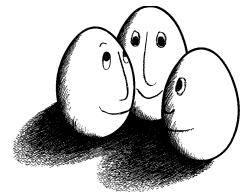


Diplomarbeit

Automatischer Aufbau einer Produkt-Datenbank mit Hilfe von Information Extraction

Tchavdar Marinov



Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz

Betreuer:
Prof. Dr. Katharina Morik
Dipl.-Inform. Felix Jungermann

18. November 2009

Inhaltsverzeichnis

1	Einleitung	2
2	Verwandte Arbeiten	4
3	Webshops	6
3.1	Gängige Onlineshopssysteme	6
3.2	Softwarearchitektur von Webshops	7
3.3	Seitenstruktur und Produktdarstellung in Webshops	9
3.4	Sonderfälle und Einschränkungen	12
3.4.1	Webshops ohne Kategoriebaum	12
3.4.2	Parameter auf der Produktseite	13
3.4.3	JavaScript und Ajax	14
3.4.4	Onlineshops mit Adobe Flash	15
4	Allgemeine Grundlagen	16
4.1	Metasprachen	16
4.1.1	Extensible Markup Language (XML)	17
4.1.2	Hypertext Markup Language (HTML)	18
4.2	Zugriff auf Metasprachen	19
4.2.1	Document Object Model	19
4.2.2	XPath	20
4.3	XML-Datenbanken	24
4.4	Information Retrieval	25
4.4.1	Architektur eines IR-Systems	25
4.4.2	Textoperationen	26
4.4.3	Index	29
4.4.4	IR-Modelle und Ranking	30
4.4.5	Retrieval Evaluation	35
4.4.6	Queries	36
4.5	Information Extraction	38
4.5.1	Informationsextrahierung aus unstrukturierten Dokumenten	38
4.5.2	Information Extraction in semi-strukturierten Webdokumenten	41
4.6	Information Extraction in Webshops	42
4.6.1	Struktur der Produktseiten	42
4.6.2	Klassifizierung von Webseiten	44
5	Eigene Implementierungen und Tools	49
5.1	Software-Architektur	49
5.2	Crawler	50

5.3	HTML Cleaner	51
5.4	XML-Datenbank	52
5.5	Lucene	55
5.5.1	Indexaufbau	56
5.5.2	Suchen mit Lucene	57
5.6	XPath Generator	59
5.6.1	XPath-Ausdruck für den Produktnamen	61
5.6.2	XPath-Ausdruck für die Produktbeschreibung	62
5.6.3	XPath-Ausdruck für den Produktpreis	63
5.6.4	Analyse der Produktseiten und XPath-Generierung	66
5.6.5	Generalisierung der XPath-Ausdrücke	67
5.7	Property Extraction	68
5.8	Datenbankimport	70
6	Experimente	73
6.1	Ziel und Ablauf der Experimente	73
6.2	Erste Experimente	75
6.2.1	Informationsextrahierung mit Webharvest	75
6.2.2	Erste Version eines Programms für Produktextrahierung	76
6.3	Datenaufbereitung	79
6.4	Manuelle Produktextrahierung	83
6.5	Automatische Produktextrahierung	85
6.5.1	Klassifizierung der Webseiten mit RapidMiner	86
6.5.2	Suche nach Produktseiten mit Lucene	88
6.5.3	Klassifizierung der Webseiten mit XPath	92
6.5.4	Datenextrahierung und Datenbankimport	96
7	Zusammenfassung und Ausblick	98

Abbildungsverzeichnis

3.1	Drei-Schichten-Architektur bei Onlineshops	7
3.2	Beispiel für eine Kategorienseite (Quelle: technikbilliger.de)	8
3.3	Beispiel für eine Produktseite (Quelle: redcoon.de)	10
3.4	HTML-Code für die Darstellung des Produktnamens	11
3.5	HTML-Code für die Darstellung von Preisen mit Bilddateien	12
3.6	Beispiel für einen Webshop mit Suchformular (Quelle: autoteilestore.com)	13
3.7	Vergleich zwischen den herkömmlichen Modellen (links) und dem Ajax-Modell (Quelle: sitepoint.com)	15
4.1	Entwicklung von SGML und verwandten Standards [FG99]	16
4.2	XML-Darstellung von Informationen (Quelle: w3c.de)	17
4.3	Repräsentation einer einfachen Html-Seite in DOM	19
4.4	Graphische Darstellung der möglichen Achsen in XPath (Quelle: [GW00]).	22
4.5	Ablaufplan eines Retrieval-Prozesses	26
4.6	Struktur und Aufbau von einem invertierten Index mit Hilfe einer Stoppwortliste; in der Positionsliste (Spalte <i>Document</i>) stehen die Positionen der Terme im Text (Quelle: developer.apple.com)	29
4.7	Graphische Darstellung der drei Terme in der Query q	31
4.8	Beispielstabelle für die Berechnung der tf-idf-Werte.	34
4.9	Graphische Repräsentation der Menge der richtig erkannten Dokumente (R_a)	36
4.10	Beispiel für Precision-Recall-Kurve.	37
4.11	Struktur des klassischen IE-Prozesses	39
4.12	Struktur einer Webseite für Restaurants, repräsentiert als OEM-Graph (Quelle: [Abi97])	42
4.13	Baumdarstellung einer Produktseite	44
5.1	Architektur der entwickelten Software	49
5.2	Struktur einer Xml-Datenbank für ein Webshop	53
5.3	Die GUI von BaseX	54
5.4	Struktur des XPath-Generators	60
5.5	Hashtabelle als Vokabular für unterschiedlichen Zahlenformate	65
5.6	Struktur der Produktdatenbank in MySQL	71
6.1	Ablauf der automatischen Produktextrahierung	74
6.2	Produktextrahierung in einem Webshop mit Web-Harvest	76
6.3	Ablauf der automatischen Generierung von XPath-Ausdrücke	77
6.4	Beispiel für Produktseite mit Kategorienavigation (links) und Produkttabelle (rechts). (Quelle: hoh.de)	78
6.5	Hierarchische Dokumentenstruktur.	82

6.6	Flache Dokumentenstruktur.	82
6.7	Manuelle Konstruierung von einem XPath-Ausdruck für den Produktname	84
6.8	Beispielsexperiment für die Klassifizierung der Seiten innerhalb eines Webshops	87
6.9	Auszug aus der Produkttabelle in MySQL	90
6.10	Auszug aus der Datenbank. Die ersten drei Spalten zeigen die Beispielsprodukt- daten, die letzten drei - die Daten aus der erkannten Produktseiten. Die falsch erkannten Produkten sind gelb markiert.	93
6.11	Beispiel für Produktseite in hoh.de, die Positionen der Produktattributen werden im Quellcode markiert.	94
6.12	Quellcode für die Ausführung des Datenbankkomports	97
6.13	Screenshot von MySQL Query Browser. Auszug aus der Produkttabelle.	97

1 Einleitung

Es liegt in der menschlichen Natur, möglichst viele zeitaufwendige, lästige und sich wiederholende Aufgaben zu vermeiden und nach Möglichkeit zu automatisieren. Mit der ständig wachsenden Anzahl und Vielfalt von Webseiten im Internet gehört das Sammeln von interessanten Informationen von verschiedenen Seiten definitiv zu einer dieser Aufgaben. In der Regel ist die gezielte Informationsbeschaffung für den Internetnutzer mit einer zeitaufwendigen Recherche verbunden, die sich dadurch kennzeichnet, dass sich der User durch unzählige Internetdokumente klicken muss, um eine vollständige Datensammlung zu generieren. Leistungsfähige Suchmaschinen sind für die Vereinfachung der Internetrecherche entwickelt worden. Sie liefern zwar relevante Dokumente, doch Auswertung und Analyse des Datenmaterials muss in der Regel vom User selbst übernommen werden. Die komfortabelste Lösung für den User wäre die Möglichkeit, eine direkte Abfrage zu senden und die Ergebnisse präzise und webseitenübergreifend wie von einer strukturierten Datenbank zu erhalten. Zum Beispiel „Finde alle 17 Zoll Notebooks, die unter 1000 Euro kosten und sofort lieferbar sind“.

Mit diesem Problem beschäftigt sich das Gebiet der „Information Extraction“ (IE). Die Kernaufgabe dieser Technik ist die automatische Extrahierung von relevanten Daten aus Text- bzw. Webdokumenten. Es werden aus einer vorgegebenen Menge an Webseiten nur die notwendigen Textfragmente entnommen, um anschließend eine vordefinierte Datenbank mit Daten zu füllen. Um so einen Prozess zu automatisieren, hat man mit dem Problem der fehlenden bzw. uneinheitlichen Struktur der Ausgangsmaterialien zu kämpfen. Schließlich sind die Webseiten für den realen Besucher gestaltet und nicht für die Nutzung durch Maschinen, die mit den rohen Daten im Seitenquellcode zurecht kommen müssen. Um das zu ermöglichen, wird in der Regel für jede Webseite ein sogenannter „Wrapper“ von Hand erstellt, der die relevanten Stellen im Dokument anhand bestimmter Regeln, wie Regular Expressions und XQuery, mit den Feldern in der zu füllende Datenbank verbindet. Dieser Ansatz hat den Nachteil, dass der Wrapper jedes Mal bei Änderungen in der Struktur des HTML-Codes angepasst werden muss. Außerdem ist es sehr zeitaufwendig, für alle zu untersuchenden Domains ein solches Mapping-Schema per Hand zu erstellen. Deswegen ist es von großem Interesse, eine Lösung zu entwickeln, die so einen Wrapper automatisch mit Hilfe der Techniken aus dem Bereich des Maschinellen Lernens für jede Webseite generiert. Die Idee ist, anhand einer Trainingsmenge, die man manuell für einige Dokumente zusammenstellt und als *Input* vorgibt, die neuen Transformationsregeln für den Wrapper automatisch zu erkennen. Anwendung findet die Information-Extraction-Technik in vielen wissenschaftlichen und kommerziellen Bereichen. Einige Beispiele dafür sind „News Tracking“, „Citation Databases“, „Opinion Databases“, Produktvergleichssysteme und andere. Die Entwicklung eines universellen intelligenten Agenten, der in allen Bereichen gleichzeitig erfolgreich einsetzbar wäre, ist vielleicht utopisch. Deswegen sollte man sich mit einer speziell angepassten Lösung für jedes Einsatzgebiet zufrieden geben.

Das Ziel dieser Diplomarbeit besteht darin, eine Lösung für die automatische Extrahierung von

Produkten und deren Eigenschaften wie Beschreibung und Preise direkt aus der Webseiten der Internetshops zu entwickeln. Das zu entwickelte Verfahren soll so robust und flexibel sein, dass es an einer Vielzahl an unbekanntem Webshops angewendet werden kann, ohne ein manuelles Eingreifen seitens der Softwareadministrator oder der Webshopbetreiber. Als Ausgangsbasis soll lediglich eine Trainingsmenge von Produktdaten zur Verfügung gestellt werden, als auch eine Liste mit Webshops und deren Internetadressen, die der zu entwickelten System dient, sie zu besuchen und auszulesen. Durch Analyse der gespeicherten Seiten und die Verwendung der Beispielmengen von Produktdaten, sollen mit Hilfe von Mustererkennung die Webseiten in einem neuen Shop in zwei Klassen eingeteilt werden. In der ersten Klasse kommen alle Seiten, auf denen jeweils nur ein Produkt detailliert präsentiert wird. In die zweite Klasse sollen alle restlichen Webseiten kommen, die für die Extrahierung der Produktinformationen nicht nützlich sind. Anhand der Webseiten aus der ersten Klasse sollen die einzelnen Produktattributen, wie Beschreibung und Preis, erkannt und entnommen werden. Schließlich wird eine webshopübergreifende Produktdatenbank aufgebaut und verwaltet, die alle Produktattributen von unterschiedlichen Verkäufern beinhaltet. Zusätzlich können die gesammelten Daten als Ausgangsbasis verwendet werden, um den Nutzern eine einzige, auf diese Datenbank basierende Internetseite anzubieten, wo die Produktinformationen in einem einheitlichen Format präsentiert werden, ggf. nach Preis sortierbar sind und somit einen schnellen Vergleich ermöglichen. Ein solches System könnte sowohl für private Internetnutzer von Interesse sein sowie der E-Commerce-Branche dienen, um Marktforschung und Wettbewerbsanalysen effektiver und einfacher durchzuführen.

Zunächst werden in Kapitel 2 ähnliche Verfahren und Systeme kurz beschrieben. Kapitel 3 gibt Einblick in die Architektur und Funktionsweise der Webshops-Systeme. Zusätzlich werden verschiedene Fälle, die von dem zu entwickelten Verfahren aus technische Gründe nicht berücksichtigt werden können. Kapitel 4 behandelt die theoretischen Grundlagen der Methoden, die in der vorliegenden Diplomarbeit zum Einsatz kommen. In Kapitel 5 wird die Architektur der entwickelten Software vorgestellt und erläutert. Die Ergebnisse und die Auswertung der durchgeführten Experimente werden anschließend in Kapitel 6 präsentiert.

2 Verwandte Arbeiten

Die Idee, Produkte im Internet automatisch zu vergleichen, ist nicht neu, und es gab in der Vergangenheit viele wissenschaftliche Ansätze zu diesem Thema. Einer der Pioniere in diesem Bereich ist Oren Etzioni, der in seinem Paper [DEW97] einen der ersten Ansätze für Extrahierung von Produktinformationen von Webshops verfolgt hat. Die Idee besteht darin, mit einem modifizierten Webcrawler das Produktsuchfeld auf der Seite automatisch zu entdecken und dann eine Suche nach bestimmten Produkten zu senden und anschließend die Produkteigenschaften aus der zurückgelieferten Artikelaufistung, die in einer tabellarischer Form vermutet wird, zu entnehmen. Dieser Ansatz ist erfolgreich, wenn es darum geht, die Eigenschaften bereits bekannter Produkte zu sammeln. Neue Produkte werden damit nicht erkannt und logischerweise nicht in die Produktdatenbank aufgenommen.

Mittlerweile existieren unzählige *Preisvergleichsportale* im Internet, die den Besuchern eine Möglichkeit bieten, nach Produkte in eine Reihe von Internetshops gleichzeitig zu suchen und deren Preise zu vergleichen. Die bekanntesten davon sind *Froogle*¹, *Shopping.com*² so wie *Günstiger.de*³ und *Billiger.de*⁴ für den Deutschen Internetraum. Darüberhinaus existieren viele Portale, die sich in einem Bereich spezialisiert haben, wie z.B. Elektronik, Finanzleistungen, Reisen und Flüge, und bieten erweiterte Suchmöglichkeiten für die Produktsuche in diesem Bereich. Die meisten Portale erfordern von den Betreibern der Internetshops eine Anmeldung, damit sie ihre Produkte in der Portalsdatenbank aufnehmen. In den meisten Fällen ist das mit gewissen Kosten für den Händler verbunden, wie bei *Billiger.de*. Die Aufnahme des Produktangebots in die Portaldatenbank geschieht anhand spezieller Schnittstellen, die seitens der Preisvergleichsportale definiert sind. Das ist einerseits vorteilig, denn die Daten werden in einer strukturierten Form (CSV,XML, etc.) übertragen, was das *Matching* der Produkte vereinfacht. Der Nachteil dabei ist, dass die Händler sich an jedem gleichartigen Portal anpassen müssen. Für die Umsetzung der Übertragung der Produktdaten braucht der Webshop-Betreiber auch gewisse Fachkenntnisse, die besonders kleinere Onlineverkäufer nicht immer besitzen. Es besteht zusätzlich die Möglichkeit die Produktdaten mit Hilfe einer Tabelle zu übertragen, die in bestimmten Abständen von den Händler an die Preisvergleichssysteme zur Verfügung gestellt wird. Hier ist der Nachteil, dass dies nicht immer automatisch geschieht, was zu veralteten Produktangebote führen kann. *Günstiger.de* bietet als einziger Preisvergleichsdienst die Möglichkeit, bestimmte Produktangebote zusätzlich auch manuell in ihrer Datenbank aufzunehmen.

