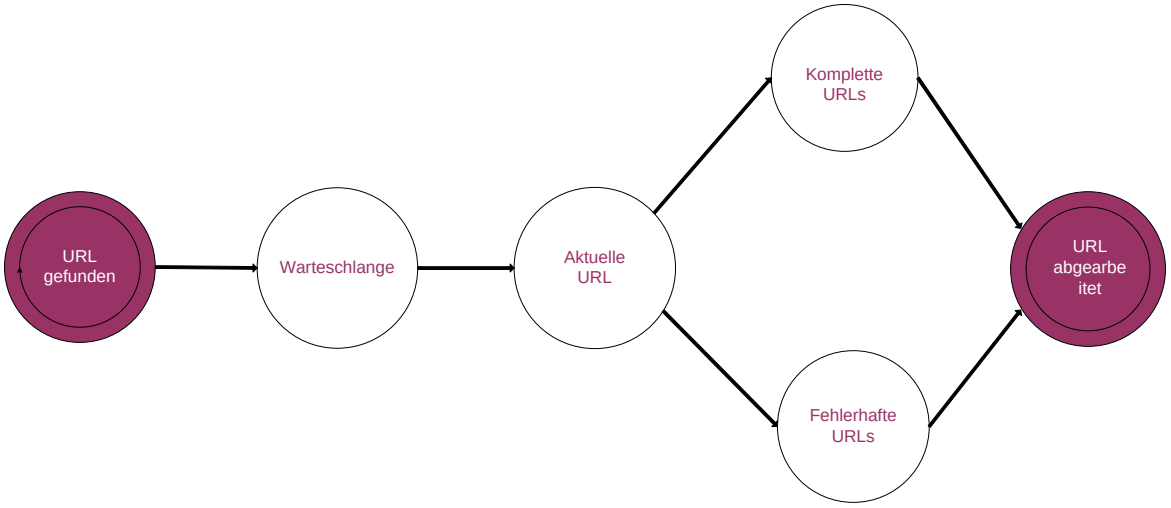

Abbildung 3.3: URL Status nach [Heat02]

Außerdem schlägt [Heat02] vor, die HTML Seiten nach dem String „href“ zu durchsuchen um die Verweise in den Dokumenten zu finden. Damit werden nicht nur die gewöhnlichen Verweise im `<a>`-Tag abgedeckt, sondern auch andere Verweise, wie in `<link>`-Tags, `<map>`-Tags und `<area>`-Tags.

3.4.2 HTML parsen

Ein Hauptproblem beim Parsen von HTML-Seiten ist, dass es sich häufig um nicht valides HTML handelt. Da die gängigen Browser sehr fehlertolerant sind stellen sie auch nicht der Spezifikation entsprechendes HTML fehlerfrei dar. Daher sind viele der sich im Netz befindlichen HTML-Seiten nicht valide. Dies kann ein Problem für einen Parser darstellen, da dieser, genauso wie der Browser, sehr fehlertolerant arbeiten muss.
Es existieren mehrere freie Bibliotheken zum Parsen von HTML, mit dem Ziel auf HTML-Dokumente, wie auf XML-Dokumente zugreifen zu können[1].
Dazu werden die HTML-Dokumente, wie XML-Dokumente in einen DOM (**D**ocument **O**bject **M**odel) geladen. Der DOM ist eine vom W3C [Phil05] veröffentlichte, standardisierte Schnittstelle zur programmatischen Be- und Verarbeitung von XML.

[1] http://java-source.net/open-source/html-parsers

3.4.3 HTML - Tag Klassifikation

In [XTMa05] wird eine Einteilung für HTML-Tags vorgeschlagen. Die HTML-Tags werden in drei Gruppen eingeteilt:

Definition 1 (Harte Tags) *Tags, die die Linearität (Fluss) des Textes unterbrechen, wie* `<title>`, `<li>` *und* `<p>`.

Definition 2 (Weiche Tags) *Tags, die signifikante Teile des Dokuments kennzeichnen, die allerdings beim Lesen des Textes transparent werden, wie* `<b>`, `<i>` *oder* `<sc>`.

Definition 3 (Sprung Tags) *Tags, die alle möglichen Arten von Referenzen repräsentieren, wie* `<a>` *oder* `<area>`.

Wichtig für die Identifikation der Contentatome als auch der Contentelemente sind die harten Tags. Da sie eine Abgrenzung zwischen den Elementen darstellen können.

Eine weitere Einteilung der HTML-Tags wird in [Nied02] vorgenommen. Die Tags werden in folgende Gruppen eingeteilt:

- Struktur-Tags
- Text-Tags (Blockelemente)
- Text-Tags (Inline-Formatierung)
- Text-Tags (Logische Formatierung)
- Text-Tags (Darstellungsorientierte-Formatierung)
- Listen-Tags
- Platzhalter- und Positionierungs-Tags
- Link-Tags
- Tabellen-Tags
- Rahmen-Tags
- Formular-Tags
- Multimedia-Tags
- Skript-Tags

In Tabelle 3.1 sind die wichtigen Tag-Gruppen zusammen mit ihren Mitgliedern aufgelistet.

Tag-Gruppe	Zugehörige Tags
Text-Tags: Blockelemente	<dd>, , <dl>, <dt>, <h1> bis <h6>, , , <p>,
Listen-Tags	<dl>, <dd>, <dt>, , bis
Link-Tags	<a>, <map>, <area>, <link>
Skript-Tags	<script>, <noscript>
Mulitmedia-Tags	<applet>, <bgsound>, <embeded>, <object>, <param>
Tabellen-Tags	<caption>, <table>, <tr>, <td>, <th>

Tabelle 3.1: Tag-Gruppen nach [Nied02]

3.4.4 Inhaltsidentifikation

Wie in [FuLi04] beschrieben befindet sich der Inhalt von Webseiten einer Domäne häufig an der gleichen Position. Dies ist unabhängig davon, ob die Seiten manuell oder maschinell erstellt wurden. In Abbildung 3.4 ist der Aufbau einer Standard-HTML-Seite zu sehen. Die meisten im Netz befindlichen Webseiten entsprechen diesem Aufbau.

Verschiedene Inhaltsthemen im Contentbereich werden durch Trennelemente voneinander abgegrenzt.

[FuLi04] unterscheidet zwischen expliziten und impliziten Bindungen von Inhaltselementen:

Definition 1 (Explizite Bindung) *Explizite Bindungen werden durch bestimmte Tags identifiziert. Tags, die eine explizite Bindung anzeigen sind:* <hr>, <table>, <tr>, <td>, , <p>, <bloquote>, <pre>, <form>, <ol> *und* .

Definition 2 (Implizite Bindung) *Implizite Bindungen werden durch Begrenzungszeichen, wie* </br> *angezeigt.*

Die Zusammengehörigkeit von Inhaltsatomen kann über deren Position im Inhaltsbaum identifiziert werden. Es ist beispielsweise wahrscheinlich, dass Schwesternelemente im Inhaltsbaum zusammen gehören. Die Beziehung der Schwesterelemente ist in Abbildung 3.5 dargestellt.

Doch diese Schwesterbeziehung zwischen Inhaltselementen kann lediglich als Anhaltspunkt dienen und liefert keinerlei Sicherheit über die Zusammengehörigkeit der Elemente.

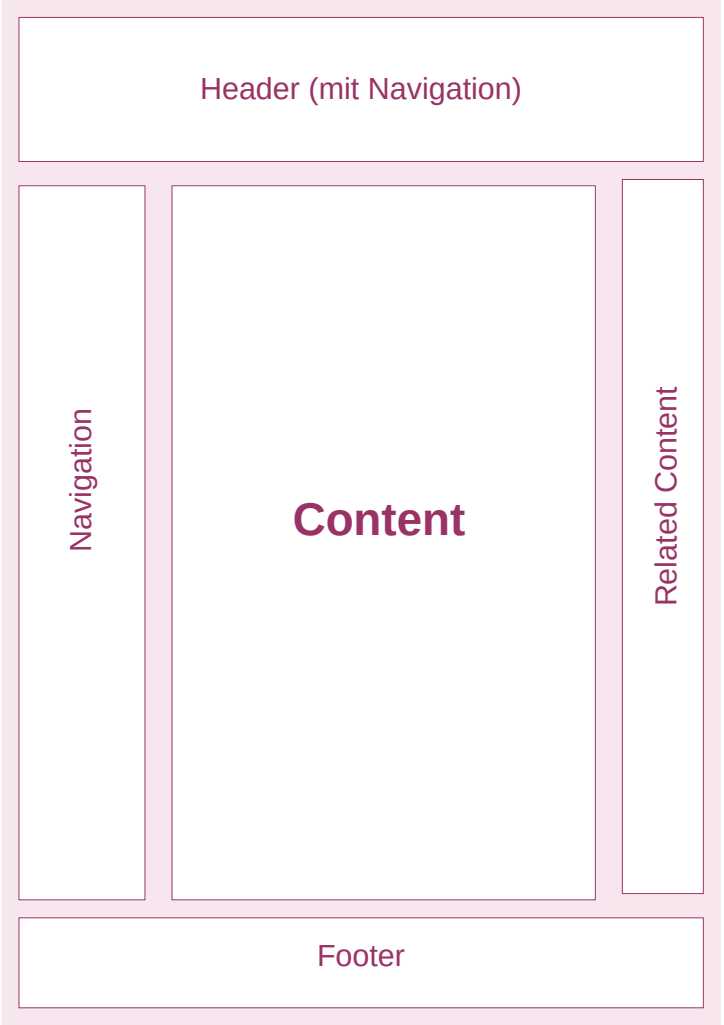

Abbildung 3.4: Aufbau einer Webseite

3.5 Verarbeitung von XML

Es gibt zwei standardisierte Vorgehensweisen XML in einer Anwendung zu verarbeiten.

Die eine Methode durchläuft eine XML-Datei sequenziell und sucht während dieses Durchlaufes nach einem vorgegebenen Muster. Ist dieses Muster gefunden, wird ein Ereignis ausgelöst und der Entwickler kann eine gewünschte Aktion durchführen.

Die zweite Methode lädt die komplette XML-Datei in Form einer Baumstruktur (DOM [Phil05]) in den Arbeitsspeicher. Dies ist zwar speicheraufwendiger, als die erste Methode, wenn aber viele Änderungen an einer XML-Datei durchgeführt werden müssen, ist es die effizientere Methode. Es kann auf jedes Element, mit Hilfe des Pfades oder der Vater-Kind-Beziehungen im XML-Baum, zugegriffen werden.