Die Aufnahme der Produktdaten bei *Froogle* funktioniert etwas anders als die bereits beschriebenen Verfahren und basiert auf dem *GoogleBot*, der Webcrawler von *Google* [Col04]. Auf der *Froogle* Webseite wird ein Suchfeld angeboten, wo die Besucher nach einem bestimmten Pro-

¹Google Product Search (Froogle) - <http://www.google.com/products>

²Shopping.com - <http://www.shopping.com>

³Günstiger.de - <http://www.guenstiger.de/>

⁴Billiger.de - <http://www.billiger.de/>

dukt suchen können. Damit die Webshops ihre Daten in der *Froogle-Suche* präsentieren, ist eine Anmeldung bei Google notwendig. Anschließend müssen die Händler eine Produktliste (*Froogle Data Feed*) zur Verfügung stellen, die alle wichtigen Produktdaten in einer strukturierten Form beinhaltet. Das sind unter anderem die Produktnamen, die Beschreibungen, Preise und Bilder. Zusätzlich muss auch die Internetadresse (URL) der Produktseite in der Liste angegeben werden. Das besondere bei Froogle ist, dass nach dem Übertragung der Produktdaten, werden alle Produktseiten einmal von dem Googlebot besucht und indexiert, um einen *Page-Rank* für jede Seite zu berechnen. Dabei werden die Inhalte der Seite mit den in der zur Verfügung gestellten Produktliste verglichen. Bei diesem Schritt werden die Webseiten zusammen mit der Quellcode-Struktur analysiert und die einzelnen Produktattributen wie Beschreibung und Preis aus der Seite extrahiert. Damit das reibungslos funktioniert, müssen die Seiten in den Internetshops zuerst entsprechend optimiert, indem die einzelnen Attribute dem Googlebot kenntlich gemacht werden. Ansonsten kann die berechnete Relevanz und entsprechend der Rang der Produktseiten betroffen werden. Als Fazit entsteht bei der Informationsintegration bei Froogle für die Webshopbetreiber deutlich mehr Arbeit. Zum einen müssen die Produktdaten übertragen werden, zum anderen sind auch die Webseiten zusätzlich anzupassen.

In der Diplomarbeit vorgestellte Verfahren hat die Aufgabe die Seiten der Internetshops zu analysieren und ihre Produktdaten völlig automatisch zu extrahieren. Dabei können die Daten in einer strukturierten Form gebracht und anschließend in einer Produkttabelle eingefügt werden, die als Ausgangsbasis für einen Preisvergleich dienen können. Der Hauptvorteil dieses Ansatzes ist, dass die Webshops dabei nichts machen müssen, damit ihre Produktdaten in der Datenbank aufgenommen werden. Zusätzlich können die Daten einfacher und aktuell gehalten werden, denn es keine manuelle Zwischenschritte vorgesehen sind, die eventuell den Prozess der Datenextrahierung verzögern können.

3 Webshops

Von Jahr zu Jahr gewinnt der Internethandel immer mehr an Bedeutung und Popularität. Laut dem Bundesverband des Deutschen Versandhandels (BDV) betrug die Anzahl der Onlinekäufer im Jahr 2008 mehr als 31 Mio. [Ver06], Tendenz steigend. Allein in Deutschland beläuft sich der Umsatz von über das Internet verkauften Waren auf 13,4 Mrd. Euro. Seit dem Ende der neunziger Jahren ist die Anzahl an Onlineshops enorm gestiegen. Die Gründe sind vor allem der einfachere Einstieg in den Internethandel und das enorme Kundenpotenzial im World Wide Web.

3.1 Gängige Onlineshopssysteme

Einen Onlineshop zu eröffnen ist nicht schwer und relativ günstig. Alles, was man braucht, ist ein Server (evtl. mehrere, je nach Popularität) und die eigentliche Webshop-Anwendung, eine webbasierte Software, die auf diesem Server ausgeführt wird. Die meisten Verkäufer entscheiden sich aus Zeit- und Kostengründen für fertige Lösungen. Hierzu gibt es eine grosse Auswahl sowohl an kommerziellen als auch an Open-Source-Produkten. Die bekanntesten sind:

- osCommerce¹ - eine der bekanntesten Open-Source-Software für Webshops
- xt:Commerce² - eine kommerzielle Weiterentwicklung von osCommerce
- OXID eshop³ - eine sehr flexible und stabile Webshop-Anwendung; es gibt sowohl eine kommerzielle (*Enterprise Edition*) als auch eine Open-Source-(*Community*-)Version.
- Smartstore⁴ - einfaches und benutzerfreundliches kommerzielles Webshopssystem mit vielen Optionen, allerdings relativ teuer
- Epages⁵ - kommerzielles Webshopssystem, das zusätzlich einen Mietservice anbietet; die Software wird oft in Kooperation mit den Hostinggesellschaften angeboten
- FWP Shop⁶ - eine Open-Source-Anwendung, die besonders an den europäischen Markt angepasst ist
- Magento⁷ - eine weitere kommerzielle Webshop-Software, die eine Community-(Open-Source-)Version anbietet.

¹osCommerce - <http://www.oscommerce.com/>

²xt:Commerce - <http://www.xt-commerce.com/>

³OXID eShop - <http://www.oxid-esales.com/>

⁴Smartstore - <http://www.smartstore.com/>

⁵Epages - <http://www.epages.com/>

⁶FWP Shop - <http://www.fwpshop.org/>

⁷Magento Commerce - <http://www.magentocommerce.com/>

Zusätzlich finden sich hunderte von Open-Source-Webshop-Projekten bei SourceForge⁸. Der Nachteil gegenüber Individualsoftware ist die geringere Flexibilität, was besonders bei individuellen Anpassungen auffällt. Besonders größere und bekannte Shops entwickeln ihr eigenes Shopsystem selbst und gehen so zu 100% auf die speziellen Anforderungen ein.

3.2 Softwarearchitektur von Webshops

Wie die meisten Webapplikationen ist ein Webshop mittels drei Schichten strukturiert [Rät98]. Im Hintergrund steht immer eine relationale Datenbank, in der auch die Produktdaten verwaltet werden, die sogenannte Datenschicht. Im Vordergrund ist die Präsentationsschicht oder die GUI (Graphical User Interface), die bei den Webanwendungen durch dynamisch generierte HTML-Seiten repräsentiert wird. In der Mitte steuert die Logikschicht die Abläufe im Programm, verarbeitet die Benutzereingabe, liefert Daten von der Datenbank an die GUI und führt bestimmte Befehle aus. Auf Figur 3.1 wird grob die Struktur eines Systems mit der Drei-Schichten-Architektur dargestellt.

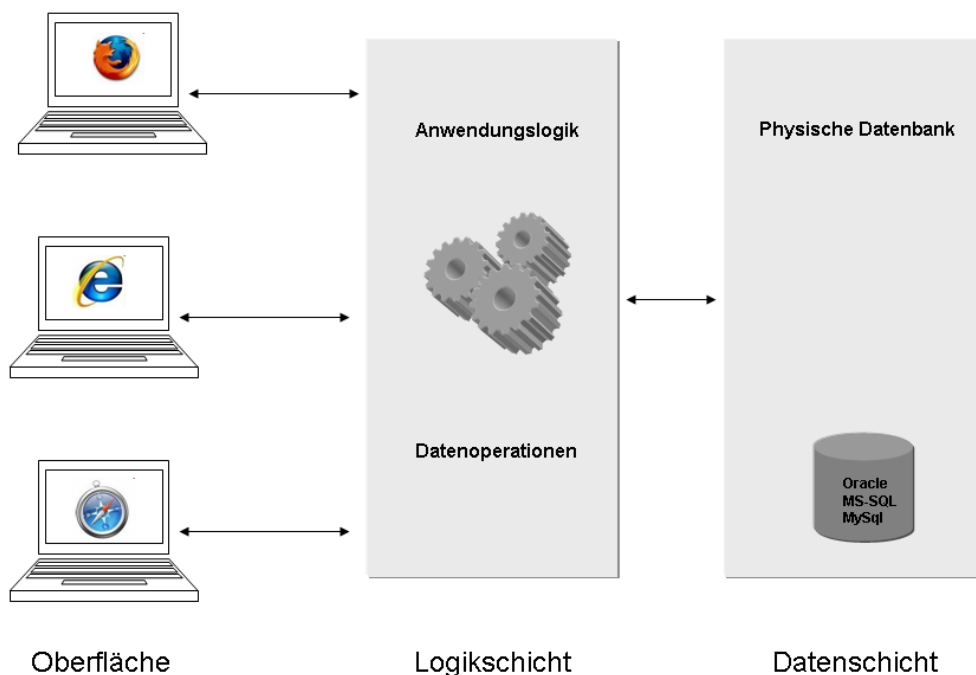


Abbildung 3.1: Drei-Schichten-Architektur bei Onlineshops

Ein Webshop besteht aus eine Vielzahl an dynamisch generierten Seiten, die sich hauptsächlich in folgende Kategorien klassifizieren lassen:

⁸SourceForge - <http://sourceforge.net>

- Startseite - bietet einen Überblick über alle Kategorien und Funktionen; zusätzlich können einige ausgewählte Produkte angezeigt werden
- Kategorieauflistung - präsentiert alle Produkte einer ausgewählten Kategorie in einer tabellarischen Form (siehe Figur 3.2)
- Suchergebnisliste - stellt die Ergebnisse einer Suche in tabellarischer Form ähnlich wie bei der Kategorieauflistung dar
- Produktseite - zeigt ein Produkt samt der kompletten Spezifikationen und Preise
- Warenkorb - zeigt den Inhalt des Warenkorbs, also die bereits ausgewählten Produkte

Startseite » Katalog » Monitore » Monitor TFT 19 Zoll

VERTRAUENSWÜRDIGKEIT UND SICHERHEIT

TECHNIBILLIGER.DE ist zertifiziert durch das Europäische Online-Shopping-Gütesiegel

Monitor TFT 19 Zoll

ALLE Acer ASUS BenQ Elzo Fujitsu HANNSPREE Lenovo LG Electronics NEC Philips Samsung WORTMANN AG

Sortieren nach ...

Zeige 1 bis 20 (von insgesamt 39 Artikeln) Seiten: 1 2 [nächste >>]

<p>1905W9FS/00 LCD Monitor</p> <p>19" LCD/ 48 cm/ 16:9/ analog/ silber/ 10.000 :1/ 300 cd/qm/ horiz.: 176 °/ vert.: 170 °/ 5 ms/ 1.440 x 900/ Eigenschaften: neigbar/ TC003/ Pixelfehlerklasse: II/ VESA Bohrung: 100 x 100 mm/ DVI-D, ...</p>	<p>101,15 EUR (inkl. 19 % MwSt. zzgl. Versandkosten)</p> <p>Lieferzeit: sofort lieferbar</p> <p>MERKEN KAUFEN</p>
<p> ACER V193WAb 19Zoll Wide TFT analog 10.000:1 300cd/m² 5ms schwarz matt</p> <p>Monitor V193WAb / 19" wide / LCD / D-Sub / 1440 x 900 / 10.000:1 / 300 cd/m² / 5ms / schwarz matt / 3Jahre / 4,3kg</p>	<p>95,33 EUR (inkl. 19 % MwSt. zzgl. Versandkosten)</p> <p>Lieferzeit: unbekannt</p> <p>MERKEN KAUFEN</p>
<p> ACER V193Wbb 19Zoll Wide TFT analog 10.000:1 5ms 250cd/m²</p> <p>Acer V193Wbb - LCD-Display - TFT - 19" - Breitbildformat - 1440 x 900 - 300 cd/m² - 10000:1 (dynamisch) - 5 ms - 0.284 mm - VGA - Schwarz</p>	<p>106,90 EUR (inkl. 19 % MwSt. zzgl. Versandkosten)</p> <p>Lieferzeit: sofort lieferbar</p> <p>MERKEN KAUFEN</p>
<p> BENQ E910 19Zoll TFT analog+digital 800:1 250cd/m² 5ms Speaker schwarz</p> <p>19" LCD/ 48 cm/ 5:4/ schwarz/ 2.500 :1/ 250 cd/qm/ horiz.: 160 °/ vert.: 160 °/ 5 ms/ 1.280 x 1.024/ TC003/ Pixelfehlerklasse: II/ VESA Bohrung: 100 x 100 mm/ DVI-D, D-Sub 15 pol./ 3,67 kg netto/ ...</p>	<p>133,80 EUR (inkl. 19 % MwSt. zzgl. Versandkosten)</p> <p>Lieferzeit: unbekannt</p> <p>MERKEN KAUFEN</p>
<p> BENQ E910T 19Zoll TFT analog+digital 800:1 250cd/m² 5ms Speaker silber</p> <p>19" LCD/ 48 cm/ 5:4/ silber/ 2.500 :1/ 250 cd/qm/ horiz.: 160 °/ vert.: 160 °/ 5 ms/ 1.280 x 1.024/ TC003/ Pixelfehlerklasse: II/ VESA Bohrung: 100 x 100 mm/ D-Sub 15 pol., DVI-D, RGB Analog/ 3,86 ...</p>	<p>158,25 EUR (inkl. 19 % MwSt. zzgl. Versandkosten)</p> <p>Lieferzeit: unbekannt</p> <p>MERKEN KAUFEN</p>
<p> FSC Scenicview P19-3 19Zoll TFT analog+digital 1500:1 300cd/m² 178/178</p> <p>Lichtsensiv/ 19" LCD/ 48 cm/ digital/analog/ PVA Panel/ grau/schwarz/ 1.500 :1/ 300 cd/qm/ horiz.: 178 °/ vert.: 178 °/ 8 ms/ 1.280 x 1.024/ Eigenschaften: höhenverstellbar, neigbar, Pivot drehbar ...</p>	<p>364,30 EUR (inkl. 19 % MwSt. zzgl. Versandkosten)</p> <p>Lieferzeit: ca. 3 bis 7 Tage</p> <p>MERKEN KAUFEN</p>
<p> HANNS-G HG191AP 19Zoll wide TFT analog WXGA+ 5ms 300cd/m² 700:1 VGA sp</p> <p>HANNS.G HG191AP - LCD-Display - TFT - 19" - Breitbildformat - 1440 x 900 / 75 Hz - 300 cd/m² - 700:1 - 5 ms - 0.2835 mm - VGA - Lautsprecher</p>	<p>97,19 EUR (inkl. 19 % MwSt. zzgl. Versandkosten)</p> <p>Lieferzeit: unbekannt</p> <p>MERKEN KAUFEN</p>

Abbildung 3.2: Beispiel für eine Kategorienseite (Quelle: technikbilliger.de)

Daneben gibt es auch einige statische Seiten, wie *AGBs (Allgemeine Geschäftsbedingungen), Impressum, Kontakt, Versandbedingungen* usw., die aber keine Produktinformationen beinhalten

und demzufolge für unsere Aufgabe nicht von Interesse sind. Unser Hauptziel ist es, Produkte in einem Onlinekatalog zu erkennen und mit sämtlichen Details und Eigenschaften zu extrahieren und damit eine Produktdatenbank aufzubauen. Die Seiten, die für die Informationsgewinnung relevant sind, sind genau die Produktseiten, wo jedes Produkt einzeln auf einer Seite samt Beschreibung und Preis präsentiert wird. Für die Generierung der Produktseiten werden die Daten aus der Webshop-Datenbank entnommen und in einem einheitlichen Format dargestellt. Normalerweise existiert eine Vorlage, die für jede Produktseite verwendet wird, so dass auf diesen Seiten sich nur bestimmte Inhalte ändern, wie zum Beispiel der Preis und die Produktbezeichnung. Entsprechend wird davon ausgegangen, dass bestimmte sich wiederholende Muster in der Seitenstruktur erkannt und so die Seiten erfolgreich klassifiziert und die Daten extrahiert werden können.

Die Onlineshops nutzen wie alle Webanwendungen *Cookies*, um die Besucher zu identifizieren [Kri01]. Im Allgemeinen sendet der Webserver mit Hilfe der Cookies bestimmte Informationen an die Benutzer, die er dann abfragen und eventuell ändern kann. Die Cookies werden vom Browser verwaltet und können als Objekte angesehen werden, die folgende Informationen beinhalten:

- Value - ein alphanumerischer Wert, der eine maximale Größe nicht überschreiten darf
- Name - ein eindeutiger Name, anhand dessen auf den Wert zugegriffen wird
- Expire-Time - definiert, wie lange das Cookie im Browser gültig ist
- Domain - die Domainadresse des Webserver; für eine Serveranwendung sind nur die Cookies sichtbar, die auch von dem gleichen Domainnamen gesetzt sind

Damit die meisten Onlineshops funktionieren, wird vorausgesetzt, dass der Browser des Benutzers Cookies akzeptiert. Der Warenkorb wird so implementiert, dass er sich die vom Benutzer ausgewählten Produkte merkt, indem für jedes Produkt eine Cookie zum Browser geschickt wird. Es kann zum Beispiel die Produktnummer und die Anzahl der ausgewählten Produkte beinhalten. So wird eine bessere Benutzerfreundlichkeit erreicht, indem alle vom Benutzer dem Warenkorb hinzugefügten Artikel gespeichert und in einem Schritt bezahlt beziehungsweise bestellt werden können. Ohne die Möglichkeit, diese Information zu speichern, müsste jedes Produkt einzeln bearbeitet werden. Einige Webshops nutzen die Cookies, um Informationen über einen Benutzer zu speichern und so die Seiten für ihn zu individualisieren. Zum Beispiel können die letzten angesehenen Produkte oder verwendeten Artikel angezeigt werden. Damit die Produktdaten aus den Webshops maschinell erfolgreich extrahiert werden können, muss eine Unterstützung für die Cookies in diesem Projekt berücksichtigt werden.

3.3 Seitenstruktur und Produktdarstellung in Webshops

Es existieren Ansätze, die automatisch und erfolgreich Daten aus der Webseite eines Webshops entnehmen. Sie basieren darauf, dass die Webdokumente eine wiederkehrende Quellcode-Struktur besitzen und bestimmte Daten eine konstante Position haben, die als Muster für die Erkennung weiterer Daten dienen kann. Ein Verfahren zu entwickeln, bei dem Informationen aus einer unbestimmten Menge unterschiedlicher Webshops entnommen werden, ist dagegen aus den folgenden Gründen keine triviale Aufgabe. Die Webseiten unterschiedlicher Onlineshops fallen optisch meist sehr unterschiedlich aus. Das Layout, das Design und die Farben werden absichtlich von den

Webshopbetreiber individuell gestaltet, damit sie sich von den anderen Seiten im Internet abheben und so einen hohen Wiedererkennungseffekt erreichen. Auch ist der Quellcode der Webseiten unterschiedlich. Eine Ausnahme bilden nur die Fertiglösungen, die auf Grund der gleichen Serveranwendungen einen gleich strukturierten Quellcode besitzen.

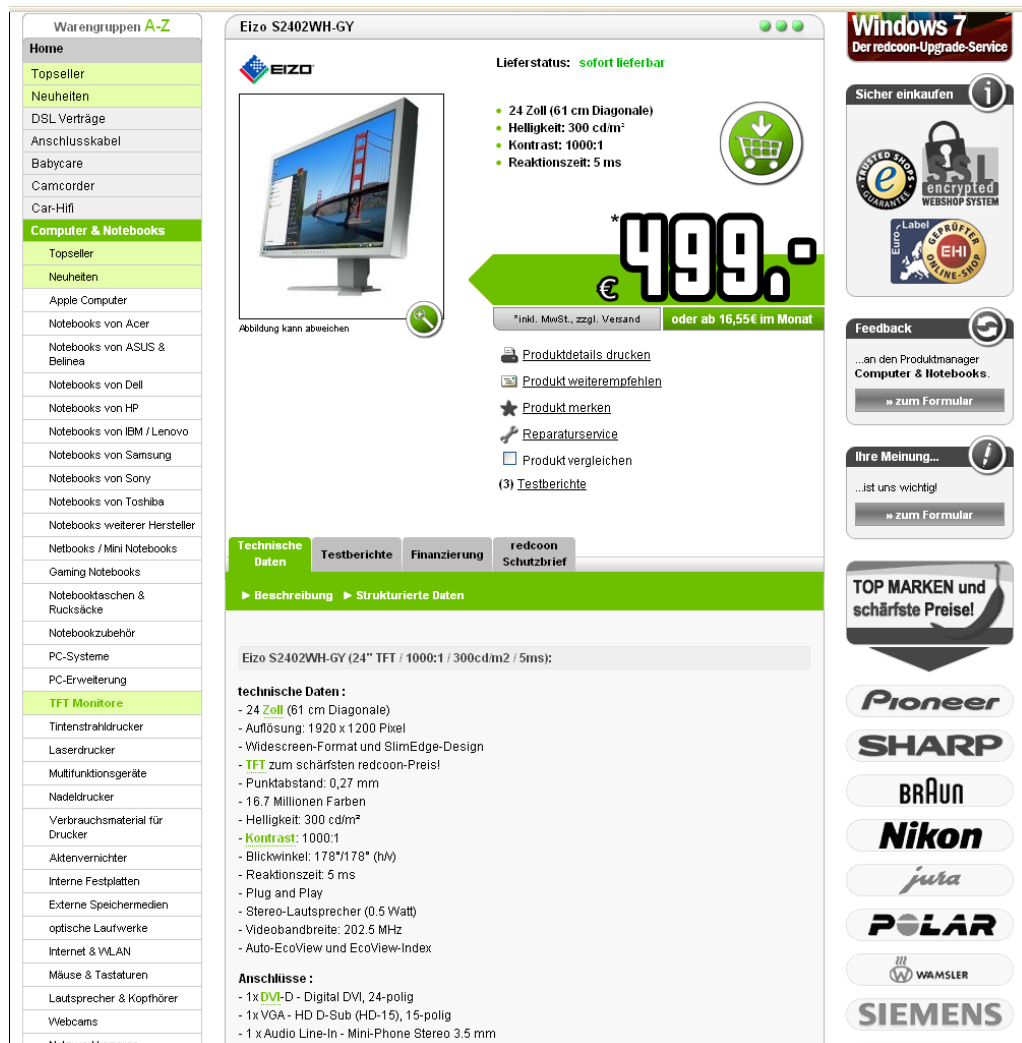


Abbildung 3.3: Beispiel für eine Produktseite (Quelle: redcoon.de)

Denoch existieren Regeln und Standards, an die sich die Webshopdesigner bei der Gestaltung meistens halten. Die Onlineshops besitzen eine einheitliche Seitenstruktur und Navigation, um den Besuchern eine einfache und schnelle Orientierung zu ermöglichen. Zum Beispiel werden die Navigationsmodule mit allen Produktkategorien immer im linken Bereich der Seite positioniert, die in einer hierarchischen Baumstruktur dargestellt werden. Beim Auswählen einer Kategorie werden die Produkte in einer tabellarischen Form aufgelistet, wo die wichtigsten Produktinformationen, wie Produktname, Bild und Preis, auf einen Blick abgelesen werden können. Die Suchfunktion wird meistens in der Kopfzeile der Seite sichtbar platziert. Das Warenkorbmodul ist normalerweise

se rechts oben auf der Seite zu finden und beinhaltet Informationen über die Preise der bereits ausgewählten Produkte. Auch die Navigation und die Verlinkung innerhalb der Webshops unterliegen gewissen Standards. Im Allgemeinen bieten die meisten Anwendungen unterschiedliche Wege, um zum gesuchten Produkt zu gelangen. Die Besucher können über das Suchformular eine Suche ausführen, wenn sie die genaue Produktbezeichnung kennen. Alternativ können sie die hierarchische Kategorienavigation benutzen, wo die Produkte entsprechend ihrer Kategorie klassifiziert sind. In einigen Webshops werden bestimmte Produkte nach dem Zufallsprinzip als Sonderangebote angezeigt oder verwandte Artikel auf unterschiedlichen Seiten präsentiert. Selbst die Produktseite, auf der detaillierte Informationen über ein Produkt präsentiert werden, besitzt in den unterschiedlichen Internetshops den gleichen Aufbau. Auf der Figur 3.3 wird ein Screenshot von einer Produktseite angezeigt. Die wichtigsten Produktinformationen, die auf der Seite auftauchen, sind: der Produktname bzw. die Modellbezeichnung, ein oder mehrere Produktfotos, der Preis, eine detaillierte Beschreibung und die Spezifikation sowie ein „Kaufen“- bzw. „Zum Warenkorb“-Button. Darüber hinaus können auch weitere Eigenschaften vertreten sein, wie Lieferbarkeit und Versandkosten, Finanzierungsmöglichkeiten, Userbewertungen und andere Informationen, die für unser Projekt nicht von Interesse sind.

- Der Produktname ist immer am Anfang der Seite platziert und besteht meistens aus wenigen Wörtern. Auf der Beispielseite ist das Attribut in der Kopfzeile der Produktkachel zu finden (Eizo S2402WH-GY). In einigen Shops ist der Text sogar farblich vorgehoben. Im HTML-Quellcode werden für solche Elemente normalerweise die sogenannten „Heading“-Tags verwendet. Zum einen wird so der Text extra hervorgehoben, zum anderen wird die Relevanz des Textes bei den Suchmaschinen verbessert. Aus diesem Grund greifen viele Webmaster gerne auf diese Technik zurück. Das kann in unserem Projekt bei der Analyse der Produktseiten ausgenutzt werden und wird in Unterkapitel 4.6.1 detailliert beschrieben. In Figur 3.4 wird ein Auszug aus dem Quellcode der Webseite aus dem Beispiel in Figur 3.3 wiedergegeben.

```
<div class="list-box">
  <div class="detail-box-header">
    <div class="left">
    <div class="right">
    <div class="available-button right">
    <div class="desc">
      <div>
        <h1>Eizo S2402WH-GY</h1>
      </div>
    </div>
  </div>
  <div class="detail-box">
  <div class="list-box-content">
  <div class="list-box-footer">
</div>
```

Abbildung 3.4: HTML-Code für die Darstellung des Produktnamens

- Das Produktfoto wird normalerweise auf der linken Seite des Produktkacheln eingebunden. Oft gibt es eine Vergrößerungsoption.
- Der Produktpreis ist neben der Produktbezeichnung die wichtigste Eigenschaft, die für die Informationsgewinnung von Interesse ist. Meistens ist er zentral positioniert und für die Benutzer gut zu erkennen. Für eine Maschine stellt dagegen die genaue Lokalisierung auf der Produktseite eine größere Herausforderung dar. Es gibt unzählige Darstellungsarten und

Zahlen- bzw. Währungsformate, was die Aufgabe so kompliziert macht. Im günstigsten Fall ist der Preis im Klartext angegeben und beinhaltet ein Währungssymbol, zum Beispiel:

1499.00 EUR	1,499.00 EUR	€1499	€1.499,-	1.499,00
-------------	--------------	-------	----------	----------

Leider existieren Onlineshops, bei denen der Produktpreis graphisch mit Hilfe von Bilddateien dargestellt wird. Der Grund kann unterschiedlich sein, unter anderem eine bessere Hervorhebung, mehr Freiheit beim Layout und Design, aber auch, die Arbeit von Preisvergleichssystemen zu erschweren. Die gängigste Methode, Preise mit Bildern zu präsentieren, ist, für jede Zahl eine separate Graphik im Sourcecode einzubinden. Auf Figur 3.5 wird anschaulich an einem Beispiel angezeigt, wie der Preis mit Bilddateien dargestellt wird.



Abbildung 3.5: HTML-Code für die Darstellung von Preisen mit Bilddateien

Vom Layout her wird der Preis typischerweise in der Nähe der Produktbezeichnung platziert. Unmittelbar neben dem Preis ist ein Button oder ein Link zu finden, mit die Besucher den Artikel in den Warenkorb aufnehmen können. Aus technischer Sicht wird dieser Vorgang durch ein HTML-Formular und einen Submit-Button realisiert. Unter dem Preis werden zusätzlich Informationen über die Lieferbarkeit und die Versandkosten angezeigt. Besonders größere Shops bieten Finanzierungsoptionen und zeigen zusätzlich zum Produktpreis eine monatliche Finanzierungsrate, die bei der automatischen Informationsgewinnung zusätzlich für Verwirrung sorgen kann.

- Die Beschreibung ist in der unteren Hälfte der Produktseite positioniert. Sie besteht in der Regel aus einem oder mehreren Absätzen oder aus einer Tabelle, in der gegebenenfalls die Produktspezifikationen aufgelistet sind.

3.4 Sonderfälle und Einschränkungen

In diesem Projekt haben wir uns auf die gängigen Webshopsysteme konzentriert. Ein Verfahren zu entwickeln, das genauso gut die Daten aus allen Internetshops entnimmt, ist utopisch. Es existieren Onlineshops, deren Inhalte besonders schwierig einzulesen sind. Im Gegensatz zur konventionellen Variante, bei der sich alle Seiten maschinell mit einer Art Crawler, der alle Links verfolgt, besuchen lassen, ist das in einigen Shops nicht ohne weiteres möglich. Im Folgenden werden alle bekannten Fälle aufgeführt, die von der entwickelten Software nicht unterstützt werden.

3.4.1 Webshops ohne Kategoriebaum

Es existieren Webshops, die kein Kategoriemenü bieten, sondern nur ein Suchformular. In diesem Fall muss der User eine Menge von Kriterien eingeben, bevor die Produktliste ausgegeben

wird. Damit alle Produkte ausgegeben werden können, muss das Suchformular mit allen möglichen Kombinationen ausgefüllt und entsprechend ausgeführt werden. Ein Beispiel dafür sind Onlineshops für Ersatzteile, wo zuerst die genauen Angaben für das Auto gemacht werden müssen, bevor die Produktkategorien angezeigt werden. Bei Reifen sieht es genauso aus, dort werden zum Beispiel als Erstes der Reifentyp (Winter/Sommer) und die Reifendimensionen abgefragt (Fig. 3.6). Diese Operationen stellen eine schwierige Aufgabe für den eingesetzten Crawler dar und erfordern eine speziell angepasste Vorgehensweise für jeden Onlineshop.

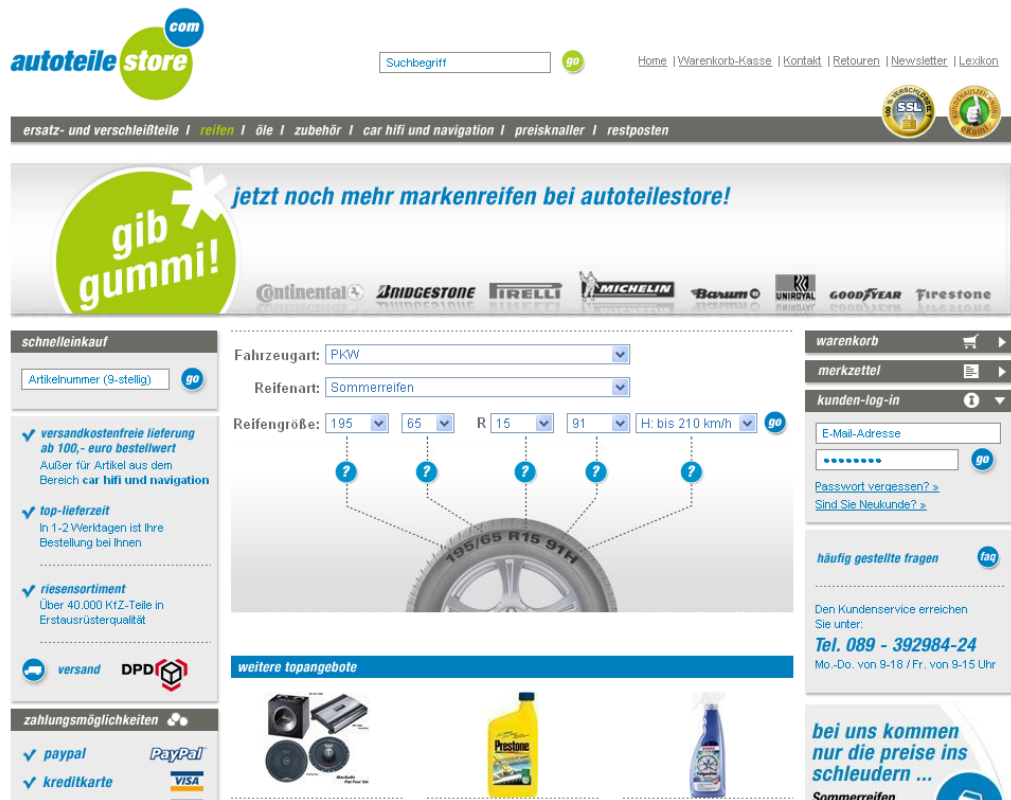


Abbildung 3.6: Beispiel für einen Webshop mit Suchformular (Quelle: autoteilestore.com)

3.4.2 Parameter auf der Produktseite

Eine weitere Schwierigkeit für die Crawler stellt das Auswählen von Parametern und Attributen auf der Produktseite dar. Eigenschaften wie Farbe, Größe, Kapazität oder Extras sind über spezielle Eingabefelder einzugeben. So lassen sich verschiedene Variationen eines Produkts auf einer Seite darstellen, wobei der Preis nicht immer gleich sein muss und unter Umständen von den gewählten Optionen abhängt. Ein Beispiel dafür ist die Produktseite von Speichermedien - je nach Kapazität fällt der Preis unterschiedlich aus.