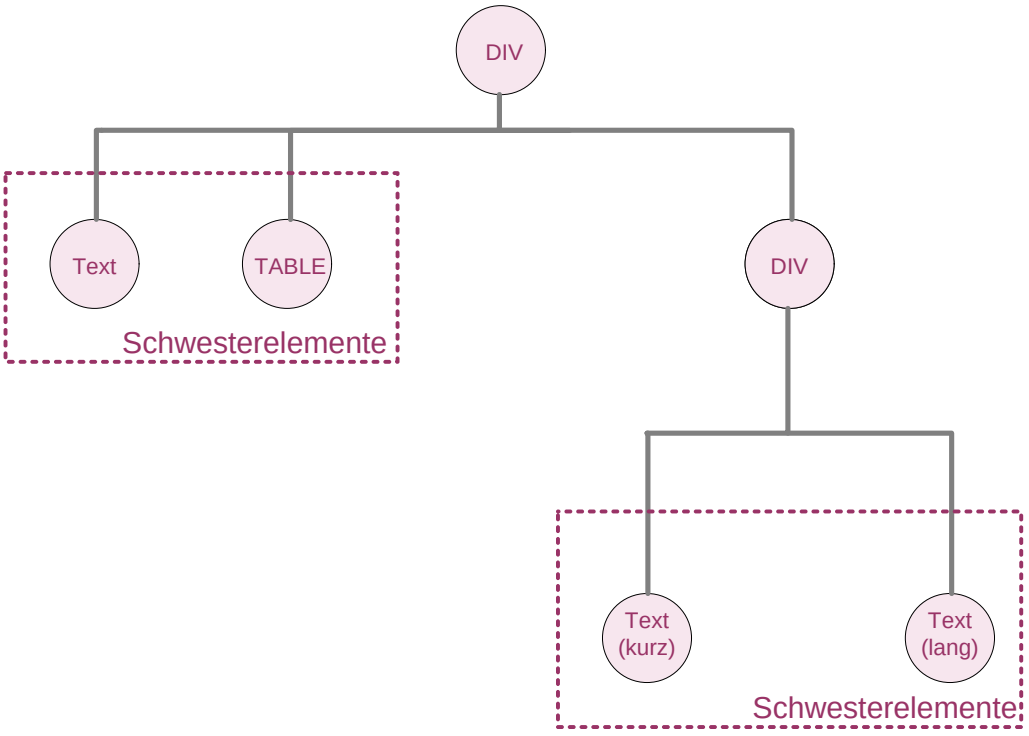

Abbildung 3.5: Inhaltsbaum mit Schwesterelementen

3.6 Zusammenfassung

In diesem Abschnitt wurde das Problem der Contentmigration analysiert und in seine Teilprobleme (Seitenmigration und Seiteninhaltsmigration) aufgespaltet. Die Seitenmigration und die Seiteninhaltsmigration wurden näher beleuchtet. Außerdem wurde auf bereits vorhandene Lösungsansätze für Erkennung von Strukturen in Inhalten vorgestellt und kurz auf die Verarbeitung von XML eingegangen.

4. Automatisierung der Contentmigration

Im Folgenden wird der, auf Grund der Problemanalyse im vorherigen Kapitel erarbeitete, Lösungsansatz vorgestellt. Es wird auf die Architektur des Gesamtsystems, sowie die Strategien der Inhaltszuordnung eingegangen.

4.1 Konzept des Gesamtsystems

Es handelt sich bei dem vorgestellten Problem nicht um ein klassisches Problem für Knowledge Discovery in Databases (KDD) oder Data Mining. Es wird nicht nach Mustern und Strukturen gesucht, sondern nach Daten, die in bestimmte Muster passen. Trotzdem beruht der Lösungsansatz auf dem Basisprozess des KDD, wie er in [Usam96] vorgestellt wurde.

Wie in Abbildung 4.1 veranschaulicht, werden die HTML-Seiten zuerst in XML konformes XHTML transformiert. Aus den einzelnen XHTML-Dokumenten werden darauf hin die nicht benötigten Tags entfernt. Auf Basis der Strukturen des Zielsystems kann die Zuordnung der Daten stattfinden. Sind die Daten zugeordnet, kann der Import ins Zielsystem durchgeführt werden.

Das zu Lösung vorgeschlagene Gesamtsystem ist modular. Es besteht aus drei Modulen, dem Spider-Modul, dem Mapping-Modul und dem Import-Modul. Als Schnittstellen zwischen den einzelnen Modulen werden verschiedene XML-Dokumente verwendet, das *pure_content.xml*, das *mapped_content.xml*, das *target_structure.xml* und das *page_structure.xml*.

In Abbildung 4.2 ist das Zusammenspiel der einzelnen Module und der Schnittstellendokumente dargestellt. Das Spider-Modul erstellt das *pure_content.xml* sowie das *page_structure.xml*. Das Mapping-Modul erstellt das *mapped_content.xml* auf der Basis des *pure_content.xml* und des *target_structrue.xml*. Das Import Modul verarbeitet

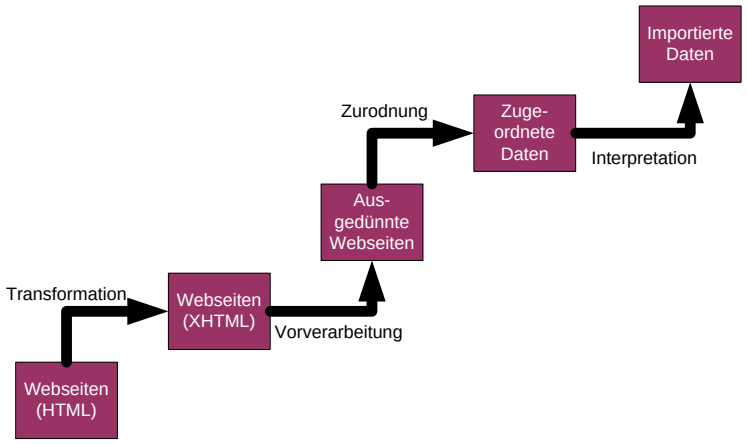

Abbildung 4.1: Schritte der Datenverarbeitung [Usam96]

schließlich das *page_structure.xml* sowie das *mapped_content.xml*.

Im Folgenden werden die einzelnen Module näher betrachtet.

4.1.1 Spider - Modul

Das Spider-Modul hat drei Aufgaben. Es muss:

1. die Seitenstruktur der Internetpräsenz erkennen
2. die Seitenattribute extrahieren
3. den relevanten Seiteninhalt extrahieren

Hierzu kann die Internetpräsenz über die Verweise in den einzelnen Dokumenten durchlaufen werden. Während des Durchlaufens der Internetpräsenz kann ein Seitenstrukturbaum erstellt werden. Um die benötigten Daten aus den einzelnen HTML-Seiten zu extrahieren, können direkt die HTML-Seiten geparst werden. Dies macht das Gesamtsystem startsystemunabhängig, denn jedes CMS liefert als Output HTML konforme Dateien. Beim Output der verschiedenen CMSe kann es sich auch um PHP-Dateien, XHTML Dateien oder ähnliches handeln. Doch diese Dateien werden im Folgenden, wie HTML-Dateien behandelt, da sie nur Ergänzungen zu HTML sind und dieser Ergänzungen nicht verarbeitet werden sollen.

Das Konzept des Spider-Moduls ist in Abbildung 4.3 dargestellt. Die HTML-Seiten werden aus dem Internet abgerufen und entsprechend der erkannten Seitenstruktur lokal abgespeichert.

Jede HTML-Seite kann nun in XHTML überführt werden und als XHTML-Datei von jedem XML-Parser verarbeitet werden.

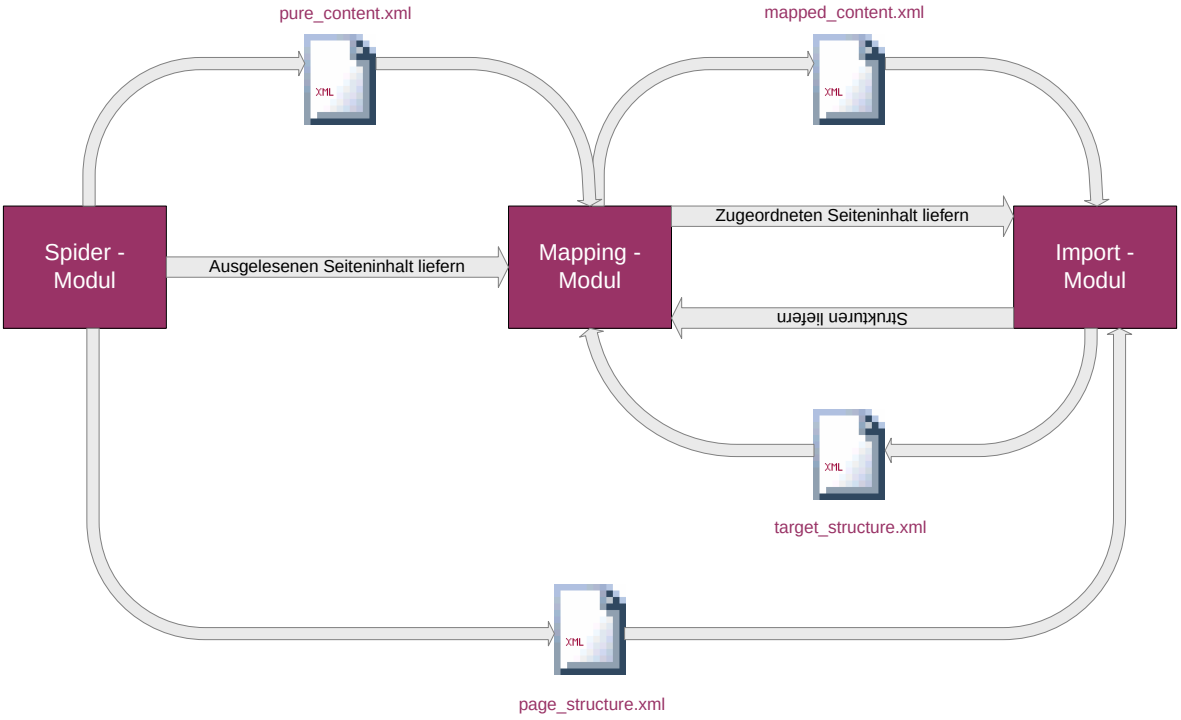

Abbildung 4.2: Gesamtsystem

Aus den konvertierten Dateien können aus dem HTML-Head die Seitenattribute ausgelesen werden. Mit den Seitenattributen und der erkannten Seitenstruktur kann das *page_stucture.xml* erstellt und an das Import-Modul übergeben werden. Das *page_stucture.xml* muss entsprechend folgender DTD aufgebaut sein.