3.4.3 JavaScript und Ajax

AJAX ist eine relativ junge Technologie für webbasierte Anwendungen. Sie baut auf Java Script auf und stellt ein Konzept dar, das eine asynchrone Kommunikation zwischen dem Server und dem Client (Browser) ermöglicht [Eer06]. Der Vorteil liegt hauptsächlich darin, dass bei Benutzerinteraktionen die Seite nicht komplett neugeladen wird, sondern nur relevante Teilbereiche bei Bedarf nachgeladen werden. Meistens wird diese Technologie für geschlossene Systeme verwendet, wo der Benutzer sich zuerst registrieren beziehungsweise einloggen muss. Solche Webanwendungen sind zum Beispiel soziale Netzwerke wie *Facebook* oder webbasierte E-Mail-Klient-Programme wie *Googlemail* oder *YahooMail*. So lassen sich Webapplikationen ähnlich wie Desktop-Anwendungen entwickeln, was die Mensch-Maschine-Interaktion verbessert und die Ladezeiten optimiert.

Es ist noch nicht weit verbreitet, dass auch Internetshops komplett mit dieser Technologie aufgebaut werden, allerdings finden sich bereits einige Trends in diese Richtung. Hierbei wird viel Wert auf die Spezialeffekte gelegt, und es werden Features für die Besucher angeboten, die mit klassischen Webapplikationsmodellen nicht möglich sind (Fig. 3.7). Mit einer asynchronen Kommunikation mit dem Server dagegen lassen sich Funktionen wie *Drag & Drop*, *Live-Search* oder Popup-Fenster mit dynamischen Informationen implementieren, ohne dass die Seite neu geladen wird. Im Gegensatz zum klassischen Webmodell, bei dem die komplette Seite in einem statischen HTML-Format an den Browser übermittelt wird, liefert der Server bei der Ajax-Technologie nur bestimmte Informationen im Rohformat (zum Beispiel XML oder JSON⁹), die mit Hilfe von JavaScript-Funktionen verarbeitet und dem Benutzer präsentiert werden. So sind die Seitenstruktur und das *Document Object Model* (DOM) nicht sofort verfügbar. Es ist die Aufgabe des Browsers, den JavaScript-Code zu interpretieren und anhand dessen das DOM-Objekt zu generieren. Mehr zum Thema XML und DOM ist in Kapitel 4 zu finden.

Ein Problem mit den Ajax-basierten Websites ist, dass deren Inhalte nicht komplett von den Webcrawlern bzw. Suchmaschinen indexiert werden können. Ein Grund dafür sind die fehlenden Href-Tags auf der Ajax-Seite, die von den Crawlern für das Navigieren zwischen den Webseiten benutzt werden. Bei den Webdokumenten, die mit der Ajax-Technologie aufgebaut sind, werden oft anstatt der konventionellen Href-Tags spezielle Elemente-Events benutzt, die bestimmte JavaScript-Funktionen aufrufen. Ein anderes Problem ist, dass die vom Webserver gelieferte HTML-Seite initial an sich keine Struktur und keine Daten beinhaltet, sondern nur einige JavaScript-Dateien. Alle Daten werden mit Hilfe von JavaScript-Funktionen und Objekten nachträglich geladen. Da die Webcrawler die JavaScript-Funktionen nicht interpretieren können, kann die Seitenstruktur samt allen Daten nicht richtig aufgebaut werden. Um dieses Problem zu umgehen, erstellen die Webentwickler meistens zwei unterschiedliche Versionen. Die eine ist 100% Ajax-basiert und die andere ist für die Suchmaschinen und älteren Webbrowser optimiert. Weil die Crawler über die *User-Agent*-¹⁰Parameter erkannt werden, entscheidet der Server, welche Version dem Besucher geliefert werden soll. In einigen Fällen wird die Option dem Benutzer angeboten, die Version selbst auszuwählen.

Es gibt bis jetzt noch keine bekannten Suchmaschinen bzw. Bot, die Ajax-Webseiten erfolgreich

⁹JavaScript Object Notation - <http://www.json.org/>

¹⁰User-Agent - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.43>

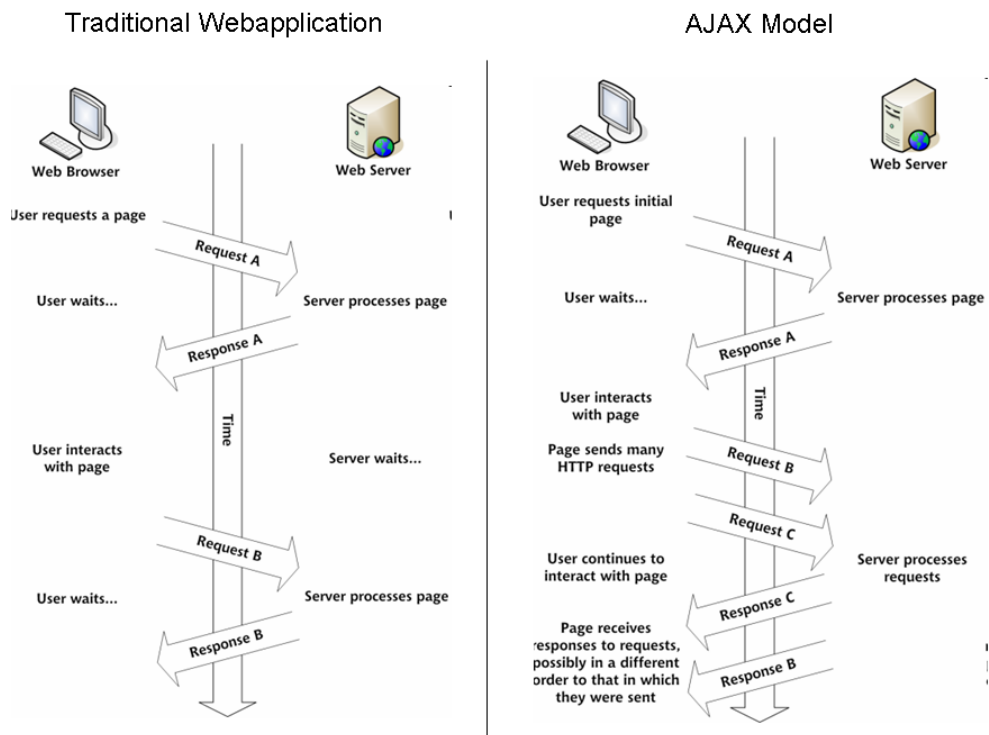


Abbildung 3.7: Vergleich zwischen den herkömmlichen Modellen (links) und dem Ajax-Modell (Quelle: sitepoint.com)

indexieren. Nicht einmal der Google-Bot¹¹ ist momentan dazu fähig. Das lässt vermuten, dass es keine triviale Aufgabe ist, einen Crawler für Ajax-Webseiten zu entwickeln. Demzufolge ist die Berücksichtigung von Ajax-basierten Onlineshops nicht als Ziel in diesem Projekt definiert. Dazu kommt das andere Argument, dass solche Shopsysteme nicht verbreitet sind und momentan eher die Ausnahme bilden.

3.4.4 Onlineshops mit Adobe Flash

Einige Shop-Betreiber versuchen die Benutzerfreundlichkeit zu verbessern und greifen auf Techniken wie animierte Navigationsmenüs und Produktpräsentation zurück. Um dies zu erreichen, wird manchmal von Adobe Flash Gebrauch gemacht. Solche Techniken erlauben ein Höchstmaß an Interaktivität und Benutzerfreundlichkeit. Besonders bei Modeartikeln neigen die Webmaster dazu, Flash zu verwenden, um die Produkte den Besuchern besser zu präsentieren. Leider gibt es eine Menge Nachteile. Zum einen ist die Entwicklung und die Pflege der Seite aufwendiger, zum anderen brauchen die Benutzer ein Browser-PlugIn (Adobe Flash Player), das nicht immer standardmäßig im Browser eingebaut ist. Und nicht zuletzt ist die Seite für die Suchmaschinen und somit auch für den bei diesem Projekt eingesetzten Webcrawler nicht lesbar.

¹¹Der Webcrawler von Google - <http://www.google.de/support/webmasters/bin/answer.py?answer=81766&ctx=sibling>

4 Allgemeine Grundlagen

4.1 Metasprachen

XML (Extensible Markup Language) ist, genauso wie HTML (Hypertext Markup Language), eine Auszeichnungssprache, die von SGML (Standard Generalized Markup Language) abgeleitet wurde - ein seit ca. 20 Jahren existierender internationaler Dokumentationsstandard (Fig. 4.1). Ursprünglich wurde sie für die Definition von Auszeichnungssprachen verwendet, mit deren Hilfe Textdokumente im elektronischen Format beschrieben wurden, was den Grundstein für die strukturierten Dokumente legte.

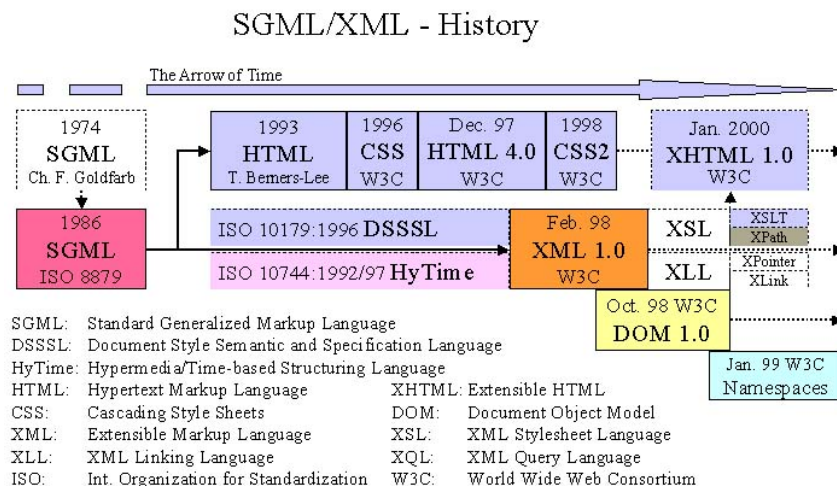


Abbildung 4.1: Entwicklung von SGML und verwandten Standards [FG99]

Eine Auszeichnung (engl. Markup) kann als eine Art Markierung bestimmter Abschnitte im Text angesehen werden. Eine Markup-Sprache ist daher eine Menge von Markups und Markup-Regeln, die Texte beschreiben und so die maschinelle Verarbeitung ermöglichen [SMB94]. Die Auszeichnungssprache spezifiziert, welche Auszeichnungen erlaubt sind, welche zwingend notwendig sind, wie sie im Text erkannt werden und was sie bedeuten. Ein Textdokument, das mit einer Markup-Sprache erstellt worden ist, beinhaltet also nicht nur den Text selbst, sondern auch zusätzliche Informationen über seine Teilabschnitte, wie zum Beispiel die Markierung der Überschrift, die Einführung oder eine Inhaltstabelle. Markiert werden sie mit den sogenannten Tags, die öffnend oder schließend sein können und den ausgezeichneten Textabschnitt umgeben. Die möglichen Tags, deren Reihenfolge und Attribute sowie die gesamte Dokumentenstruktur wird in der Dokumenttypdefinition (DTD, engl. Document Type Definition) festgelegt, die am Anfang

des Dokuments eingebettet wird. Jede Definition kann als eine Klasse von Dokumenten verstanden werden und die Dokumente selbst als Instanzen dieser Klasse.

4.1.1 Extensible Markup Language (XML)

Die SGML konnte sich wegen ihrer Komplexität nicht als Standard durchsetzen [Cla97]. Aus diesem Grund wurde von W3C (World Wide Web Consortium) die „Extensible Markup Language“ als vereinfachte Version der SGML entwickelt und als Standard spezifiziert, was besonders gut für den Dokumentenaustausch über das Netz geeignet ist. Die XML, genauso wie der Vorgänger SGML, erlaubt Definitionen von eigenen Markups und ermöglicht damit eine benutzerdefinierte Formatierung des Dokuments. XML hat auch die gleichen Sprachelemente, allerdings wurden viele Regeln festgesetzt und damit die komplette Sprache einfacher und verständlicher gemacht, was zu einer größeren Akzeptanz geführt hat. In Figur 4.2 wird die XML-Struktur an einem Beispiel anschaulich gemacht.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <my_band>
  <Bandname>Nero's Delight</Bandname>
- <musicians>
  - <musician>
    <name>Michael</name>
    <instrument>Trumpet</instrument>
  - <lives_in>
    <Town>Bonn</Town>
  </lives_in>
  </musician>
+ <musician>
  <!-- and so on -->
</musicians>
</my_band>
```

Abbildung 4.2: XML-Darstellung von Informationen (Quelle: w3c.de)

Jedes XML-Dokument fängt mit der XML-Deklaration an, in die spezielle Informationen für die Verarbeitung des Dokuments eingetragen werden, unter anderem die XML-Version, mit der das Dokument übereinstimmt, und die Codierung des Textes [Eck99]. Zusätzlich kann mit dem Parameter „standalone=yes/no“ eingestellt werden, ob die Dokumentendefinition im Dokument eingebettet wird oder in einer externen Datei, was gegebenenfalls in der zweiten Zeile geschieht. Durch die Angabe der Textcodierung ist es möglich, nicht nur ASCII-Zeichen zu verwenden, sondern auch Unicode und so das Speichern von Inhalten in verschiedenen Sprachen zu erlauben.

Ein XML-Dokument besteht aus einem oder mehreren Elementen. Jedes nicht leere Element besteht, genauso wie bei SGML, aus zwei Tags, einem öffnenden und einem schließenden. Eine Ausnahme stellt das leere Element dar, das nur ein Tag besitzt. Diese Tags sind in eckigen Klammern eingeschlossen, und das schließende Tag wird mit einem führenden Bruchstrich gekennzeichnet. Das folgende Beispiel zeigt, wie ein Preis-Element aussehen könnte:

```
<product>
<name>HP LaserJet 1200</name>
```

```
<price>120,75</price>
</product>
```

Zusätzlich lassen sich für bestimmte Elemente auch ein oder mehrere Attribute definieren, die aus einem Tupel - Name und Wert - bestehen und innerhalb des Start-Tags positioniert werden. Zum Beispiel kann man so das Element price um die Zusatzinformation „Währung“ erweitern:

```
<product>
<name>HP LaserJet 1200</name>
<price currency='EUR'>120,75</price>
<price currency='USD'>149,99</price>
</product>
```

Die Elemente können in andere Elemente eingebettet werden, so dass eine Baumstruktur entsteht. Das äußere Element heißt Wurzel und beschreibt den Anfang und das Ende des Dokumentes. Die inneren Elemente werden als Kinder oder Nachkommen betrachtet, die äußeren Elemente als Eltern oder Vorfahren, je nachdem, welchen Platz sie in der Elementehierarchie einnehmen.

Damit die XML-Dokumente maschinell verarbeitet werden können, müssen sie von einem Programm (dem XML-Processor) gelesen und geparkt werden. Das ist nur dann fehlerfrei möglich, wenn das XML-Dokument *gültig* ist, d. h., wenn es bestimmten Regeln entspricht. Gültig ist ein Dokument in erster Linie, wenn es *wohlgeformt*, also syntaktisch korrekt ist. Außerdem muss das Dokument eine *Dokumententypdefinition* besitzen und gemäß den dort enthaltenen Regeln strukturiert sein [SMYM⁺08][BPSM97]. Ein Dokument ist dann wohlgeformt, wenn die folgenden Bedingungen erfüllt sind:

- Es beinhaltet ein oder mehrere Elemente.
- Falls vorhanden, werden alle Elemente in ein einzelnes Root-Element eingeschlossen.
- Ein Element muss sowohl ein Starttag als auch ein Endtag haben, die leeren Elemente haben ausnahmsweise nur ein Tag, das einen führenden Schrägstrich beinhaltet.
- Alle Start- und End-Tags müssen richtig ineinander geschachtelt werden, d. h., wenn der Starttag im Content eines anderen Elementes ist, muss auch der Endtag im gleichen Element sein.
- Elementattribute müssen in Anführungsstrichen eingeschlossen sein.
- Markup-Zeichen, wie < oder &, dürfen im Content nicht vorkommen, sondern müssen durch dessen Referenzen, wie < oder & ersetzt werden.

4.1.2 Hypertext Markup Language (HTML)

Eine weitere von SGML abgeleitete Markup-Sprache ist HTML, die speziell für die Darstellung von Webseiten entwickelt wurde [DRJ99]. Diese Sprache bietet eine Vielzahl an Ausdrücken für Rich-Text, Multimedia und die Strukturierung der Dokumente. Das Besondere an HTML im Vergleich zu anderen Markup-Sprachen ist das Hypertext-Konstrukt, das die Verlinkung von Webdokumenten ermöglicht und ihnen so Interaktivität verleiht. Im Gegensatz zu SGML und XML ist

die Dokumenttypdefinition fixiert, so dass das Definieren der Elemente und Markups für jede Seite nicht notwendig ist. In HTML können nur bereits vordefinierte Tags, wie zum Beispiel `<body>` und `<head>`, vorkommen, und sie können weder geändert noch erweitert werden. Die DTD ist in jedem Internet-Browser integriert, weshalb die Dokumentinhalte wie Texte und Grafiken entsprechend der unterstützten HTML-Standards angezeigt werden. Genauso wie für die XML-Sprache werden laufend neue Versionen des HTML-Standards (Empfehlungen) von W3C entwickelt, die von den HTML-Entwicklern und den Browserherstellern berücksichtigt werden sollten. Die meisten Regeln für Dokumentgültigkeiten bei XML gelten auch mit einigen Ausnahmen für HTML. Leider halten sich nicht alle Entwickler daran, was ein Hindernis bei der Bearbeitung und Auswertung von HTML-Dokumenten darstellt. Auf dieses Problem wird in Unterkapitel 5.3 näher eingegangen.

4.2 Zugriff auf Metasprachen

4.2.1 Document Object Model

Das Document Object Model (DOM) ist eine von W3C standardisierte Schnittstelle (API) für XML- und HTML-Dokumente [Rob98]. Er definiert die logische Struktur der Dokumente, wie sie gelesen und manipuliert werden. Im Einzelnen wird in den DOM-Spezifikationen festgesetzt, wie Dokumente zu erstellen sind, aber auch die Navigation durch deren Struktur sowie das Hinzufügen, Ändern und Entfernen von Elementen und Inhalten. Die logische Struktur der Dokumente in DOM wird durch einen Baum aus Elementen repräsentiert (Fig. 4.3).

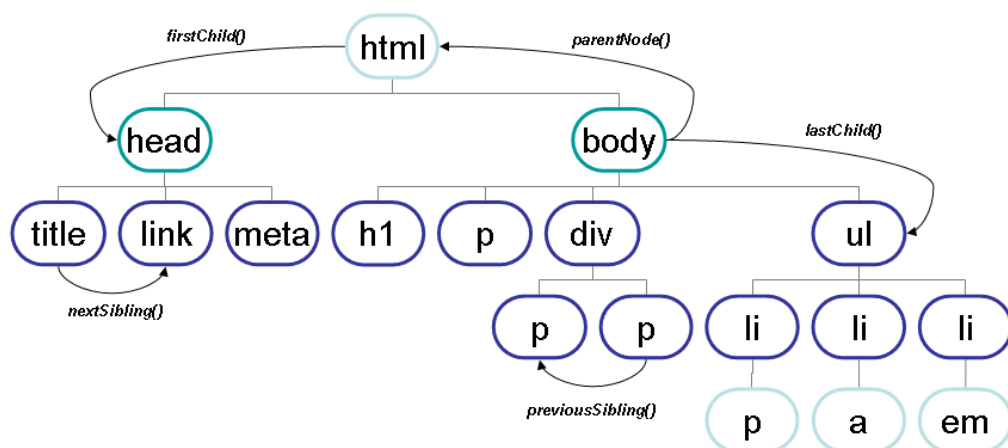


Abbildung 4.3: Repräsentation einer einfachen Html-Seite in DOM

Laut Definition soll DOM eine Standardschnittstelle liefern, die unabhängig von der Plattform und Applikation ist. Das DOM wird als ein abstraktes Modell spezifiziert, das in jede Program-

miersprache implementiert werden kann. Es existieren verschiedene Spezifikationen, die hierarchisch nach der Stufe der unterstützten Funktionalität geordnet sind. *Level0* spezifiziert die Grundfunktionen wie Lesen, Ändern und Navigieren durch Dokumente. Jede höhere Stufe (*Level1*, *Level2*, *Level3*) besitzt verschiedene Erweiterungen, wie z. B. HTML-Unterstützung, Events, Formatierung mit Stylesheets und Xpath. Der DOM ist auf das *OOP*¹-Paradigma ausgelegt und basiert auf einer abstrakten Klasse *Node*, von der sich fast alle anderen möglichen Klassen ableiten. Die wichtigsten davon sind *Document*, *Element*, *Attribute*, *Text* und *Notation*. Eine Ausnahme bildet z. B. die *NodeList*, die eine abstrakte Klasse für Listen von Nodes ist. Die Base-Klasse *Node* bietet genau definierte Funktionen für das Navigieren in der Elementehierarchie. Das geschieht entweder vertikal mit Aufrufen wie *parentNode()* und *firstChild()* sowie horizontal unter den Kindknoten eines Elternknotens - z. B. *nextSibling()*, *previousSibling()*. Zusätzlich gibt es Funktionen für das Hinzufügen, Bearbeiten und Löschen von Kindknoten.

Für den Umgang mit HTML-Dokumenten im Browser hat sich JavaScript etabliert. Der Browser interpretiert den JS-Code, führt ihn aus und modifiziert dementsprechend die Struktur der vom Server gelieferten Seite über das DOM-Interface. Dieser Vorgang fällt unter den Begriff Dynamic-HTML und erlaubt nachträgliche Änderungen der Webseite, selbst wenn sie komplett im Browser geladen wird. Trotz der Empfehlung der W3C wird der DOM leider nicht von allen Browsern gleich unterstützt. Besonders ältere Versionen haben ihre eigene Standards für den Aufbau des Elementenbaums und liefern oft unterschiedliche Ausgaben. Wie bereits im Unterkapitel 3.4.3 beschrieben, können zum Beispiel die *Crawler*² den Java-Script-Code überhaupt nicht interpretieren und ausführen.

4.2.2 XPath

XPath ist eine von W3C entwickelte Abfragesprache, mit der Teile eines XML-Dokuments adressiert werden können [Cla99]. Der Name kommt von der URL-ähnlichen Form der Xpath-Ausdrücke, die das Navigieren durch die hierarchische Struktur der XML-Dokumente beschreiben. Genauso wie bei dem Document Object Model wird ein XML-Dokument von XPath als ein Baum interpretiert, bei dem die Elemente des Dokuments als Knoten repräsentiert sind. Zusätzlich kann mit XPath im Dokumentenbaum überprüft werden, ob bestimmte Knoten einer Bedingung entsprechen oder ob die Pfade ein Muster beinhalten. In dem XML-Baum kann es folgenden Knotentypen geben:

- *Elemente* - repräsentieren ein Element im XML-Dokument und können als die Hauptknoten im Dokumentenbaum angesehen werden. Diese Knoten haben immer einen Elternknoten (die Wurzel bildet hier eine Ausnahme) und können alle Arten von Kindknoten besitzen.
- *Attribute* - für jedes der Attribute eines XML-Elements wird ein Attributknoten erstellt, der immer zwei String-Parameter besitzt, einen Namen und einen Wert. Diese Knoten werden als Blattknoten definiert und dürfen nur als Kindknoten von Elementknoten vorkommen.
- *Text* - repräsentiert den eingeschlossenen Textinhalt eines XML-Elementes. Genauso wie die Attributknoten können sie im Dokumentenbaum nur Kindknoten von Elementknoten sein und können selbst keine weiteren Kindknoten besitzen.

¹objektorientierte Programmierung

²Von Suchmaschinen benutzte Programme, die Webseiten anhand der Links besuchen, auslesen und indexieren

- *Kommentar* - repräsentiert die Kommentare in einem XML-Dokument. Es gelten die gleichen Regeln wie für die Text- und Attributknoten.

Die XPath-Ausdrücke werden immer im Hinblick auf einen Kontextknoten ausgewertet. Wenn die Wurzel die Rolle des Kontextknotens spielt, ist der Ausdruck absolut. Ansonsten ist er relativ zu dem aktuellen Knoten. Die Anfrage repräsentiert eine Navigation von dem Kontextknoten zum Suchknoten im Dokumentenbaum und filtert gegebenenfalls das Ergebnis anhand weiterer Ausdrücke. Ein Ausdruck besteht aus einem *Lokalisierungspfad*, anhand dessen eine Menge von Zielknoten ausgewählt wird. Der Lokalisierungspfad besteht an sich aus einem oder mehreren *Lokalisierungsschritten*, die durch einen Querstrich (/) getrennt sind. Zum Beispiel:

```
/catalog/office/printer[@type='laser']/hp/product/*
```

Wenn der Pfad absolut ist, wie in unserem Beispiel, fängt er mit einem Querstrich an. Ansonsten fängt ein Lokalisierungspfad direkt mit dem ersten Lokalisierungsschritt an. Die Lokalisierungsschritte haben folgende Syntax:

```
Axis::NodeTest [Predicate]
```

- *Axis* - (dt. Achse) gibt die Richtung an, in die nach dem *NodeTest* gesucht wird [GW00]. Im Gegensatz zu Pfadausdrücken im Dateisystem, die nur in eine Richtung (nach unten) gehen, können die XPath-Ausdrücke auch nach oben in Richtung Wurzel oder seitwärts nach Nachbarknoten suchen. Es existieren 13 Elemente, die als *axis* definiert werden. Die meisten davon werden anschaulich in Abbildung 4.4 dargestellt, der Kontextknoten ist mit *self* markiert. Der Parameter *axis* ist optional, und falls nichts anderes angegeben wird, wird defaultmäßig *child* verwendet. Im Folgenden werden einige Beispiele vorgestellt und erläutert:

- `head/child::meta` - adressiert alle *Meta*-Elemente (identisch mit `head/meta`).
- `head/parent::` - selektiert den Elternknoten vom *head* (äquivalent zu `head/..`).
- `/html/descendant::a` - selektiert alle Nachkommen des Knotens *html* mit Name *a*.
- `div/descendant-or-self::div` - liefert alle tiefer stehenden *div*-Knoten (inkl. den Kontextknoten).
- `p/ancestor::div` - liefert alle Vorfahren des Kontextknotens, die *div* heißen.
- `li/following-sibling::li` - adressiert alle nachfolgenden *li*-Elemente, die im XML-Baum rechts stehen; das Sternchen steht als Wildcard und selektiert alle Elemente.
- `div/a/attribute::href` - selektiert das *href*-Attribut von den *a*-Knoten, die unterhalb des Kontextknotens sind (identisch mit `div/a/@href`).

- *NodeTest* - (dt. Knotentest) spezifiziert den Typ oder den Namen des auszuwählenden Knotens. Wie bereits beschrieben, können dies Knoten vom Typ *Element*, *Attribut*, *Text* und *Kommentar* sein. Wenn der gesuchte Knoten vom Typ *Element* oder *Attribut* ist, muss sein Name angegeben werden. Alternativ kann das Sternchen (*) als Wildcard für alle Knoten verwendet werden. Zum Beispiel:

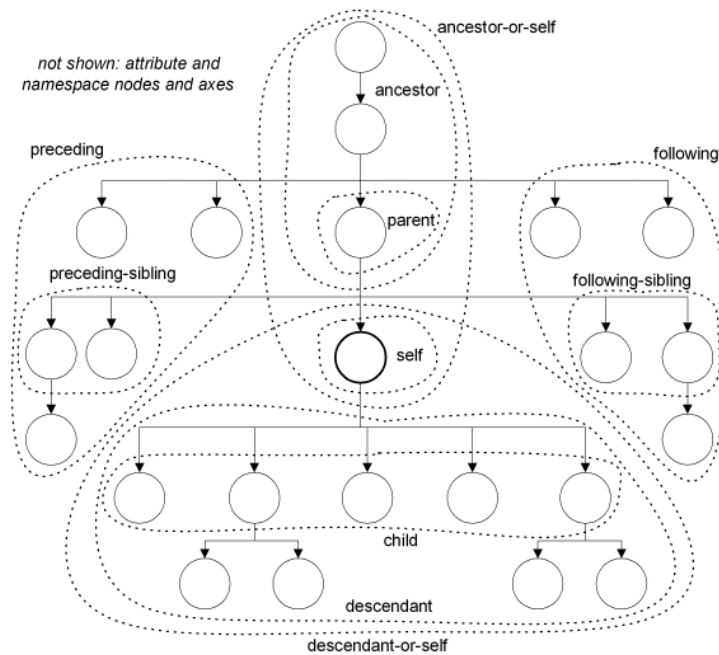


Abbildung 4.4: Graphische Darstellung der möglichen Achsen in XPath (Quelle: [GW00]).

- `product/attribute::id` - Im ersten Lokalisierungsschritt wird der Name *product* des Elements angegeben, im zweiten steht der Name des Attributknotens *id*.
- `/html/head/descendant::*` - Die ersten zwei Lokalisierungsschritte adressieren den *head*-Knoten, der dritte Lokalisierungsschritt selektiert alle Nachkommen des Kontextknotens, unabhängig von deren Name oder Typ.

Zusätzlich können die Text- und Kommentarknoten in einem Schritt anhand von deren Typ markiert werden. Als *NodeTest* kommt dann *text()* beziehungsweise *comment()* vor. Zum Beispiel lassen sich mit dem folgenden Lokalisierungspfad alle Textknoten, die als Nachkommen des Kontextknotens auftauchen, ausgeben:

```
/html/body/descendant::text()
```

- Predicate - ist ein optionaler Parameter und spielt die Rolle eines Filters, anhand dessen die Auswahl der Knoten eingegrenzt wird. Die Bedingung wird in eckige Klammern gesetzt und direkt hinter dem Knotentestelement positioniert. Prädikate können XPath-Ausdrücke beinhalten, die an sich aus beliebigen XPath-Funktionen oder Lokalisierungspfaden stammen. Zum Beispiel kann die Position des Knotens mit Hilfe der *Position()*-Funktion angegeben werden. Alternativ kann für die Position eine abgekürzte Syntax verwendet werden, indem sie als Zahl in eckige Klammern gesetzt wird. Das folgende Beispiel stellt zwei unterschiedliche Möglichkeiten vor, um den dritten *tr*-Knoten auszuwählen:

```
/html/body/table/tr[3]/td/a/text()
/html/body/table/tr[position()=3]/td/a/text()
```

Außer der Position lassen sich auch Attributwerte oder XPath-Funktionen als Bedingung setzen. Auch komplette Lokalisierungspfade können eingesetzt werden. Mehrere Prädikate können kombiniert und logisch verknüpft werden. Eine *AND*-Verknüpfung kann realisiert werden, indem die Prädikate nebeneinander gesetzt werden. Boolesche Ausdrücke können mit Hilfe von *AND*-, *OR*- und *NOT*-Operatoren implementiert werden. Die folgenden Beispiele machen die Benutzung von Prädikaten in XPath deutlich:

- `/catalog/product[@id=3450]/@name` - gibt den Wert des Attributs *name* für den Knoten *product* mit dem Attribut *id=3450* zurück.
- `/catalog/product[last()]/@id` - gibt den Wert von *id* für den letzten Knoten *product* an.
- `/catalog/product[contains(@label,'black')[@price < 500]]` - liefert alle Produktknoten, deren Attribut *label* den Text „black“ beinhaltet und deren Attribut *price* kleiner 500 ist (< steht für das Zeichen „<“).
- `/html/body/div[2]/p[a]` - selektiert alle Knoten mit dem Namen *p*, die unterhalb des zweiten *div*-Knotens sind und mindestens einen Kindknoten *a* beinhalten.
- `//div[a[contains(@src,'ibm.com')]]` - ist ein komplizierterer XPath-Ausdruck. Die zwei Querstriche (`//`) am Anfang selektieren alle Knoten mit dem Namen *div*, unabhängig von der Tiefe und Position im XML-Baum. Zusätzlich wird im Beispiel als Bedingung geprüft, ob mindestens ein Kindknoten *a* existiert, dessen *src*-Attribut den Text „ibm.com“ beinhaltet. Anders formuliert, der Ausdruck liefert alle *div*-Elemente auf einer Seite, die Links beinhalten, die zu einer Seite auf *ibm.com* führen.

Wenn in einem XPath-Ausdruck die zwei Querstriche vorkommen, werden ausgehend von dem Kontextknoten alle Knoten rekursiv in die Tiefe durchsucht. Falls `//` am Anfang des Ausdrucks vorkommt, wie in dem oben gegebenen Beispiel, wird die Wurzel als Kontextknoten betrachtet. Kommen die zwei Querstriche zwischen zwei Lokalisierungsschritten oder in einem Prädikat vor, dann wird die aktuelle Position als Kontext genommen. Zum Beispiel:

- `//img` - markiert alle *img*-Knoten im Dokument.
- `/html/body/table[3]//img` - markiert alle *img*-Knoten, die unterhalb eines bestimmten Knotens liegen.
- `//table[//a]` - adressiert alle *table*-Knoten, die mindestens einen Nachkommenknoten *a* besitzen.

Ein Ausdruck kann sowohl ein einziges Element liefern als auch ganze Unterbereiche des Dokuments. Das Ergebnis ist immer das letzte Element des Pfadausdrucks und kann eine Menge von Knoten sein, eine Zahl oder ein String. Außerdem kann das Ergebnis ein boolescher Wert sein, falls der Ausdruck ein Funktionsaufruf war. Es gibt verschiedene vordefinierte Funktionen, die in den XPath-Ausdrücken vorkommen können, wie zum Beispiel Knotenmengen-, String-, Number- und Boolean-Funktionen. Für eine vollständige Liste aller Funktionen wird auf [Cla99] verwiesen.

4.3 XML-Datenbanken

Die Datenverarbeitung mittels herkömmlichen XML-Dateien, funktioniert gut bei kleinen Mengen an Daten, aber bei den meisten Anwendungen, wo es auf einer guten Performance und strikten Datenintegration auf einer großen Basis an Daten ankommt, entstehen Probleme. Eine Alternative wäre die Daten in einer herkömmlichen relationalen Datenbank zu verwalten. Seit Jahrzehnten ist die relationale Datenbank die meistverbreitete Form, Anwendungsdaten dauerhaft zu speichern. Dort werden die Daten „flach“ in tabellarischer Form gespeichert. Wenn ein Objekt in der Datenbank zu speichern ist, muss es in einer speziell dafür vorgesehene Tabelle eingefügt werden, wobei jede Objekteigenschaft eine Spalte in der Tabelle hat. Die Beziehungen der gespeicherten Objekten werden durch Tabellenverknüpfungen realisiert. Mit der Entwicklung des OOP-Paradigmas, das heutzutage immer öfter zum Einsatz kommt, entstehen neue Anforderungen für eine vereinfachte und effiziente Art, hierarchische Objektdaten zu speichern. Es existieren objektorientierte Datenbanken, in denen die Daten hierarchisch als komplexe Objekte zusammen mit den zugehörigen Attributen und Eigenschaften gespeichert werden. So ist es nicht notwendig, komplexe Tabellen-Joins zu entwerfen, um bestimmte Objekte oder Objektdaten zu extrahieren, wie das bei den relationalen Datenbanken üblich ist.

XML-Datenbanken [Fos] funktionieren genau so wie Objektdatenbanken, und die Daten sind hierarchisch strukturiert, wobei statt der Anwendungsobjekte XML-Dokumente als Objekte verwaltet werden. Mit der verbreiteten Nutzung von XML-Dokumenten, besonders bei der Kommunikation, hat dieser Ansatz den Vorteil, Daten im ursprünglichen XML-Format zu verarbeiten und zu sichern, ohne dass eine Konvertierung notwendig ist. Genauso wie die SQL³ bei den relationalen Datenbanken lassen sich Daten mit einer Abfragesprache auslesen und modifizieren. Als Standard hat sich die XQuery etabliert, die auch vom W3C standardisiert und auf XPath ausgelegt ist. Allerdings ist XQuery als eine funktionale Sprache viel flexibler und mit einigen Erweiterungen, z. B. Unterstützung eigener Funktionen, mächtiger [SBF00]. Es existieren zwei Typen von XML-Datenbanken [Hal05]:

- *XML-Enabled Datenbanken (XEDB)* - haben als Basis eine relationale Datenbank, auf der alle Operationen durchgeführt werden. Dabei werden die Strukturen aus den XML-Dateien auf Tabellen in der Datenbank abgebildet. Das hat zum Vorteil, dass viele Datenbankanforderungen, wie Indexierung, Sicherheit, Transaktionen, Trigger, Multiuser - Fähigkeit gleich erfüllt werden können. Als Nachteil erweist sich aber auch, dass die XML-Struktur sehr komplex werden kann. Das führt zu einer expandierenden Anzahl an Tabellen in der Datenbank, und stößt irgendwann an die Grenzen, was zu einem Verlust der Tiefe der XML-Struktur führen kann.
- *XML - Native Datenbanken (NXD)* - sind speziell auf die Verarbeitung von XML-Dateien ausgerichtet. Sie unterstützen die o.g. wichtigen Anforderungen, wobei die interne Verwaltung der Daten, auf Basis von XML geschieht. Diese finden vor allem Anwendung, wenn es um die Speicherung dokumentenorientierter Daten geht, da Dokumentenordnung, Bearbeitungsanweisungen, Kommentare und so weiter besser verwaltet werden können als mit einer relationalen Datenbank.

³Structured Query Language

Der wichtigste Vorteil, XML als Speicherformat für Datenbanken zu benutzen ist, dass er selbstbeschreibend ist. Das bedeutet, Eigenschaften - Name und -Typ sind in den Tags definiert, so gesehen existiert keine Semantik und die Daten können in einer Baum- oder Graph-Struktur beschrieben werden. Der größte Nachteil ist das schnell wachsende Volumen an Daten, die verarbeitet werden und die Notwendigkeit, den Text vor der Bearbeitung zu parsen. Das führt zu einer schlechten Performance.

4.4 Information Retrieval

Information Retrieval (Abk. IR) beschäftigt sich mit der Darstellung, Speicherung, Organisation von und Zugang zu Informationen [ByRN99]. Das Ziel ist, den Benutzer mit Informationen zu versorgen, nach denen er sucht und die für ihn relevant sind. Ein wichtiger Punkt im IR-Bereich ist, die Informationsbedürfnisse des Users zu charakterisieren. Das geschieht in Form einer Abfrage, die von dem IR-System verarbeitet werden kann. Häufig besteht die Query aus einigen Schlüsselwörtern, die das Informationsbedürfnis des Users am besten beschreiben. Im Gegensatz zu Data Retrieval, wo die Daten vollständig strukturiert sind (z. B. in relationalen Datenbanken), muss ein IR-System mit halb strukturierten und unstrukturierten Dokumenten zurechtkommen. Meistens werden Textdokumente ohne vorhandene Kategorisierung oder Auszeichnungen im Text dargestellt, wie zum Beispiel Webseiten im WWW. Das System muss in der Lage sein, den Inhalt der Dokumente zu interpretieren und syntaktische und semantische Informationen aus dem Dokument zu extrahieren. Aus diesem Grund kann ein IR-System, anders als bei der Suche in einer DB, auch Informationen liefern, die nicht vollständig dem Informationsbedarf des Benutzers entsprechen. Diese Abweichungen werden anhand eines errechneten Rankings gemessen, das die Dokumentenrelevanz der Ergebnisse repräsentiert. Im Aufgabenbereich der IR-Systeme kommen außerdem die effiziente Speicherung von Dokumenten und die Indexierung, das Ausführen von performanten Suchanfragen und die Entwicklung von Ranking-Algorithmen für bessere Ergebnisse hinzu.

„Das Hauptziel eines IR-Systems ist, alle relevanten und möglichst wenige nicht-relevante Dokumente zu liefern.“[ByRN99]

4.4.1 Architektur eines IR-Systems

Die Abbildung 4.5 gibt einen groben Überblick über die Architektur eines IR-Systems [ByRN99]. Alle zu durchsuchenden Textdokumente befinden sich in einer *Text Database*. Das Modul *DB Manager Module* entscheidet, welche Dokumente durchsucht werden und welche *Text Operations* anzuwenden sind. In diesem Schritt werden die Dokumente analysiert und für die weitere Verarbeitung vorbereitet. Anschließend wird die Indexierung angestoßen, wodurch der *Index* gebildet wird. Ein Index ist eine Datenstruktur von hoher Kapazität, die effiziente Suchoperationen ermöglicht. Nachdem der Index vollständig aufgebaut ist, können sämtliche Suchoperationen nach Dokumenten ausgeführt werden. Der Benutzer gibt seine Informationsbedürfnisse ein, die mit den *Text Operations* in einer *Query* konvertiert werden. Nachdem diese Abfrage im Index ausgeführt wird, liefert das System eine Menge von Dokumenten - die Ergebnismenge (*retrieved docs*). Als letzter Schritt, bevor der User die Ergebnisse sieht, wird das *Ranking* der gefundenen Dokumente anhand eines speziellen Algorithmus berechnet. Normalerweise entsteht ein Kreis, weil die Benutzer weitere Abfragen ausführen und so ein Refining der Ergebnisse durchführen können.

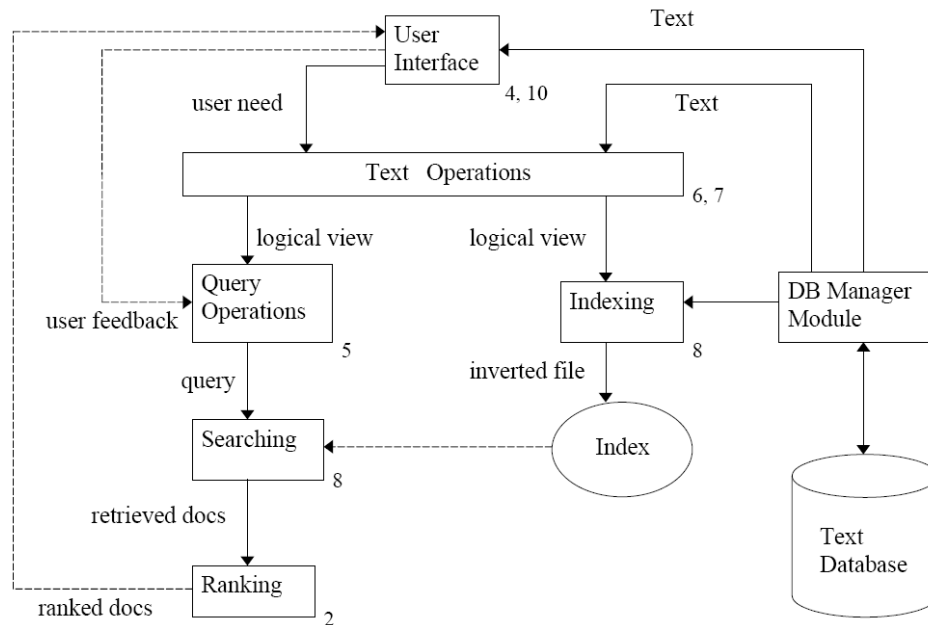


Abbildung 4.5: Ablaufplan eines Retrieval-Prozesses

4.4.2 Textoperationen

Ein Dokument kann als eine Menge von Schlüsselwörtern oder Indextermen betrachtet werden, die eine logische Sicht des Dokuments definieren. Eine der Aufgaben eines IR-Systems ist es, diese Wörter automatisch aus dem Text zu generieren. Anschließend müssen sie so gespeichert werden, dass ein effizientes Suchen nach bestimmten Informationen ermöglicht wird wie auch der einfache Zugriff auf die gesuchten Seiten. So entsteht ein Index, der aus einer Sammlung von Keywords mit Verweisen zur Information besteht (siehe Unterkapitel 4.4.3). Als Beispiel kann man auf eine alte und weitverbreitete IR-Technik verweisen - das Erstellen einer Indexliste für lange Texte, wie zum Beispiel Lehrbücher. Dort werden vom Autor alle wichtigen Schlüsselwörter mit Verweisen zu relevanten Seiten aufgelistet.

Bevor die Dokumente in den Index aufgenommen werden können, müssen sie entsprechend vorbereitet werden, indem alle Schlüsselwörter im Text erkannt und extrahiert werden. Es existieren verschiedene Möglichkeiten, wie die Dokumente in den Index aufgenommen werden. Die einfachere Variante ist die sogenannte *Volltextindexierung*. Mit dieser Technik werden alle Wörter im Text ausnahmslos als Indexterme betrachtet, was den Nachteil hat, dass sie bei einer größeren Anzahl an Dokumenten nicht sehr effizient ist. Daher werden die Texte auf sinnvolle und unikale Keywords mit Hilfe verschiedener Techniken reduziert [MRS09], die im Folgenden vorgestellt werden:

Tokenisierung Diese Technik hat die Aufgabe, den Text in Einzelteile (Tokens) zu zerlegen. Die Hauptschwierigkeit liegt darin, die richtigen Trennpositionen zu bestimmen. Die ein-

fachste Variante ist es, als Trennzeichen die Interpunktionszeichen im Text zu verwenden, also Leerzeichen, Komma, Punkt und andere.

Text	<i>The new iMac is a high-performance computer with beautiful design</i>
Tokens	The new iMac is a high performance computer with beautiful design