Listing 4.1: DTD des page_structure.xml

```
<!DOCTYPE pagestructure
    [
    <!ELEMENT pagestructure (pages)>
    <!ELEMENT pages (page)*>
    <!ELEMENT page ((id, filename, path, localpath, parentid),
                    title?, descriptions?, keywords?)>
    <!ELEMENT keywords (keyword)+>
    <!ELEMENT descriptions (descrption)+>
    <!ELEMENT id                (#PCDATA)>
    <!ELEMENT filename  (#PCDATA)>
    <!ELEMENT path          (#PCDATA)>
    <!ELEMENT localpath (#PCDATA)>
    <!ELEMENT parentid  (#PCDATA)>
    <!ELEMENT title         (#PCDATA)>
    ]>
```

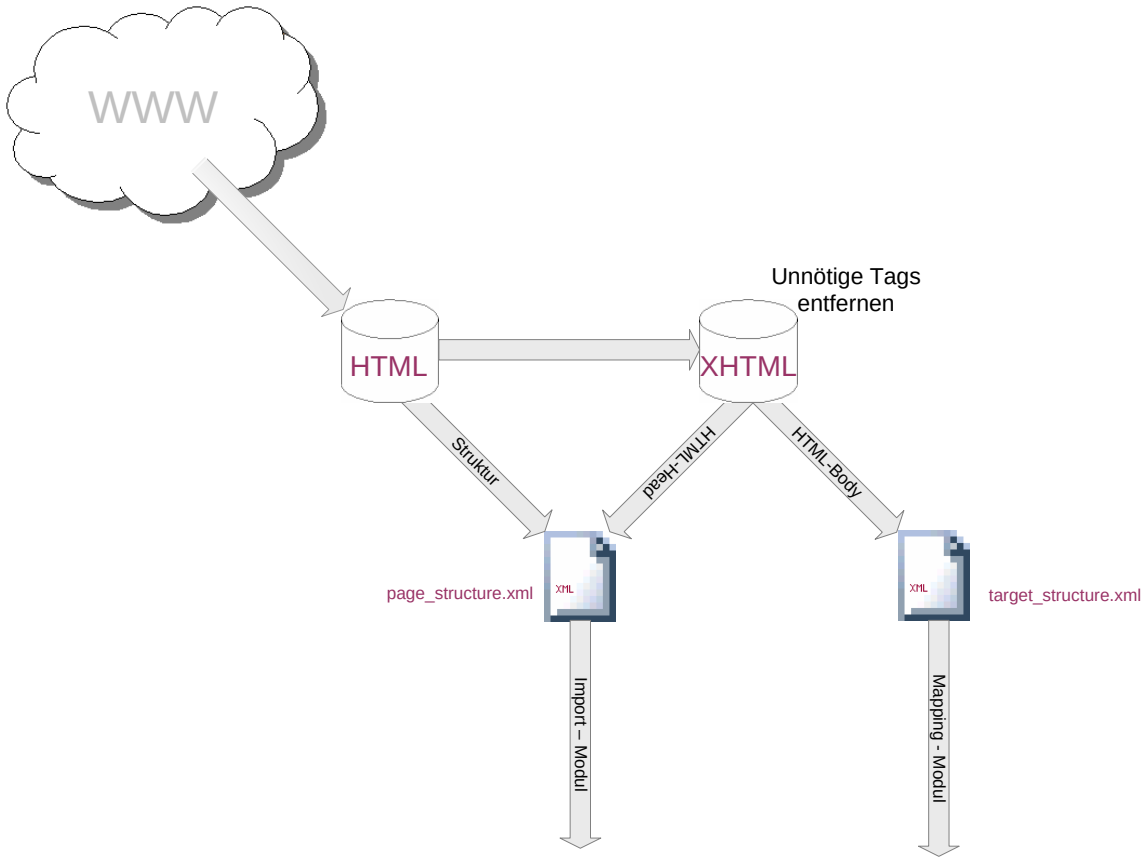

Abbildung 4.3: Spider-Modul

Das resultierende *target_structure.xml* könnte dann folgende Gestalt besitzen. Wobei die Startseite keine Parentpage besitzt. Daher wird für ihre parentid ein negativer Wert angegeben.

Listing 4.2: Beispiel eines page_structure.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<pagestructure>
    <page>
        <id>1</id>
        <filename>index</filename>
        <path>http://www.mydomain.de/index.html</path>
        <localpath>/Migration/index.xml</localpath>
        <parentid>-1</parentid>
        <title>Startseite</title>
    </page>
    <page>
        <id>2</id>
        <filename>aboutUs</filename>
        <path>http://www.mydomain.de/aboutUs.html</path>
        <lokalpath>E:/Migration/aboutUs/index.xml</lokalpath>
        <parentid>1</parentid>
        <title>Über uns</title>
    </page>
    ...
</pagestructure>
```

Aus dem HTML-Body können die nicht benötigten Tags, ein Teil der in Kapitel 3.4.3 vorgestellten "weichen Tags", sowie Skript-Tags entfernt werden. Alle Tags, die keine Rückschlüsse auf die Struktur des Dokumentes zulassen, sondern ausschließlich zur Festlegung der Darstellung dienen, werden nicht weiter benötigt.

Der so bereinigte Inhalt des Body-Tags kann daraufhin nach XML konvertiert und als *pure_content.xml* an das Mapping-Modul weitergegeben werden.

Das *pure_content.xml* enhält immer noch einen Teil des HTML-Codes der Seite. Allerdings wurde er in eine XML-konforme Form konvertiert, so dass das Mapping-Modul mit Hilfe der Standard-XML-Werzeugen darauf zugreifen kann.

Im folgenden wird davon ausgegangen, dass der Teil des HTML-Dokuments, der den zu importierenden Inhalt enthält durch Kommentare im HTML-Code gekennzeichnet ist und das *pure_content.xml* nur den zu importierenden Teil des Inhalts enthält. Dies ist ohne Probleme möglich, da für die Markierung nur wenige Templates im Ausgangssystem bearbeitet werden müssen.

4.1.2 Mapping - Modul

Das Mapping-Modul führt die Zuordnung zwischen dem zu portierenden Inhalt im *pure_content.xml* und den Strukturen des Ziel-CMS durch. Dies geschieht über ein fuzzybasiertes Regelsystem.

Es werden alle Inhaltsatome im *pure_content.xml*, zusammen mit ihren charakteristischen Attributen (bei Bilden, der Pfad zur Datei; bei Tabellen, die Inhalte der einzelnen Zellen, etc.), identifiziert und in einer Liste abgelegt. Den Inhaltsatomen in der Liste werden linguistische Variablen entsprechend der DTD des *target_structure.xml* in Kapitel 4.1.3.1 zugeordnet. Für Texte wird entschieden, ob sie kurz oder lang sind, bei Bildern, ob sie klein, mittel oder groß sind. Dies geschieht mit für das Ziel-CMS sinnvollen Schwellwerten.

Sind alle Texte und Bilder mit linguistischen Variablen versehen, so wird die Liste der Inhaltsatome in mehren Listen unterteilt und zwar so, dass in keiner Liste ein Element zweimal vorkommt. So darf keine Liste zwei Bilder enthalten oder zwei kurze Texte. Sie darf allerdings einen kurzen und einen langen Text enthalten. Jede der resultierenden Listen repräsentiert ein mögliches Inhaltselement.

Die Inhaltselemente müssen nun zugeordnet werden.

4.1.2.1 Zuordnung mittels einer Fuzzy - Regelbasis

Um die Zuordnung durchführen zu können muss entsprechend dem *target_structre.xml* eine Regelbasis aufgestellt werden.

$$
\begin{array}{llllll}
IF & shorttext & AND & longtext & & \\
 & & & THEN & text; & \\
IF & shorttext & AND & list & AND & list = unnumbered \\
 & & & THEN & listunnumbered; & \\
IF & shorttext & AND & list & AND & list = numbered \\
 & & & THEN & listnumbered; & \\
IF & shorttext & AND & longtext & AND & imagemedium \\
 & & & THEN & textimage; & \\
\ldots & & & & &
\end{array}
$$

Solche Regeln müssen für alle Inhaltselemente aufgestellt werden. Sind die Regeln aufgestellt, kann für jedes mögliche Inhaltselement überprüft werden, welchen Regeln es am ehesten entspricht. Entspricht das Inhaltselement vom Aufbau keiner der Regeln besonders gut, d.h. überschreitet es bei keiner Regel einen Schwellwert, muss es weiter aufgeteilt werden und die einzelnen Komponenten müssen geprüft werden.

Auf diese Weise können alle Inhaltsatome, zusammen geschlossen als Inhaltselemente, oder einzeln einem Inhaltselement des Zielsystems zugeordnet werden.

Diese Zuordnung kann in das *mapped_content.xml* überführt und an das Import-Modul übergeben werden.

4.1.2.2 Aufbau des mapped_content.xml

Der Aufbau des *mapped_content.xml* hängt stark vom Aufbau des *target_structure.xml* ab. Ein Beispiel ist in Listing 4.3 dargestellt. Eine DTD soll an dieser Stelle nicht angegeben werden, sie kann nach Bedarf dynamisch zu dem *mapped_content.xml* als interne DTD erstellt werden.

Listing 4.3: Beispiel eines mapped_content.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<mappedcontent>
    <contentelements>
        <!-- Einfaches Textelement bestehend aus Text mit Überschrift -->
        <text>
        <!--
        Hier muss es sich um einen Tag handeln,
        der den gleichen Namen hat, wie das
        Inhaltselement im Zielsystem: Der Name,
        der im <name>-Tag des targe\_structure.xml
        angegeben ist.
        -->
            <title>Das ist einen Überschrift</title>
            <text>
                Das ist der Text zur Überschrift,
                es muss sich um einen langen Text handeln,
                denn so war es spezifiziert.
                Das ist der Text zur Überschrift,
                es muss sich um einen langen Text handeln,
                denn so war es spezifiziert.
                Das ist der Text zur Überschrift,
                es muss sich um einen langen Text handeln,
                denn so war es spezifiziert.
            </text>
        </text>
        <!-- Text-mit-Bildelememnt bestehend aus Überschrift,
                    Text und Bild -->
        <textimage>
            <title>Das ist einen Überschrift zum Bild</title>
            <text>
                Das ist ein Text zum Bild.
                Das ist ein Text zum Bild.
                Das ist ein Text zum Bild.
                Das ist ein Text zum Bild.
                Das ist ein Text zum Bild.
                Das ist ein Text zum Bild.
            </text>
            <image>
                <scr>
                    http://www.mydomain.de/Pictures/einSchoenesBild.jpg
                </scr>
                <width>200</width>
                <height>100</height>
            </image>
        </textimage>
    </contentelements>
</mappedcontent>
```

4.1.3 Import - Modul

Das Import-Modul ist stark vom Zielsystem abhängig, da es dafür verantwortlich ist, die Strukturen des Zielsystems an das Mapping-Modul zu liefern und das Zielsystem soll variabel sein. Die Strukturen des Zielsystems werden in Form des *target_structure.xml* an das Mapping-Modul übergeben.

4.1.3.1 Aufbau des target_structure.xml

Der Aufbau des *target_structure.xml* soll hier spezifiziert werden:

Listing 4.4: DTD des target_structure.xml

```
<!DOCTYPE targetstructure
    [
    <!ELEMENT targetstructure
                (contentelements)>
    <!ELEMENT contentelements
                (contentelement)+>
    <!ELEMENT contentelement
                (name, (title, text, table, list, image)*)>
    <!ELEMENT name  (#PCDATA)>
    <!ELEMENT title EMPTY>
    <!ATTLIST title type
                (shorttext|longtext) "shorttext"    #REQUIRED>
    <!ELEMENT text  EMPTY>
    <!ATTLIST text  type
                (shorttext|longtext) "longtext"     #REQUIRED>
    <!ELEMENT table EMPTY>
    <!ATTLIST table maxrows     CDATA               #REQUIRED>
    <!ATTLIST table maxcols     CDATA               #REQUIRED>
    <!ELEMENT list  EMPTY>
    <!ATTLIST list  type
                (numbered|unnumbered) "numbered"    #REQUIRED>
    <!ATTLIST table maxitems    CDATA               #REQUIRED>
    <!ELEMENT image EMPTY>
    <!ATTLIST image size
                (small|medium|large) "medium"       #REQUIRED>
    ]>
```

Das resultierende *target_structure.xml* könnte dann folgende Gestalt besitzen:

Listing 4.5: Beispiel eines target_structure.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<targetstructure>
    <contentelements>
        <!-- Einfaches Textelement bestehend aus Text mit Überschrift -->
        <contentelemnet>
            <name>text</name>
            </title type = "‘shorttext"’>
            </text type = "‘longtext"’>
        </contentelement>
        <!-- Text-mit-Bildelememnt bestehend aus Überschrift,
                    Text und Bild -->
        <contentelemnet>
            <name>textimage</name>
            </title type = "‘shorttext"’>
            </text type = "‘longtext"’>
            </image type = "‘medium"’>
        </contentelement>
        <!-- Punktliste -->
        <contentelemnet>
            <name>list</name>
            </title type = "‘shorttext"’>
            </list type = "‘unnumbered"’>
        </contentelement>
    </contentelements>
</targetstructure>
```

Das implementierte Import-Modul wird speziell für Typo3 entwickelt und daran angepasst.
Das System wurde so entworfen werden, dass es mit möglichst wenig Aufwand für andere Content Management Systeme erweitert werden kann.
Das Import-Modul legt die Webseiten entsprechend dem *page_structure.xml* im Zielsystem an und importiert den gemappten Inhalt, wie er im content_xml geliefert wird als Inhaltselemente für die einzelnen Seiten in das Zielsystem.

4.2 Zusammenfassung

In diesem Kapitel wurde der Lösungsansatz vorgestellt. Es handelt sich hier um ein modulares System, das sich aus den drei Hauptkomponenten: Spider-Modul, Mapping-Modul und Import-Modul, sowie den Schnittstellendokumenten *page_structure.xml*, *pure_content.xml*, *mapped_content.xml* und *target_structrue.xml* zusammensetzt.

Eine fuzzybasierte Regelbasis, die Teil des Mapping-Moduls ist, bildet den Kern des Systems.

5. Umsetzung

Das im vorherigen Kapitel beschriebene System wurde im Verlauf dieser Arbeit zum Teil in Java und zum Teil in PHP umgesetzt (siehe Abbildung 5.1). Das Spider-Modul und das Mapping-Modul wurden in Java implementiert. Es wurde die zu Beginn der Arbeit aktuellste Version Java 1.5 Update 6 verwendet. Als Zielsystem diente Typo3. Das Import-Modul wurde als Typo3 Backendextension entwickelt, auch hier wurde die zu Beginn der Arbeit aktuellste Version 3.8.1 von Typo3 verwendet.

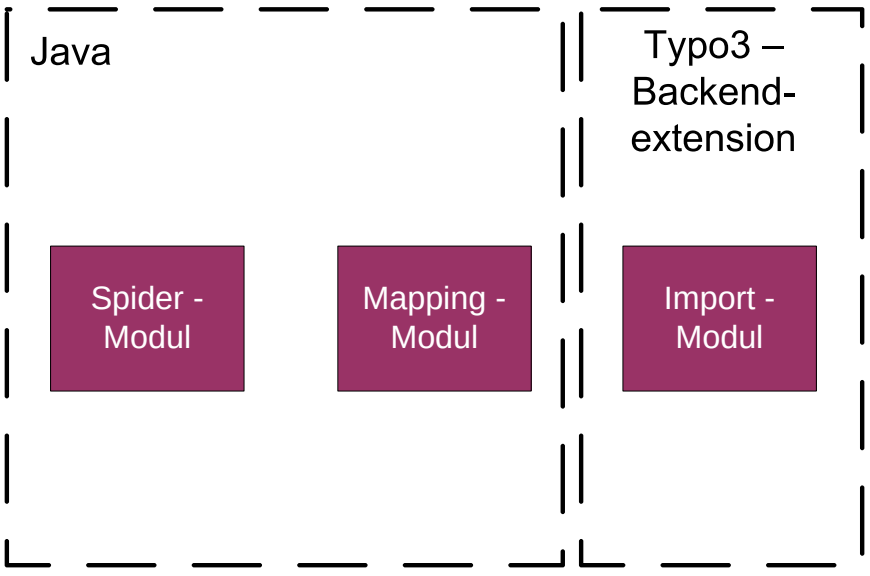

Abbildung 5.1: Verwendete Programmiersprachen

Typo3 ist ein frei konfigurierbares OpenSource CMS und steht unter der General Public License (GPL)[1]. 1998 wurde Typo3 kommerziell von der dänischen Agentur

[1] http://www.fsf.org/licensing/licenses/gpl.html

Superfish.com entwickelt und vertrieben. Anfang 1999 trennt sich die Agentur von Typo3 und Kasper Skarhoj, dem späteren Entwickler von Typo3. Typo3 wurde von Kasper Skarhoj komplett neu entwickelt und zu einem OpenSource Projekt unter der GPL erklärt. Bis Mitte 2000 entwickelte Kasper Skarhoj Typo3 selbständig. Nach der Veröffentlichung schloss sich eine immer noch wachsende Internetgemeinde zusammen, die Typo3 ständig um Funktionalität erweitert. [Meye05] Das Import-Modul stellt eine solche Erweiterung dar.

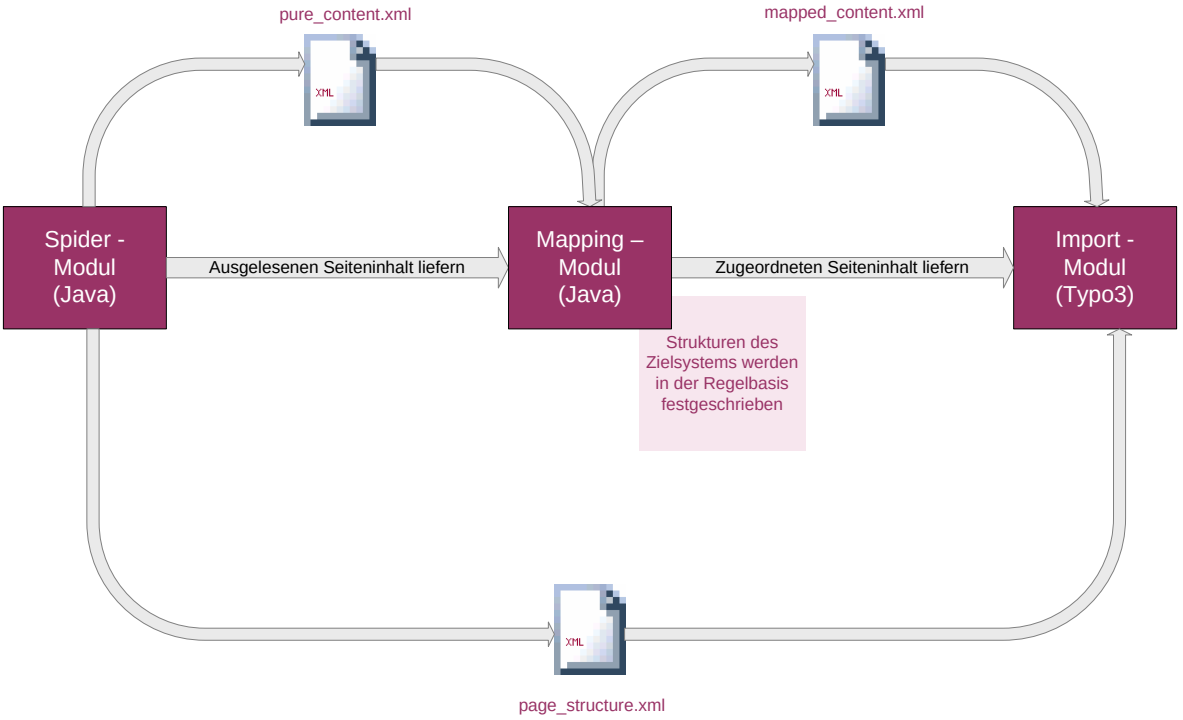

Abbildung 5.2: Gesamtsystem

Der Hauptunterschied zwischen dem im vorherigen Kapitel vorgestellten und dem implementierten System besteht, wie in Abbildung 5.2 zu sehen, in der Übergabe der Strukturen des Zielsystems vom Import-Modul zum Mapping-Modul. Auf diese Übergabe wurde verzichtet und es wurde der Aufbau der Standardelemente des CMSes Typo3 direkt in die Regelbasis des Mapping-Moduls codiert.

Eine weiter Einschränkung ist, dass ausschließlich HTML-Seiten verarbeitet werden, d.h. Dateien, die auf „.html“ enden. So kann die umgesetzte Implementierung PHP-Seiten, XHTML-Seiten o.ä. nicht verarbeiten.

Bilder werden während der Migration nicht zwischengespeichert. Die Dateipfade zu den Bildern werden vom Spider-Modul umgeschrieben, so dass sie vom Import-

Modul vom Liveauftritt in den Pfad „fileadmin/images“ geladen werden können.

In den folgenden Kapiteln wird die Umsetzung der einzelnen Module beschrieben. Beim Mapping-Modul wird besonderes auf die Umsetzung der Zuordnungsstrategie eingegangen.

Es wird davon ausgegangen, dass der zu portierende Inhalt in den alten Templates gekennzeichnet wurde und sich zwischen den Kommentar-Tags „`<!- Content starts ->`“ und „`<!- Content ends ->`“ befindet.

5.1 Spider - Modul

Das Vorgehen des Spider-Moduls ist in Abbildung 5.3 dargestellt. Das Spider-Modul legt zu Beginn die HTML-Seiten der zu migrierenden Internetpräsenz lokal ab. Um das zu erreichen wird das in Kapitel 3.4.1 vorgestellte Verfahren, zum Spidern mit Warteschlangen, verwendet. Um die Verweise in den HTML-Seiten zu finden werden die Seiten nach dem String „href“ durchsucht. Dabei wird die Groß- und Kleinschreibung ignoriert.

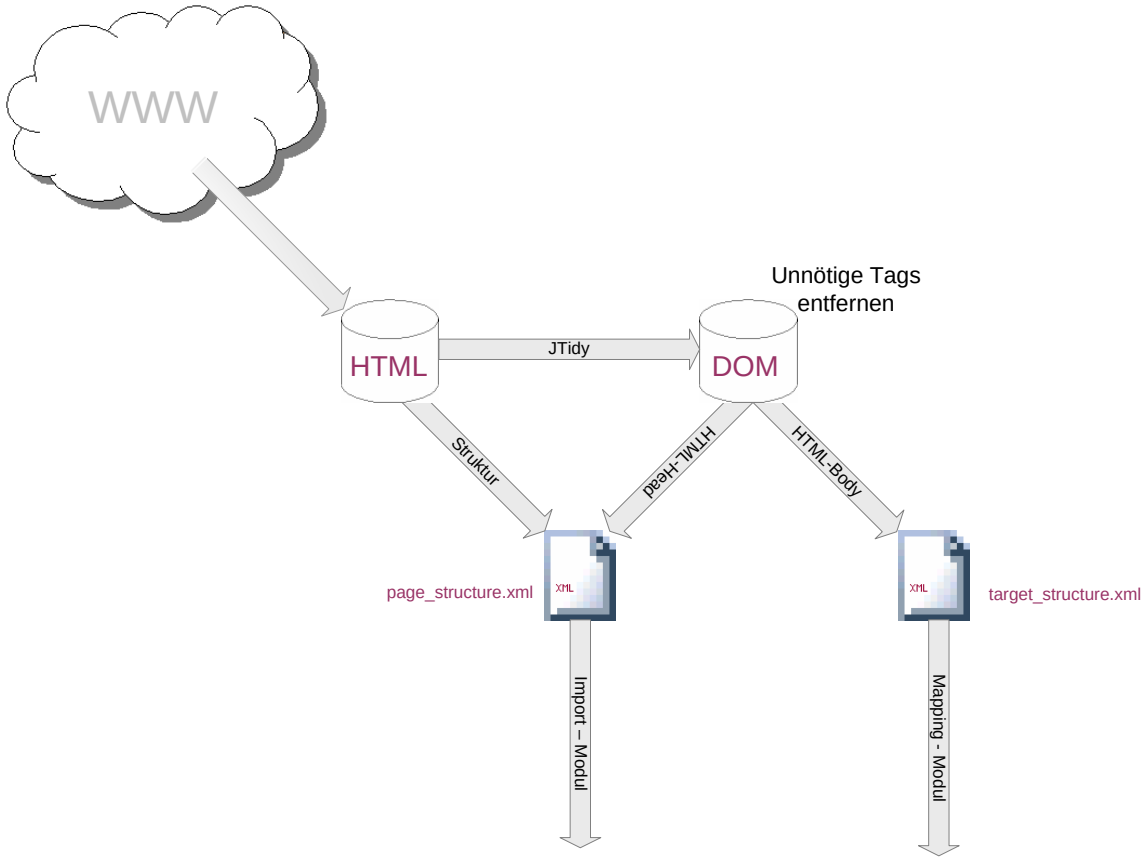

Abbildung 5.3: Spidermodul

Vor der lokalen Speicherung der HTML-Seiten werden sämtliche in der Datei vorkommenden URLs so umgeschrieben, dass sie keine relativen, sondern absolute Adressen

sind. Auf der Seite der Universität Karlsruhe wird beispielsweise auf der Homepage „`http://www.uni-karlsruhe.de/`“ ein Bild angezeigt. Dieses Bild hat im HTML-Code die Adresse „`/Uni/img/news/personen_1.jpg`“. Diese Adresse ist eine relative Adresse, die sich auf die Seite, auf der das Bild angezeigt werden soll bezieht. Die Adresse wird in „`http://www.uni-karlsruhe.de/Uni/img/news/personen_1.jpg`“ umgeschrieben. Für das Umschreiben der URLs existieren folgende Regeln:

- Ein relativer Pfad bezieht sich immer auf die Adresse des HTML-Dokumentes, das den Pfad enthält.
- Beginnt der relative Pfad mit dem String „`../`“, so bedeutet dies, dass von der aktuell verarbeiteten Seite aus, im Pfad eine Ebene nach oben gegangen werden muss. Kommt der String mehrfach am Anfang des relativen Pfades vor, muss für jedes Vorkommen eine Ebene in Richtung Wurzel gegangen werden.

Weiter werden alles Skript- und Multimedia-Tags, wie sie in Tabelle 3.1 aufgezählt werden, zusammen mit ihren enthaltenen Tags oder Texten entfernt.