Allerdings gibt es viele sprachspezifische Sonderfälle, die eine spezielle Vorgehensweise erfordern. Zum Beispiel kann es zu Problemen kommen, wenn der Apostroph für die Trennung benutzt wird. Dann führt die Zerlegung von Sätzen mit Wörtern wie *aren't* oder *can't* zu sinnlosen Tokens (*aren|t|can|t*), die bei einer Suche nicht nützlich sind. Im Allgemeinen ist es von Vorteil, wenn die Sprache des Textes im Voraus bekannt ist. Dann können speziell angepasste Tokenizer verwendet werden, die sprachspezifische Besonderheiten berücksichtigen. In der englischen und französischen Sprache können spezielle Regeln für Wörter mit dem Apostroph und Bindestrich erstellt werden. Aber auch die deutsche Sprache stellt für den Tokenizer eine Herausforderung dar, weil oft ein Wort aus mehreren Wörtern zusammengesetzt ist, wie zum Beispiel *Versicherungsgesellschaftsangestellter*. Die Einzelwörter aus den *Komposita*⁴ zu entnehmen wird normalerweise mit Hilfe von sprachspezifischen Vokabularen gemacht. Bei einigen asiatischen Sprachen werden die Wörter ohne Leerzeichen geschrieben und die Tokenisierung kommt ohne ein spezielles Vokabular nicht aus.

Sehr oft müssen bestimmte Wortkombinationen im Text als ein Token erkannt werden. Für die Erkennung von speziellen Objekten wie Namen (Stadtnamen, Personennamen, Produktnamen etc.), Daten oder Zahlen sind sie besonders zu berücksichtigen. Bei Objekten mit einem bestimmten Format, wie zum Beispiel Daten, Zahlen, E-Mails und Internetadressen, werden spezielle Regeln (z. B. reguläre Ausdrücke) erstellt, die nach einem bestimmten Muster suchen. Die Namen können mit Hilfe von individuellen Vokabularen erkannt und als ganze Token aufgenommen werden. Im Folgenden sehen wir eine Tabelle mit Sonderfällen, die bei der Tokenisierung zu beachten sind:

Text	Tokens
<i>He's left-handed and can't write fast.</i>	He's, left-handed, and, can't, write, fast
<i>Peter Smith wurde als Versicherungsmakler am 04. März 2007 angestellt.</i>	Peter Smith, wurde, als, Versicherung, Makler, am, 04. März 2007, angestellt
<i>The distance between Los Angeles and San Francisco is exactly 614.76 km.</i>	The, distance, between, Los Angeles, and, San Francisco, is, exactly, 614.76km

Stoppwort-Filter Mit diesem Verfahren werden oft vorkommende Wörter wie Konjunktionen und Präpositionen (*und, oder, in, der, das, die etc.*) aus dem Text entfernt und nicht indexiert. Solche Wörter beinhalten keine wertvollen Informationen und können nicht als relevante Terme angesehen werden. Normalerweise werden spezielle Listen mit Stoppwörtern für jede spezifische Sprache definiert. Bei Sprachen aus einer gemeinsamen Sprachgruppe, wie zum Beispiel *romanische* oder *germanische* Sprachen, lassen sich auch gemeinsame Filter

⁴Substantive, die durch Komposition entstehen

benutzen. Im folgenden Beispiel werden typische Stoppwörter für die englische Sprache aufgelistet:

a an and are as at be by for from
 has he in is it its of on that the
 to was were will with

Es existieren auch alternative Verfahren für den Aufbau der Stoppwort-Liste. Anstatt die Wörter manuell für jede Sprache und jeden Einsatzbereich zu spezifizieren, werden Techniken benutzt, die automatisch die Liste generieren. Sie basieren auf der Term-Häufigkeit in einem Dokument, wobei Wörter, die eine bestimmte Grenze überschreiten, in die Stoppliste aufgenommen werden.

Stemming und Lematisierung Mit Hilfe dieser Techniken werden die Wörter auf die grammatischen Stämme reduziert (engl. *stem*). So werden verschiedene Ableitungen von einem Stammwort erkannt und berücksichtigt. Zum Beispiel können von dem Wort *rechnen* Wörter wie *berechnen*, *rechnerisch*, *Rechnung* etc. abgeleitet werden. Es existieren verschiedene Algorithmen für dieses Problem, die meist wegen der unterschiedlichen Morphologie für jede Sprache angepasst werden müssen. Der Vorteil dabei ist zum einen die Reduzierung der Terme in den Dokumenten und zum anderen die genauere Bestimmung der Relevanz der Terme. *Stemming* ist die einfachere Variante. Um die Wörter in eine Basisform zu bringen, werden syntaktische Änderungen an den Wörter vorgenommen, meistens durch das Weglassen von bestimmten Präfixen oder Suffixen. Es existieren verschiedene Stemming-Algorithmen, wobei der bekannteste *Porter Stemmer* ist, der für englische Texte verwendet wird. Die folgende Tabelle zeigt anhand eines Beispiels, wie Stemming-Regeln angewendet werden:

Regel	Beispiel
SSES -> SS	classes -> class
IES -> I	activities -> activiti
SS -> SS	class -> class
S ->	cats -> cat

Der Nachteil beim Stemming-Verfahren ist dessen Ungenauigkeit bei der Bildung der Stammformen des jeweiligen Wortes. Abhängig von den definierten Regeln kann zum Beispiel für das Verb *saw* nur ein *s* geliefert werden, wobei *see* das bessere Ergebnis wäre. Die *Lematisierung* gilt als eine bessere Alternative zum Stemming und nutzt ein Vokabular. Zusätzlich wird eine morphologische Analyse der Wörter durchgeführt. Der Nachteil bei diesem Verfahren ist die schlechtere Performance. Die folgende Tabelle zeigt einige Ergebnisse der Lematisierung:

bin,bist,ist -> sein
 am,are,is -> be
 operating -> operate
 operation -> operate
 sprung -> springen
 went -> go

4.4.3 Index

Meistens müssen die IR-Systeme sehr große Sammlungen von Text bearbeiten, oft im Gigabyte-Bereich. Die triviale Textsuche wäre in diesem Fall nicht anwendbar. Um die Suche zu beschleunigen, werden die Dokumente in eine optimierte Datenstruktur überführt, den Index. Dort werden die Schlüsselwörter, deren Position im Dokument und die entsprechenden Verweise zum Dokument gespeichert. Bevor der Text indexiert wird, wird er analysiert, indem Textoperationen (siehe Unterkapitel 4.4.2) wie Tokenising, Stoppwort und Stemming angewendet werden. Die am meisten verwendete und bekannte Indexierungstechnik ist die invertierte Indexierung (*Inverted Files*).

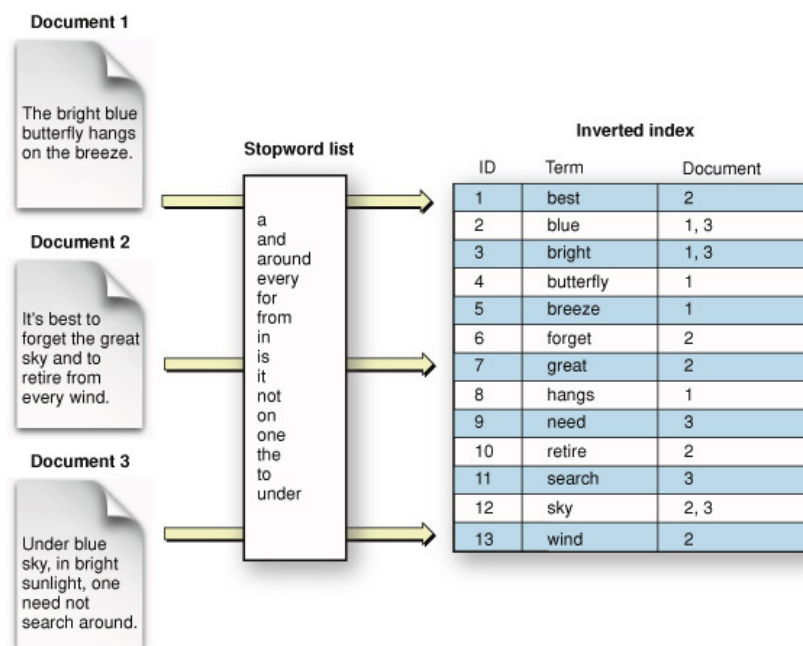


Abbildung 4.6: Struktur und Aufbau von einem invertierten Index mit Hilfe einer Stoppwortliste; in der Positionsliste (Spalte *Document*) stehen die Positionen der Terme im Text (Quelle: developer.apple.com)

Der invertierte Index besteht aus einem Vokabular und einer Häufigkeitsliste [ByRN99]. Im Vokabular werden alle Indexterme ohne Duplikate aufgelistet. In der Positionsliste (*Occurrences*) wird zu jedem Term die Position, an der er auftritt, gespeichert. Die Positionen können als Zeichen- oder Wortpositionen angegeben werden (Fig. 4.6). Im zweiten Fall werden die *Phrase* und *Proximity* Queries beschleunigt (siehe Unterkapitel 4.4.6). Das Vokabular braucht relativ wenig Speicherplatz. Nach dem Heaps'schen Gesetz kann die Größe mit $O(n^\beta)$ abgeschätzt werden [Hea78], wobei β abhängig vom Text einen Wert zwischen 0 und 1 einnimmt (in der Praxis $0.4 < \beta < 0.6$). Die Positionsliste braucht mehr Speicherplatz, weil sie für jedes Wort im Text einen Verweis enthält. Der Platzbedarf wird in diesem Fall mit $O(n)$ geschätzt und kann mit Hilfe von Textoperationen leicht reduziert werden. Mit der Technik *Block-Adressierung* wird dagegen der Platzbedarf wesentlich reduziert. Der Text wird in Blöcke aufgeteilt, und in die Positionsliste kommt anstatt die Position des Wortes im Text die Nummer eines Blocks. Mit Hilfe dieser Technik wird erheblich Speicherplatz eingespart, wobei die Größe der Positionsliste in der Praxis um die

5% des Textvolumens beträgt. Zum einen gibt es weniger Blöcke als Positionen im Text, demzufolge auch weniger Blockadressen. Zum anderen gibt es weniger Verweise auf Terme in der Positionsliste. Nachteilig wird dies jedoch bei der Kontextsuche, wo mehrere benachbarte Blöcke untersucht werden müssen.

4.4.4 IR-Modelle und Ranking

Dokumente werden als eine Menge von Indextermen (Schlüsselwörtern) repräsentiert. Bei der Suche nach Dokumenten in einem sehr großen Pool ist zu beachten, dass nicht jedes Schlüsselwort für die Beschreibung des Dokumentinhalts gleich wichtig ist. Zum Beispiel ist ein Wort, das in fast jedem Dokument auftaucht, wertlos für die Beschreibung des Inhaltes. Dagegen grenzen eindeutige Wörter, die nur in ein paar Dokumenten auftauchen, die möglichen Ergebnisse deutlich ein. Deswegen haben die Indexterme in jedem Dokument eine Gewichtung, die von deren Wichtigkeit im jeweiligen Dokument abhängt [ByRN99].

Anschließend werden einige Definitionen wiedergegeben, die in jedem der vorgestellten IR-Modelle gültig sind. Wenn k_i ein Indexterm ist und d_j ein Dokument, so ist $w_{i,j} \geq 0$ eine Gewichtung, die die Wichtigkeit des Terms im Dokument angibt. Das Dokument kann auch als ein Vektor aus allen Gewichtungen der Indexterme repräsentiert werden - $\vec{d}_j = (w_{1,j}, w_{2,j}, \dots, w_{i,j})$. Die Menge der Indexterme ist endlich und kann so formal aufgeschrieben werden - $K = k_1, k_2, \dots, k_t$. Zusätzlich kann eine Funktion $g_i(\vec{d}_j) = w_{i,j}$ definiert werden, die die Gewichtung des Indexterms k_i im Dokument d_j zurückliefert.

Boolesche Modell

Das boolesche IR-Modell basiert auf der Mengentheorie und ist ziemlich einfach und intuitiv. Es kommt meistens bei der Suche nach Dokumenten in Volltextdarstellungen zum Einsatz. Die Gewichte der Indexterme können entweder 0 oder 1 sein ($w_{i,j} \in \{0, 1\}$), d. h., ein Indexterm kann in einem Dokument entweder vorhanden oder nicht vorhanden sein. Daraus folgt, dass auch kein Ranking der Dokumente möglich ist. Ein Dokument kann entweder relevant oder nicht relevant sein, was als der größte Nachteil angesehen wird. Die Abfragen werden als boolesche Ausdrücke formuliert, sie bestehen aus Indextermen und den drei booleschen Operationen AND, OR, NOT und lassen sich in die disjunktive Normalform (DNF) konvertieren. Zum Beispiel lässt sich die folgende Query in DNF konvertieren und als Vektor aufschreiben:

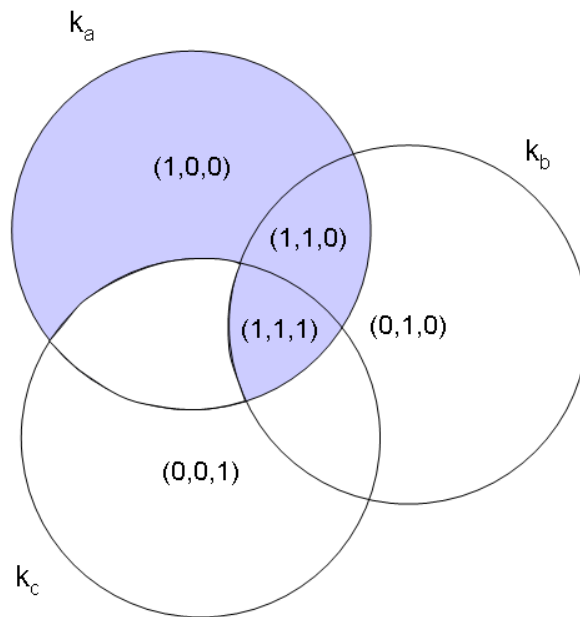
Query	$q = k_a \wedge (k_b \vee \neg k_c)$
Vektor	$\vec{q}_{dnf} = (1, 1, 1) \vee (1, 1, 0) \vee (1, 0, 0)$

wobei jede Komponente ein gewichteter Vektor (k_a, k_b, k_c) ist, genannt Konjunktiv-Komponente (q_{cc}) (Fig. 4.7).

Die Ähnlichkeit eines Dokuments zu der Query ist folgendermaßen definiert:

$$sim(d_j, q) = \begin{cases} 1 : & \text{if } \exists \vec{q}_{cc} | \vec{q}_{cc} \in \vec{q}_{dnf} \text{ und } \forall k_i, g_i(\vec{d}_j) = g_i(\vec{q}_{cc}) \\ 0 : & \text{sonst} \end{cases}$$

Ein Dokument d_j ist für die Query q nur dann relevant, wenn $sim(d_j, q) = 1$. Das ist dann erfüllt, wenn eine konjunktive Komponente \vec{q}_{cc} in der Query existiert, die mit allen Komponenten

Abbildung 4.7: Graphische Darstellung der drei Terme in der Query q

des Dokuments d_j gleich ist.

Vektormodell

Im Gegensatz zum einfachen booleschen Modell, bei dem die Dokumente strikt in zwei Mengen (relevante und nicht relevante) unterteilt sind, bietet der Vektormodell-Ansatz die Möglichkeit eines partiellen Matchings. Die Grundidee ist, natürliche Zahlen als Gewichte der Indexterme in die Dokumente und die Query einzuführen. Die Ähnlichkeit zwischen der Query und den Dokumenten wird anhand dieser Werte kalkuliert. Je ähnlicher ein Dokument ist, desto besser steht es in der Rankingliste. So werden sehr präzise Antwortmengen erreicht und der Benutzer kann zwischen wenig relevanten und sehr relevanten Dokumenten auswählen.

Ähnlich wie beim booleschen Modell wird jedes Dokument als ein mehrdimensionaler Vektor aus den Gewichten der Indexterme repräsentiert: $\vec{d}_j = (w_{1,j}, w_{2,j}, \dots, w_{t,j})$, wobei $w_{i,j} \geq 0$. Die Queries werden ebenfalls als Vektoren dargestellt: $\vec{q} = (w_{1,q}, w_{2,q}, \dots, w_{t,q})$. Die Ähnlichkeit eines Dokuments mit der Query wird als die Korrelation der beiden Vektoren \vec{q} und \vec{d}_j und formal als der Kosinus der Winkel zwischen den Vektoren definiert:

$$\text{sim}(d_j, q) = \cos(\vec{d}_j, \vec{q}) \quad (4.1)$$

$$= \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}_j| |\vec{q}|} \quad (4.2)$$

$$= \frac{\sum_{i=1}^t w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^t w_{i,j}^2} \cdot \sqrt{\sum_{i=1}^t w_{i,q}^2}} \quad (4.3)$$