Für die lokale Ablage der einzelnen HTML-Seiten wird eine Ordnerstruktur aufgebaut, die die Vater-Kind-Beziehung der einzelnen Seiten widerspiegelt. Dabei wird für jede gefundene Seite eine HTML-Datei angelegt. Hat diese Seite Unterseiten wird ebenfalls ein Ordner angelegt und dieser Ordner enthält alle Kindseiten und deren Ordner mit Kindseiten, falls Kindseiten existieren. Dieses Vorgehen wird wiederholt, bis eine Seite keine Kindseiten mehr hat. Eine Vater-Kind-Beziehung wird hier gemäß der Begriffserklärung in Kapitel 3.1 definiert. Als Datei- und auch Ordnername wird der Name der Datei auf dem Webserver verwendet. Wurde die Seite über HTTP-GET mit Parametern aufgerufen, so wird der Titel im `<title>`-Tag der HTML-Seite als Dateinamen verwendet. Hat die Seite auf dem Webserver keinen Dateinamen und enthält die Seite keinen `<title>`-Tag, dann enspricht sie nicht der HTML-Spezifikation und eine weitere Verarbeitung ist nicht möglich.

Nach der lokalen Ablage der HTML-Seiten kann das *page_stucture.xml* erstellt werden. Hierfür wird jedem HTML-Dokument zusammen mit dessen eindeutigem Pfad eine ID zugewiesen. Das *page_structure.xml* wird, gemäß der in Listing 4.1 aufgeführten DTD, erstellt (siehe Abbildung 5.4).

Hierfür werden die bereits lokal abgelegten HTML-Dokumente über ihren Verzeichnisbaum abgelaufen. Zum Abschreiten der Dateien im Verzeichnisbaum wird die Reihenfolge der Breitensuche verwendet. Im Spider-Modul wird eine Liste von Page-Objekten aufgebaut, wobei ein Page-Objekt gerade die Werte hält, die für das Schreiben des *page_structure.xml* benötigt werden. Die ID der Seite, die ID der Vaterseite, die über deren Pfad in der zuvor gespeicherten Liste abrufbar ist, den Dateinamen, der Datei auf dem Server (dieser muss nicht eindeutig sein), den Pfad der Datei auf dem Webserver und den lokalen Pfad der Datei. Um weitere benötigte Werte, wie Titel, Beschreibung und Schlüsselwörter aus den HTML-Dateien zu extrahieren wird die HTML-Datei mit Hilfe der offene Bibliothek, JTidy[2], für Java, in den DOM

[2] http://jtidy.sourceforge.net/

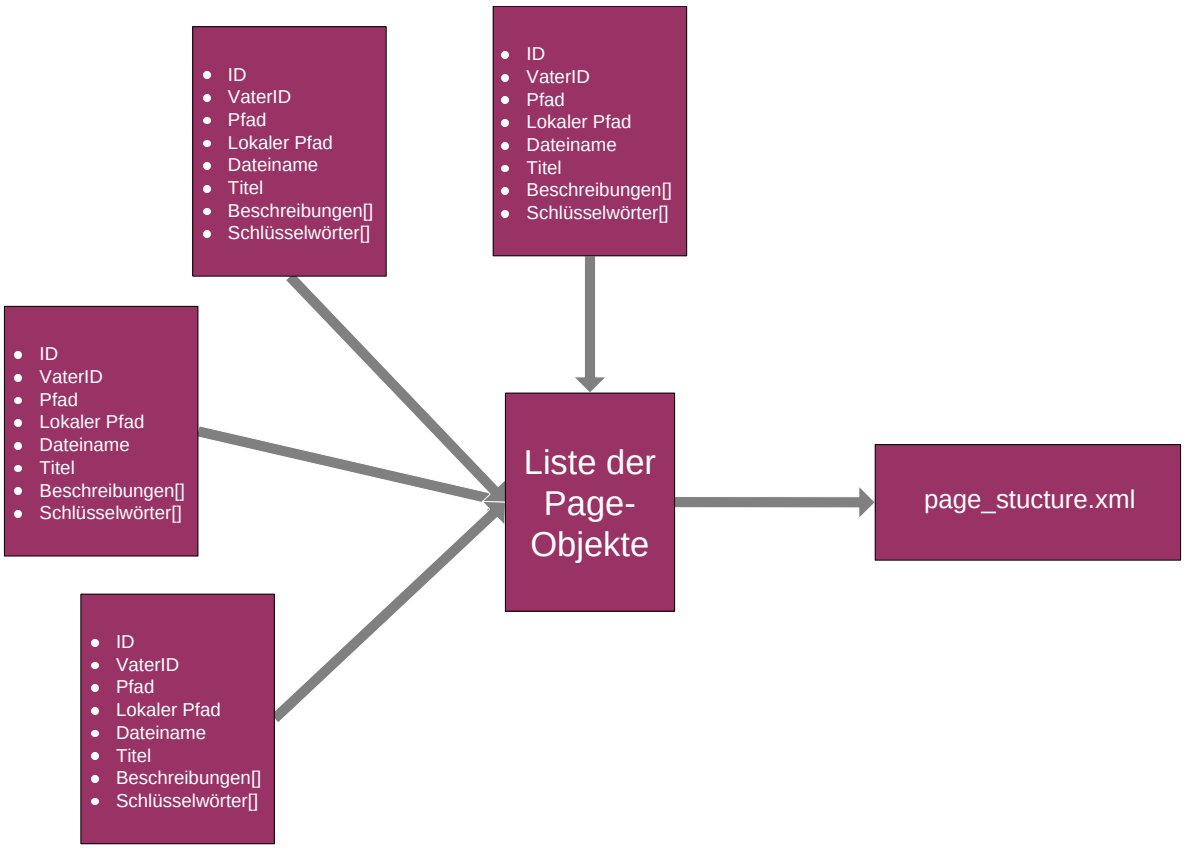

Abbildung 5.4: Erstellung des page_structure.xml

geladen. JTidy steht unter der GPL.

Mit JTidy ist es möglich HTML-Dokumente in den DOM zu laden. Dadurch kann auf sie zugegriffen werden, als wären sie valides XML. Dabei werden nicht XML-konforme Knoten als Textknoten betrachtet. Das führt dazu, dass nicht korrektes HTML zwar verarbeitet werden kann, das Ergebnis jedoch nicht vorhersehbar ist.

Wurde das HTML-Dokument in den DOM geladen kann über die gängigen Befehle auf den Titel und die Metadaten zugegriffen werden. Die extrahierten Daten können dem entsprechenden Page-Objekt hinzugefügt werden.

Wurden die benötigten Daten aus dem HTML-Header extrahiert, kann alles aus dem Dokument entfernt werden, was für die künftige Weiterverarbeitung nicht mehr benötigt wird.

Im HTML-Dokument wird nach den Strings „`<!- Content starts ->`“ und „`<!- Content ends ->`“ gesucht. Alles, was sich vor und nach diesen Markierungen befindet wird gelöscht. Um spätere Parserprobleme zu vermeiden werden Textknoten, die mit einem öffnenden Tag „`<`“ beginnen, gelöscht, es sei den es handelt sich um einen `<img>`-Tag. Durch diesen Löschvorgang gehen zwar Informationen, wie beispielsweise

Zeilenumbrüche, verloren, doch die folgende Verarbeitung wird erheblich vereinfacht.

Das restliche Dokument wird mit Hilfe von JTidy als XML-Datei abgespeichert. Die Datei wird im selben Verzeichnis und unter dem gleichen Namen abgelegt wie das HTML-Dokument auf dem es basiert. Das so entstandene XML entspricht dem in Kapitel 4.1.1 erwähnten *pure_content.xml*.

Für die Erstellung des *page_structure.xml* sind alle benötigten Informationen in der Liste der Page-Objekte vorhanden. Es wird erstellt und ausgegeben. Das *page_structure.xml* kann an das Import-Modul übergeben werden.

5.2 Mapping - Modul

Das Mapping-Modul identifiziert die einzelnen Contentelemente im *pure_content.xml*. Hierfür läd es das *pure_content.xml* in den DOM.

Zuerst werden alle Tabellen-Knoten überprüft, ob es sich bei einer Tabelle um ein Designelement oder eine Inhaltstabelle handelt, die auch als solche in das CMS importiert werden soll. Es wurde für die Implementierung des Mapping-Moduls angenommen, dass Tabellen, die nicht als Designelemente betrachtet werden, keine weiteren Tabellen oder andere Containerelemente enthalten.

Ist ein Tabellenkopf (<th>-Tag) angegeben, wird davon ausgegangen, dass es sich um eine Inhaltstabelle handelt. Enthält eine Tabelle keine andere Tabelle und ist kein Tabellenkopf angelegt, soll als letztes Charakteristikum dienen, dass mindestens eine Spalte oder eine Zeile der Tabelle Zahlen enthält. Dies wurde abgeschätzt, indem die Tabellenzellen, die Zahlen enthalten, gezählt werden. Ist diese Anzahl der Zellen, die Zahlen enthalten, größer als die Anzahl der Zeilen der Tabelle oder größer als die Anzahl der Spalten in der Tabelle, wird davon ausgegangen, dass sie sich in derselben Zeile oder Spalte befinden und es sich um eine Inhaltstabelle handelt. Der Algorithmus zu Tabellenidentifikation ist in Prozedur 2 nochmals zusammengefasst.

Tabellen, die als Inhaltstabellen identifiziert wurden, wird das Attribut `contenttable = „yes“` hinzugefügt.

Im nächsten Schritt werden nicht benötigte Elemente entfernt. Darunter fallen alle Bilder, deren Höhe oder Breite die größe 50px unterschreitet, da in diesem Fall davon ausgegangen wird, dass es sich um Platzhalter handelt. Ebenso entfernt werden Knoten, die ausschließlich Leerzeichen enthalten, die in HTML durch den String „ “ editiert werden, und Textknoten, die drei oder weniger Zeichen enthalten und nicht Teil einer Inhaltstabelle oder einer Liste sind.

Es werden die Contentatome im *pure_content.xml* identifiziert. Insbesondere gesucht werden Tabellen, Listen, Texte, Bilder und Links. Für alle diese Ausprägungen von Contentatomen existieren Klassen, als deren Objekte sie instanziiert werden können.

```
Data : tableBranch
Result : boolean isContenttable
isContenttable = false; if tableBranch enthält Containerelement then
 |  isContenttable = false;
else
 |  if tableBranch enthält <th>-Tag then
 |   |  isContenttable = true;
 |  else
 |   |  numberOfRows = getNuberOfRows(); numberOfColumns =
 |   |  getNuberOfColumns(); numberOfNumberEntries =
 |   |  getNumberOfNumberEntries(); if numberOfNumberEntries >
 |   |  numberOfColumns OR numberOfNumberEntries >
 |   |  numberOfNumberEntries then
 |   |   |  isContenttable = true;
 |   |  else
 |   |   |_ isContenttable = false;
 |   |_
 |_
return isContenttable;
```

Prozedur `isTable`

Die Klassenhierarchie, sowie die Attribute der einzelnen Klassen sind in Abbildung 5.5 dargestellt.

Es wird eine Liste der Contentatome erstellt. Hierzu wird der Contentbaum, der durch das Laden des *pure_content.xml* in den DOM erstellt wurde, nach dem Tiefensuchealgorithmus durchlaufen. Jedes Mal, wenn ein Contentatom gefunden wurde, wird es in die Liste der Contentatome geschrieben. Contentatome sind Texte, Bilder, Links, Listen und Tabellen.

Wurde das Dokument komplett durchlaufen, ist die Liste der Contentatome erstellt. Es werden die linguistischen Variablen für die Textlängen und die Bildergrößen gesetzt. Die Schwellwerte und die Fuzzy-Mengen für die Texte sind in Abbildung 5.6 dargestellt, die für Bilder in Abbildung 5.7
.

Texte, die drei oder weniger Worte enthalten gehören eindeutig zur Menge der kurzen Texte. Unter einem Wort versteht sich eine Zeichenfolge, die durch Leerstellen oder Tags von anderen Zeichenfolgen getrennt ist. Texte, die mehr als elf Wörter enthalten, gehören eindeutig zur Menge der langen Wörter. Die Texte dazwischen lassen sich nicht eindeutig zuordnen, daher gehören sie zu einem gewissen Grad in beide Mengen.

Bilder werden in drei Mengen unterteilt: klein, mittel und groß. Es wurde diese unterscheidung getroffen um wirklich nur mittlere Bilder zu importieren. Sollte das System um Inhaltselemente erweitert werden, die diese Unterscheidung benötigen, wie beipielsweise Popups in einer Bildergallerie, so sind die Fuzzymengen bereits definiert.

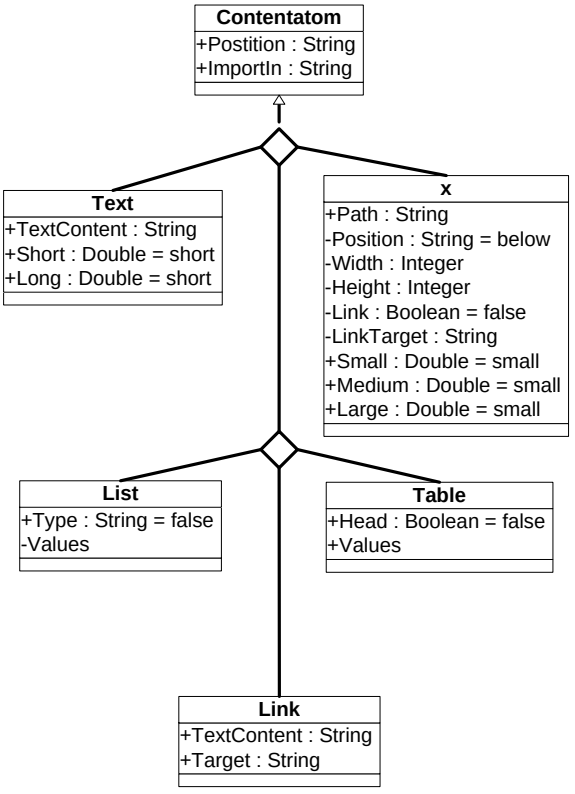

Abbildung 5.5: Contentatome und Subklassen

Die auf diese Weise ermittelten linguistischen Variablen werden als Attribute der Objekte der Klassen Text bzw. Image abgelegt.

Um Contentelemente zu erhalten wird die Liste in kleinere Listen aufgeteilt. Die Aufteilung erfolgt, indem die Liste durchwandert wird und sobald auf ein Contentatom gestoßen wird, das die Liste bereits enthält, wird eine neue Liste begonnen. Bei auftreten eines Textes, dessen Zugehörigkeit zu der Menge der kurzen Texte den Wert 0,5 überschreitet, wir ebenfalls eine neue Liste begonnen. Das Beginnen der neuen Liste beim Auftauchen eines kurzen Textes ist sinnvoll, da kurze Texte im Zielsystem immer die ersten Contentatome der Contentelemente sind. Nach dieser Aufteilung enthält keine der Teillisten zwei Contentatome des gleichen Typs.

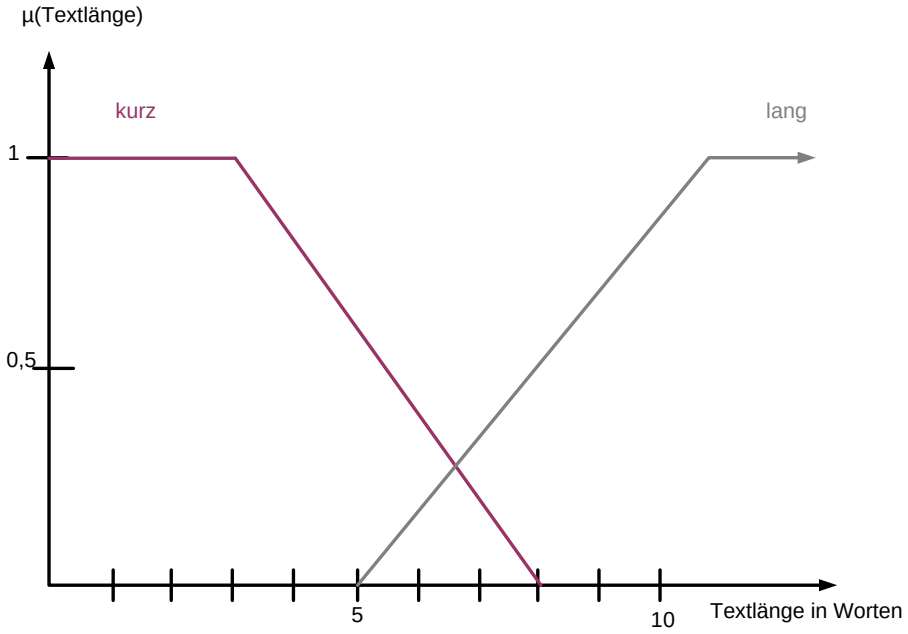

Abbildung 5.6: Fuzzy-Mengen für Texte

Es wird davon ausgegangen, dass jede Teilliste ein Contentelement repräsentiert. Jede Liste kann maximal acht Elemente enthalten:

1. kurzer Text
2. langer Text
3. kleines Bild
4. mittleres Bild
5. großes Bild
6. Liste
7. Tabelle
8. Link

Wobei ein Text als kurz oder lang angesehen wird, je nach dem, welcher Zugehörigkeitsgrad zu welcher der beiden Mengen höher ist.

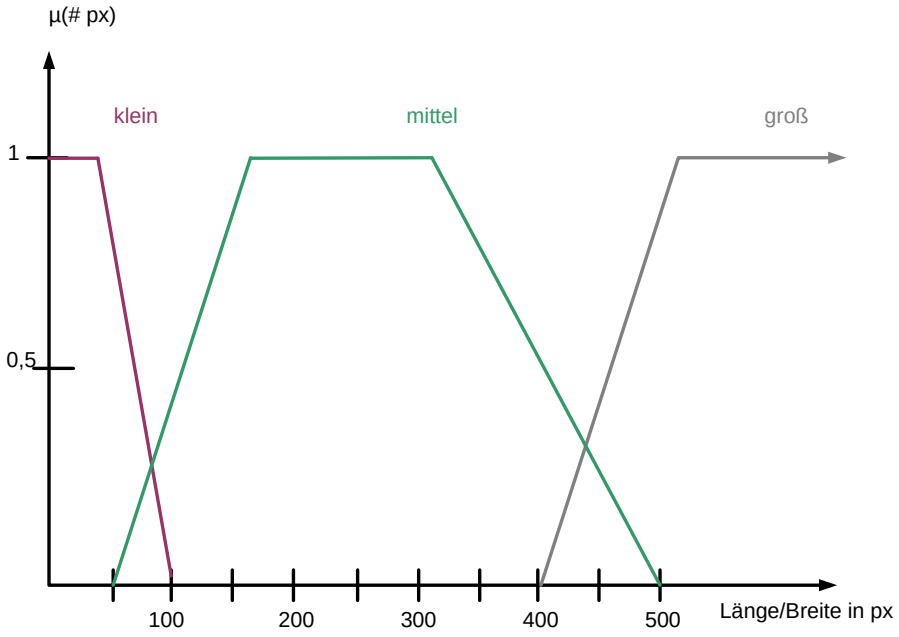

Abbildung 5.7: Fuzzy-Mengen für Bilder

In Typo3 existieren standardmäßig, ohne installierte Erweiterungen, folgende Contentelemente in die importiert werden kann:

1. Text (Überschrift und Text)
2. Text mit Bild (Überschrift, Text und Bild mit Dimensionen)
3. Bild (Überschrift, Bild mit Dimension)
4. Liste (Überschrift, Listenelemente, Listentyp)
5. Tabelle (Überschrift und Tabelle)

Da keines der Contentelemente im Zielsystem aus mehr als drei Contentatomen besteht, werden die Listen nochmals aufgeteilt. Die Aufteilung erfolgt so, dass die Contentelemente maximal drei Contentatome enthalten, wenn ein Bild vorhanden ist, andernfalls nur zwei.

Die Contentelemente werden nun in die erste Regel der Regelbasis eingegeben. Die Regelbasis ist entsprechen den Zielstrukturen entworfen. Die erste Regel hat folgende Gestalt:

$$R1: \quad IF(ce\,[1]\,IS\ \widetilde{kurzer}Text)AND(ce\,[2]\,IS\ \widetilde{kurzer}Text)AND(ce\,[3]\,IS\ \widetilde{mittleres}Bild)\\ THEN(ce\ IS\ Text\widetilde{mit}Bild)$$

Ist der Wert $mittleresBild > 0,5$ so wird davon ausgegangen, dass es sich um das Contentelement „Text mit Bild“ handelt. Andernfalls wird das Contentelement weiter aufgespaltet.

Handelt es sich beim ersten Contentatom des Contentelementes um ein Bild, so wird dies als eigenständiges Bild angesehen und als solches in die Liste der Contentelemente geschrieben. Der verbleibende Teil des Contentelementes wird wieder an die Regelbasis weitergegeben. Ist das erste Element kein Bild, so wird das letzte Contentatom des Contentelementes entfernt und vorne an das folgende Contentelement angefügt. Für dieses Element muss nun geprüft werden ob es bezüglich Größe und Mitgliedern noch den Regeln entspricht (drei Contentatome nur wenn ein Bild dabei ist). Ansonsten muss hier das letzte Contentatome entfernt werden und an das folgende Contentelement angehängt werden. Dieser Vorgang muss solange fortgeführt werden, bis alle Contentelemente wieder die richtige Gestalt haben. Im Folgenden sind die Regel der Regelbasis für die zweidimensionalen Contentelemente aufgeführt:

$$
\begin{aligned}
R2: \quad & IF(ce\,[1]\,IS\ \widetilde{kurzerText})AND(ce\,[2]\,IS\ \widetilde{langerText}) \\
& THEN(ce\ IS\ \widetilde{Text}) \\
R3: \quad & IF(ce\,[1]\,IS\ \widetilde{kurzerText})AND(ce\,[2]\,IS\ \widetilde{mittleres Bild}) \\
& THEN(ce\ IS\ \widetilde{Bild}) \\
R4: \quad & IF(ce\,[1]\,IS\ \widetilde{kurzerText})AND(ce\,[2]\,IS\ \widetilde{List}) \\
& THEN(ce\ IS\ \widetilde{Liste}) \\
R5: \quad & IF(ce\,[1]\,IS\ \widetilde{kurzerText})AND(ce\,[2]\,IS\ \widetilde{Tabelle}) \\
& THEN(ce\ IS\ \widetilde{Tabelle})
\end{aligned}
$$

Überschreitet das maximale Ergebnis eines Contentelementes den Wert 0,5 nicht, so wird es abermals geteilt. Nach dieser Teilung enthält es nur noch ein Element, das entsprechend folgender Regeln zugeordnet wird:

$$
\begin{aligned}
R6: \quad & IF(ce\,[1]\,IS\ \widetilde{kurzerText})OR(ce\,[1]\,IS\ \widetilde{langerText}) \\
& THEN(ce\ IS\ \widetilde{Text}) \\
R7: \quad & IF(ce\,[1]\,IS\ \widetilde{mittleres Bild}) \\
& THEN(ce\ IS\ \widetilde{Bild}) \\
R8: \quad & IF(ce\,[1]\,IS\ \widetilde{List}) \\
& THEN(ce\ IS\ \widetilde{Liste}) \\
R9: \quad & IF(ce\,[1]\,IS\ \widetilde{Tabelle}) \\
& THEN(ce\ IS\ \widetilde{Tabelle}) \\
R10: \quad & IF(ce\,[1]\,IS\ \widetilde{Link}) \\
& THEN(ce\ IS\ \widetilde{Text})
\end{aligned}
$$

Zu jedem Contentelement, das zugeordnet werden konnte wird das Ergebnis der Zuordnung gespeichert. Konnte es nicht zugeordnet werden wird es ignoriert und damit nicht in das Zielsystem importiert.

Für jedes zugeordnete Contentelement wird ein Objekt einer Subklasse der Klasse Contentelement erstellt, welches eine Liste von Contentatomen hält, sowie den Namen des Contentelementes im Zielsystem.

Alle diese Objekte der Subklassen der Klasse Contentelement werden wiederum in einer Liste geführt. Aufgrund der anfangs benutzen Reihenfolge zum Durchschreiten des Contentbaumes, ist die Reihenfolge der Contentelemente in der Liste gleich derer auf der Webseite.

Aufgrund der Liste der Contentelemente, kann das *mapped_content.xml* geschrieben werden. Der Aufbau entspricht der Beschreibung in Kapitel 4.1.2.2.

5.3 Import - Modul

Das Import-Modul wurde als Backenedextension des CMS Typo3 implementiert. Die Sprache in der Typo3 entwickelt wurde ist PHP, daher wurde auch das Import-Modul in PHP implementiert.

Da Typo3 sämtliche Daten in einer mySQL-Datenbank hält, gab es zwei Möglichkeiten das Import-Modul zu implementieren. Die erste Möglichkeit besteht darin, die Daten direkt in die Datenbank zu schreiben. Die Felder und Tabellen, die die Daten der entsprechenden Contentelemente enthalten, können mit Hilfe der Dokumentation gefunden werden.

Dieses Vorgehen, ist allerdings fehleranfällig, da keine Tabelle übersehen werden darf und sich die Tabellenstrukturen beim nächsten Typo3-Update eventuell ändern könnten.

Daher wurde hauptsächlich die zweite Möglichkeit verwendet. Sie benutzt zum anlegen der Seiten und der Contentelemente, soweit als möglich, die Typo3-API. Contentelemente, die nicht über die Typo3-API angelegt werden konnten, wurden direkt in die Datenbank importiert.

So werden mit Hilfe der Typo3-API die Seiten angelegt, wie sie im *page_structure.xml* übergeben wurden. Dabei werden Vater-Kind-Beziehungen berücksichtigt und Titel, sowie die Metadaten für die Beschreibung und die Schlüsselwörter gesetzt. Für das Anlegen der Seiten ist die fehlende Angabe des Seitentemplates nicht hinderlich, da das Seitentemplate in Typo3 im Nachhinein noch für bestimmte Seitenzweige gesetzt werden kann. Damit ist die Auswahl des Seitentemplates nicht Teil der Contentmigration sondern Teil der Konfiguration.

Nach dem Erstellen der Seiten werden die Contentelemente der jeweiligen Seite auf dieser angelegt, wie sie im *mapped_content.xml* übergeben wurden. Implementiert wurde nur der Import der Contentelemente, die vom Typ „Text“ waren. Die Implementierung des Imports der Seiteninhalte war aufwendig und für die Erfolgsprüfung des Verfahrens nur wenig relevant.

5.4 Zusammenfassung

In diesem Kapitel wurde auf die Umsetzung des im vorherigen Kapitel vorgestellten Systems eingegangen. Die Implementierung der einzelnen Module wurde vorgestellt und auf die nicht implementierten Teile des Entwurfs hingewiesen. Besonders eingegangen wurde auf die Identifikation von Inhaltstabellen und die Zuordnung der Contentelemente zu den Strukturen des Zielsystems.

6. Evaluierung

In diesem Kapitel wird das in Kapitel 4 entworfene und in Kapitel 5 umgesetzte System beurteilt. Es wird versucht ein Maß für das Funktionieren der Contentmigration zu finden, mit Hilfe derer die Beurteilung durchgeführt werden kann. Ein solches Maß ist für die Migration der Webseiten die Anzahl der korrekt migrierten Seiten, bezüglich ihrer Position im Contentbaum und der korrekten Übernahme von Titel und Metadaten. Für die Seiteninhalte kann als Maß die Anzahl der korrekt identifizierten Contentelemente zusammen mit den noch nötigen Änderungen dienen.

6.1 Seitenmigration

Um die Seitenmigration zu bewerten wurden verarbeitbare Seiten aus dem Netzt für Testzwecke verwendet. Soche Seiten waren:

- `http://www.namics.com`
- `http://www.gustain.de`
- `http://www.eldoron.de`

Wobei es sich bei dem Internetauftritt der Namics AG `http://www.namics.com` um 591 einzelne HTML Seiten handelt. Hinter dem Internetauftritt steckt bereits das CMS Typo3. Die Seiten sind fast vollständig in validem HTML abgefasst. Die Vater-Kind-Beziehungen der Seiten können bei diesem Internetauftritt über den Dateipfad der aufgerufenen Seite abgelesen werden.

Aufgrund des nahezu fehlerfreien HTML-Codes der Seiten, konnten sie alle verarbeitet werden. Die Vater-Kind-Beziehungen der Seiten konnten korrekt extrahiert werden, ebenso die Titel. Eine Angabe der Metadaten war in den meisten Fällen nicht möglich, da keine Metadaten in den Seiten angegeben waren.

Der Internetauftritt unter der Domäne `http://www.gustain.de` besteht aus nur sechs Seiten und stellt ein sehr übersichtliches Testszenario dar. Hierbei handelt es um einen Internetauftritt, der nicht von einem CMS generiert wurde, sondern um „handgemachte“ Seiten. Der HTML-Code der Seiten ist nur zum Teil valide und viele der im Vorfeld getesteten Parser hatten mit ihnen Schwierigkeiten.

Es war auch bei diesem Auftritt möglich die Seitenstruktur korrekt zu extrahieren. Da jedoch die `<title>`-Tags leer waren, konnten die Titel der Seiten nicht extrahiert werden. Auch Metadaten waren nicht gesetzt und konnten so ebenfalls nicht ermittelt werden.

Beim Internetauftritt unter der Domäne `http://www.eldoron.de` handelt es sich zwar um eine optisch sehr ansprechende Seite, doch dahinter steckt „schlechtes“ HTML. JTidy gibt bei der Verarbeitung dieser Seite pro Seite im Schnitt 120 Fehler im HTML-Code an. Die Seitenstruktur wird bei diesem Internetauftritt nicht über die Ordnerstruktur auf dem Server vorgegeben. Daher wird sie über die Verweishierarchie ermittelt.

Alle HTML-Seiten werden mit `Domäne + Dateiname.html` aufgerufen. Trotz der Fehler im HTML-Code und er fehlenden Ordnerstruktur auf dem Webserver konnte auch hier die Seitenstruktur korrekt extrahiert werden. Auch die Titel konnten mit Hilfe von JTidy ermittelt werden. Die Metadaten waren zwar gesetzt doch es wurden die falschen Attribute verwendet und so war es nicht möglich sie korrekt auszulesen.

6.2 Seiteninhaltsmigration

Für die Bewertung der Seiteninhaltsmigration und damit vor allem des Mapping-Moduls musste auf lokale Testdaten zurückgegriffen werden, da das Spider-Modul auf Markierungen im HTML-Code angewiesen ist um den zu migrierenden Inhalt von zu generierendem Inhalt zu unterscheiden. Hierfür wurden einzelne Webseiten aus dem Internet lokal abgelegt, markiert und in das System eingegeben.

Das Ergebnis dieser Versuche lässt sich wie folgt zusammenfassen: Eine Identifikation der Contentelemente ist nur bei validem HTML möglich, es treten zwar keine Fehler im Programm auf, aber die Ausgabe ist nicht verwertbar. Dasselbe gilt für schlecht strukturierten Code. Je sauberer der Code strukturiert wurde, umso exakter konnten die Contentatome und damit auch die Contentelemente identifiziert werden.

Besonders gut zuzuordnen waren einfach strukturierte Listen und Tabellen, die dem Aufbau in Kapitel 5.2 entsprachen. Allerdings liefern gerade CMSe Inhaltstabellen, die aus geschachtelten Tabellen bestehen und daher vom Mapping-Modul nicht als Inhaltstabellen erkannt werden. Tabellen, die als Designelement eingesetzt wurden, wurden nicht für Inhaltselemente gehalten.

Überschriften und Text wurden als solche erkannt, wenn sie nicht von weiteren Tags unterbrochen wurden. Wenn ein Text mit Hilfe von `<p>`-Tags strukturiert wurde, so konnte das System nicht erkennen, dass es sich eigentlich um einen einzigen, längeren Text handelte.

Ebenfalls nicht erkannt wurde, die Position eines Bildes relativ zum zugehörigen Text. So waren die Angaben für oben/links und unten/rechts jeweils gleich.

6.3 Import in das Zielsystem

Die Qualität des Imports der Seitenstruktur in das CMS Typo3 war von der Qualität des *page_structure.xml* abhängig. An dieser Stelle konnte, wie erwartet, ein befriedigendes Ergebnis erzielt werden.

Der Import der Seiteninhalte ist ebenfalls von der Qualität des jeweiligen *mapped_content.xml* abhängig. Die exemplarische Implementierung des Imports von Contentelementen vom Typ „Text“ war bei korrekten Vorgaben fehlerfrei.

6.4 Mögliche Erweiterungen

Da die Migration der Inhaltsseiten zufrieden stellend arbeitet, beschränken sich die Vorschläge für die möglichen Erweiterungen auf Seiteninhaltsmigration.

Als mögliche Ergänzung für die Seiteninhaltsmigration wäre eine komplexere Struktur der Contentelement wünschenswert. Mit Hilfe dieser zusätzlichen Attribute könnten neue Regeln zu Regelbasis hinzugefügt werden und eine noch genauere Zuordnung erfolgen.

Bisher wird noch nicht darauf geachtet, ob ein kurzer Text in einem `<h1-6>`-Tag steht. Es wird direkt davon ausgegangen, dass es sich bei einem kurzen Text um eine Überschrift handelt, aber auch lange Texte vor allem, wenn sie in `<h1-6>`-Tags vorkommen können Überschriften darstellen.

Textformatierungen werden bislang noch ignoriert und müssen von Hand nachgetragen werden, eine Erweiterung des Systems dahin gehend wäre wünschenswert.

In der aktuellen Version des Systems werden die Regeln der Zuordnung der Contentelemente noch fest in die Regelbasis codiert. Die Erstellung der Regeln könnte mit Hilfe eines Clustering-Verfahrens dynamisiert werden.

6.5 Zusammenfassung

Die Evaluierung des vorgestellten Verfahrens hat gezeigt, dass eine Seitenmigration auch bei schlecht strukturierten Seiten möglich ist. Die Seiteninhaltsmigration jedoch stark von der Qualität des zugrunde liegenden HTML-Codes abhängt. Eine noch stärkere Berücksichtigung des Kontextes der Contentatome könnte das Verfahren weiter verbessern.

Doch schon eine Automatisierung der Migration der Seiten bedeutet eine enorme Einsparung an Arbeitszeit und damit Kosten.

7. Fazit

Das entworfene System ist im Bezug auf die Seitenmigration sehr erfolgreich und kann industriell eingesetzt werden. Die Migration der Seiteninhalte ist zwar möglich, doch ist hier der Aufwand für die Nacharbeitung umfangreich. Eine Verbesserung der Berücksichtigung des Kontextes der einzelnen Contentelemente wäre wünschenswert. Eine Erweiterung und dynamische Erstellung der Regelbasis könnte das Verfahren weiter verbessern.

Die Fehler, die bei der Migration der Seiteninhalte auftreten, gehen zu einem Teil auf nicht oder schlecht strukturierten HTML-Code zurück. Eine in der Praxis besser umgesetzte Trennung zwischen Inhalt und Darstellung, beim Erstellen von HTML-Seiten, wäre für das Verfahren sehr hilfreich.

Schlecht strukturierter und fehlerhaft codierter HTML-Code sind nicht nur ein Problem für die Contentmigration, sondern für viele Applikationen die auf Daten im HTML-Code von Webseiten zugreifen möchten. Auch eine Trennung von Inhalt und Design ist nicht nur für die maschinelle Weiterverarbeitung von HTML-Dokumenten von Interesse, es erleichtert auch behinderten Menschen den Umgang mit dem WWW.

Daher sollten alle Entwickler von Webseiten darauf achten, dass sie eine saubere Trennung von Inhalt und Darstellung praktizieren und nur validen Code im Internet verbreiten.

A. Abkürzungen

API	:	**A**pplication **P**rogramming **I**nterface
CMS	:	**C**ontent **M**anagement **S**ystem
DOM	:	**D**ocument **O**bject **M**odel
DTD	:	**D**okument**t**yp**d**efinition
SGML	:	**S**tandard **G**eneralized **M**arkup **L**anguage
HTML	:	**P**ype**t**text **M**arkup **L**anguage
HTTP	:	**P**ype**t**text Transfer Protocol
PHP	:	Hypertext Preprocessor, früher: „**P**ersonal **H**ome **P**age **t**Tools“
URL	:	**U**niform **R**esource **L**ocator
URI	:	**U**niform **R**esource **I**dentifier
W3C	:	**W**orld **W**ide **W**eb **C**onsortium
WWW	:	**W**orld **W**ide **W**eb
XHTML	:	**E**xtensible **P**ype**t**text **M**arkup **L**anguage
XML	:	**E**xtensible **M**arkup **L**anguage

Tabelle A.1: Abkürzungen

Literaturverzeichnis

[Biew97] Benno Biewer. *Fuzzy-Methoden*. Springer-Verlag. 1997.

[Conn92] Dan Connolly. http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/MarkUp.html, 1992.

[FuLi04] Tao Fu und Menchi Liu. A Gateway From HTML to XML. 2004.

[Harr02] Elliotte Rusty Harrold. *Die XML Bibel*. mipt-Verlag. 2002.

[Heat02] Jeff Heaton. *Programming Spiders, Bots, and Aggregators in Java*. Sybnex. 2002.

[John02] Jeffrey D. Ullman John E. Hopcroft, Rajeev Motwani. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. ADDISON WESLEY LONGMAN. 2002.

[Lipp06] Wolfram-Manfred Lippe. *Soft-Computing*. Springer-Verlag. 2006.

[Mada] E. H. Madami. An experiment in linguistic synthesis with a fuzzy logic controller. *Int. Journal of Man-Mashines Studies 7*.

[Meye05] Robert Meyer. *Praxiswissen Typo3*. O'Reilly. 2005.

[Münz05] Stefan Münz. SELFHTML: Version 8.1.1 http://de.selfhtml.org/, 2005.

[Nied02] Jennifer Niederst. *HTML kurz&gut*. O'Reilly. 2002.

[Oliv01] Rik Zahradka Oliver Zschau, Dennis Traub. *Web Content Management - Websites professionell planen und betreiben*. Galilo Press. 2001.

[Phil05] Lauren Wood Philippe Le Hégaret, Ray Whitmer. Document Object Model (DOM) http://www.w3.org/DOM/, 2005.

[Ragg97] Dave Raggett. HTML 3.2 Reference Specification http://www.w3.org/TR/REC-html32, 1997.

[Tim 94] et al. Tim Bernes-Lee. The World-Wide Web. *Communications Of The ACM*, Band 37, 1994.

[Tolk03] Prof. Dr.-Ing. Robert Tolksdorf. *HTML & XHTML - die Sprache des Web*. dpunkt.verlag GmbH. 2003.

[Usam96] et al. Usama M. Fayyad. *Advances In Knowledge Discovery And Data Mining*. AAAI Press/ The MIT Press. 1996.

[XTMa05] Jean-Jacques Girardot Xavier Tannier und Mihaela Mathieu. Classifying XML Tags through 'Reading Contexts'. 2005.