Demzufolge liefert die Funktion eine reelle Zahl zwischen 0 und 1, was auch das Ranking der Dokumente repräsentiert. Bei 1 stimmt das Dokument komplett mit der Query überein, bei 0 besitzt es überhaupt keine Relevanz. Um das Ranking zu bekommen, müssen vorher die Termgewichte automatisch berechnet werden. Die bekannteste und meistverwendete Technik basiert auf dem Clustering-Verfahren. Das Ziel des Clusterings ist, eine Menge von Objekten automatisch in Cluster aufzuteilen. Dabei sollen die Objekte in einem Cluster möglichst ähnlich sein (Intra-Clustering), und die Objekte in unterschiedlichen Clustern sollen möglichst unterschiedlich sein (Inter-Clustering). Im IR-Bereich ist es ausreichend, mit zwei Clustern zu arbeiten; in den ersten kommen die relevanten Dokumente zu einer gegebenen Query, in den zweiten alle anderen Dokumente. Die Query kann als eine vage Spezifizierung der Menge der relevanten Dokumente betrachtet werden. Damit das System die Dokumente richtig in die zwei Cluster einordnet, muss die Ähnlichkeit und die Unähnlichkeit zwischen den Dokumenten messbar sein.

Für die Messung der Ähnlichkeit lässt sich die Term-Häufigkeit in den Dokumenten verwenden. Dieser Wert ist als *tf faktor* (Term Frequency) definiert und gibt an, wie gut ein Term den Dokumentinhalt beschreibt. Wenn $\text{freq}_{i,j}$ die Häufigkeit des Terms k_j in einem Dokument d_j ist (Anzahl des Vorkommens) und $\max_{k_l \in d_j}(\text{freq}_{l,j})$ die maximale Häufigkeit eines Terms in dem Dokument d_j ist, so ist die normierte Häufigkeit folgendermaßen definiert:

$$f_{i,j} = \frac{\text{freq}_{i,j}}{\max_{k_l \in d_j}(\text{freq}_{l,j})}$$

Für die Bestimmung der Unähnlichkeit wird die inverse Dokumenthäufigkeit, der sogenannte *idf faktor* (Inverse Document Frequency) verwendet:

$$\text{idf}_i = \log \frac{N}{n_i}$$

wobei N die Anzahl aller Dokumente ist und n_i die Anzahl der Dokumente, in denen der Term k_i vorkommt. Der idf-Faktor kommt zum Einsatz, um die Gewichte von Termen zu reduzieren, die in vielen Dokumenten vorkommen und für die Relevanz unwichtig sind. Anschließend kann man aus diesen Werten die Termgewichte berechnen. Es gibt verschiedene Ansätze für die Berechnung. Eines der bekanntesten *tf - idf*-Schemas für die Termgewichte in den Dokumenten ist:

$$w_{i,j} = f_{i,j} \times \text{idf}_i$$

und für die Termgewichte in der Query:

$$w_{i,q} = 0.5(1 + f_{i,q}) \times idf_i$$

Das Vektormodell ist eines der meistverbreiteten und in der Praxis eingesetzten IR-Verfahren. Im Vergleich zum booleschen Modell ermöglicht das Vektormodell eine partielle Übereinstimmung zwischen der Abfrage und den Dokumenten und liefert demzufolge eine präzisere Antwortmenge, die nach Relevanz sortiert ist. Für dieses Projekt wird Lucene⁵ eingesetzt, eines der bekanntesten Open-Source-IR-Systeme. Die Berechnung der Relevanz der indexierten Dokumente basiert auf dem Vektormodell. Im Folgenden wird ein Beispiel für eine Produktsuche in einer Menge von Dokumenten gezeigt. Gegeben seien fünf Dokumente $d_1 \dots d_5$, deren Text bereits analysiert und in Indextermen konvertiert ist:

d_1 : 2 * Notebook, IBM, 4GB, Intel
 d_2 : Notebook, 2 * Dell, 2 * 2GB
 d_3 : 2 * Notebook, 3 * Sony, 2GB
 d_4 : 2 * IBM, 2 * 2GB, Intel
 d_5 : 2 * Dell, 2 * 2GB, AMD

Außerdem werden zwei Queries angegeben, wobei die eine speziell und die andere etwas allgemeiner ist:

q_1 : Notebook, IBM, 4GB
 d_2 : Notebook, 2GB

Die Tabelle für die Berechnung der Termgewichte wird in Abbildung 4.8 dargestellt. Bei der Kalkulierung der Gewichtungen der Queries wurde die Formel leicht modifiziert mit dem Ziel, die Relevanz der in der Query nicht vorhandenen Terme zu reduzieren:

$$w_{i,q} = 0.5(0.1 + f_{i,q}) \times idf_i$$

Nach der Berechnung der Similarity für jedes Dokument und jede Query ergibt sich die folgende Ergebnistabelle:

	d_1	d_2	d_3	d_4	d_5
q_1	0.850	0.357	0.607	0.589	0.089
q_2	0.500	0.973	1.000	0.101	0.137

Probabilistisches Modell

Die Grundidee des probabilistischen Modells ist die Repräsentation der Relevanz als Wahrscheinlichkeit, dass zu einer Benutzerabfrage eine Menge von Dokumenten für den Benutzer von Interesse ist. Es wird angenommen, dass zu einer bestimmten Abfrage eine Menge von Dokumenten existiert, die genau den Informationsbedürfnissen des Benutzers entspricht (*ideale Antwortmenge*). Die Beschreibung der Dokumente in dieser Menge ist initial nicht bekannt und wird durch Interaktion mit dem User generiert und verfeinert. Ein Dokument wird, genauso wie beim booleschen Modell, als Menge von Indextermen und deren Gewichte beschrieben, die beim probabilistischen

⁵Lucene - <http://lucene.apache.org>

		Notebook	IBM	Dell	Sony	2GB	4GB	Intel	AMD
d_1	tf_{11}	1,00	0,50	0,00	0,00	0,00	0,50	0,50	0,00
	w_{11}	0,22	0,20	0,00	0,00	0,00	0,20	0,20	0,00
d_2	tf_{22}	0,50	0,00	1,00	0,00	1,00	0,00	0,00	0,00
	w_{22}	0,11	0,00	0,40	0,00	0,10	0,00	0,00	0,00
d_3	tf_{33}	0,66	0,00	0,00	1,00	0,33	0,00	0,00	0,00
	w_{33}	0,15	0,00	0,00	0,70	0,03	0,00	0,00	0,00
d_4	tf_{44}	0,00	1,00	0,00	0,00	1,00	0,00	0,50	0,00
	w_{44}	0,00	0,40	0,00	0,00	0,10	0,00	0,20	0,00
d_5	tf_{55}	0,00	0,00	1,00	0,00	1,00	0,00	0,00	0,50
	w_{55}	0,00	0,00	0,40	0,00	0,10	0,00	0,00	0,35
idf_i		0,22	0,40	0,40	0,70	0,10	0,40	0,40	0,70
q_1	tf_{11}	1,00	1,00	0,00	0,00	0,00	1,00	0,00	0,00
	w_{11}	0,12	0,21	0,01	0,02	0,00	0,21	0,01	0,02
q_2	tf_{22}	1,00	0,00	0,00	0,00	1,00	0,00	0,00	0,00
	w_{22}	0,11	0,00	0,00	0,00	0,05	0,00	0,00	0,00

Abbildung 4.8: Beispielstabelle für die Berechnung der tf-idf-Werte.

Modell auch boolesche Werte annehmen. Schrittweise wird versucht, die Beschreibung der idealen Antwortmenge zu berechnen, indem der Benutzer relevante Dokumente aus einer vom System gelieferten Menge selektiert. Durch mehrfache Interaktion mit dem Benutzer wird die Wahrscheinlichkeit, dass bei einer Abfrage q das Dokument d_j relevant ist, maximiert. Die Ähnlichkeit der Query mit einem Dokument wird als der Quotient aus der Wahrscheinlichkeit, dass das Dokument relevant ist, und der Wahrscheinlichkeit, dass das Dokument nicht relevant ist, berechnet:

$$sim(d_j, q) = \frac{P(d_j \text{ relevant zu } q)}{P(d_j \text{ nicht relevant zu } q)}$$

Die Menge der relevanten Dokumente sei mit R gekennzeichnet und \bar{R} sei die Menge der nicht relevanten Dokumente, so lässt sich die Wahrscheinlichkeit, dass d_j für eine Query q relevant ist, mit $P(R|\vec{d}_j)$ darstellen. Die Wahrscheinlichkeit, dass d_j nicht relevant für q ist, ist dann $P(\bar{R}|\vec{d}_j)$. Unter der Verwendung der Wahrscheinlichkeitsrechnung und der Baye'schen Regel kann man die Ähnlichkeit der Query q mit dem Dokument d_j folgendermaßen formalisieren:

$$sim(d_j, q) = \frac{P(R|\vec{d}_j)}{P(\bar{R}|\vec{d}_j)} = \frac{P(\vec{d}_j|R) \times P(R)}{P(\vec{d}_j|\bar{R}) \times P(\bar{R})}$$

wobei $P(\vec{d}_j|R)$ die Wahrscheinlichkeit ist, dass d_j zufällig von R ausgewählt wird und $P(R)$ konstant für alle Dokumente ist ($P(\vec{d}_j|\bar{R})$ und $P(\bar{R})$ analog). $P(\vec{d}_j|R)$ lässt sich wie folgt berechnen:

$$P(\vec{d}_j|R) = \prod_{w_{i,j}=1} P(k_i|R) \prod_{w_{i,j}=0} P(\bar{k}_i|R)$$

Hier ist $P(k_i|R)$ die Wahrscheinlichkeit, dass ein Indexterm k_i in einem relevanten Dokument vorkommt. Zusätzlich wird angenommen, dass die Indexterme unabhängig sind.

Der Vorteil beim probabilistischen Modell ist, dass durch die Wahrscheinlichkeit der Relevanz der Dokumente ein Ranking entsteht und sie sich gemäß ihrer Relevanz anordnen lassen. Das ist

beim booleschen Modell nicht gegeben. Als Nachteil wird das Fehlen der Gewichte bei den Index-terminen gesehen. Außerdem ist die Notwendigkeit einer initialen Antwortmenge und die Annahme der Unabhängigkeit der Indexterme problematisch. Aus diesen Gründen und wegen der Komplexität des Verfahrens konnte sich das probabilistische Modell in der Praxis nicht durchsetzen. Das Vektormodell dagegen ist am weitesten verbreitet. Im Vergleich zum probabilistischen Modell ist es einfacher und performanter, außerdem ist es nicht so streng und erlaubt durch die Gewichte der Indexterme ein partielles Matching.

4.4.5 Retrieval Evaluation

Das klassische Vorgehen, um die Güte der IR-Systeme zu messen, ist der Vergleich zwischen den vom System gelieferten Ergebnissen und denen, die von einem Experten bereitgestellt werden. Dabei benutzt man eine Testmenge von Dokumenten, eine Testmenge von Abfragen und die wirklich relevanten Dokumente für jede Abfrage. Für jede Abfrage werden die vom System gelieferten Ergebnisse mit den manuell erstellten Ergebnissen ausgewertet und ihnen gegenübergestellt.

Die am häufigsten verwendeten Maße für die Evaluierung eines IR-Algorithmus sind *Precision* und *Recall*. R sei die Menge der relevanten Dokumente und A die Menge der vom IR-System gelieferten Dokumente. Mit Ra wird der Schnitt der beiden Mengen R und A gekennzeichnet, der die wirklich relevanten Dokumente in der Antwortmenge A repräsentiert. *Recall* und *Precision* werden folgendermaßen definiert:

$$\text{Recall} = \frac{|Ra|}{|R|}$$

$$\text{Precision} = \frac{|Ra|}{|A|}$$

wobei $|Ra|$, $|R|$ und $|A|$ die Anzahl der Dokumente in den entsprechenden Mengen sind. *Recall* ist der Anteil derjenigen Dokumente, die vom System richtig ausgewählt wurden (Fig. 4.9). *Precision* misst, wie präzise die Ergebnisse sind, also den Anteil der wirklich relevanten von allen gefundenen Dokumenten.

Die Recall- und Precision-Werte lassen sich auch für einzelne Dokumente berechnen. Normalerweise werden die Dokumente in der Antwortmenge nach Relevanz absteigend sortiert. Nachdem Recall und Precision für alle Dokumente in der Antwortmenge der Reihe nach berechnet wurden, kann man die Werte graphisch einander gegenüberstellen. So entsteht eine Recall-Precision-Kurve für eine Abfrage. Auf die y-Achse kommen die Precision-Werte und auf die x-Achse die Recall-Werte. Zum Beispiel sei R_q die Menge mit den relevanten Dokumenten und A_q die Antwortmenge:

$$R_q = \{d_3, d_5, d_9, d_{25}, d_{39}, d_{44}, d_{56}, d_{71}, d_{89}, d_{123}\}$$

$$A_q = \{d_{123}, d_{84}, d_{56}, d_6, d_8, d_9, d_{511}, d_{129}, d_{187}, d_{25}, d_{38}, d_{48}, d_{250}, d_{113}, d_3\}$$

Das System hat 5 relevante Dokumente zurückgeliefert, und zwar $d_{123}, d_{56}, d_9, d_{25}, d_3$. Bei dem ersten Dokument d_{123} ist die Precision 100% und die Recall 10%. Beim zweiten relevanten Dokument d_{56} sind die Werte 66% Precision (2 Dokumente von 3 gefundenen sind relevant) gegenüber 20% Recall (2 Dokumente von den 10 relevanten sind entdeckt worden). Die Berechnung verläuft für alle weiteren relevanten Dokumente analog, wobei für jedes Dokument aus der Menge eine

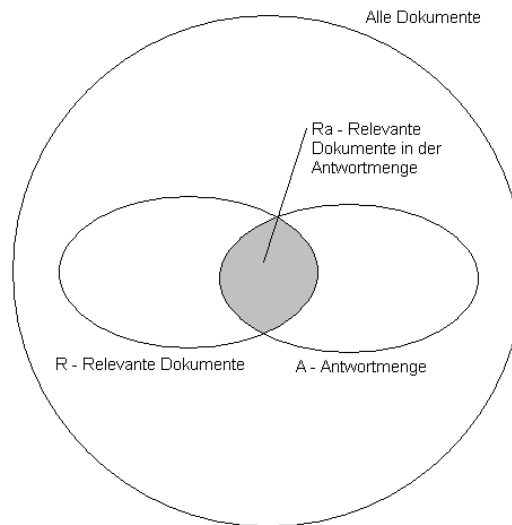


Abbildung 4.9: Graphische Repräsentation der Menge der richtig erkannten Dokumente (R_a)

Recall-Ebene (*Standard Recall Level*) existiert. Die Ergebnisse werden graphisch in Abbildung 4.10 präsentiert.

Beim Auswerten der Güte eines IR-Algorithmus ist es notwendig, einen mittleren Precision/Recall-Wert zu berechnen. In diesem Fall werden alle Queries ausgewertet und anschließend wird ein Precision-Mittelwert für alle Recall-Levels ermittelt:

$$\bar{P}(r) = \sum_{i=1}^{N_q} \frac{P_i(r)}{N_q}$$

$\bar{P}(r)$ ist die mittlere Precision bei Level r , N_q ist die Anzahl der Abfragen und $P_i(r)$ ist die Precision bei Level r und Query i . Die Recall-Levels können vom *Standard* abweichen, zum Beispiel in dem Fall, dass für bestimmte Queries weniger Dokumente zurückgegeben werden als die Anzahl der *Standard Recall Levels*. In diesem Fall werden die Precision-Werte für die fehlenden Recall-Levels interpoliert.

4.4.6 Queries

Der Benutzer spezifiziert seine Informationsbedürfnisse mittels einer Abfrage (Query). Je nach IR-System existieren verschiedene Ansätze für den Aufbau einer Query. Die einfachsten davon sind keyword-basiert. Darüber hinaus kommen auch boolesche und Kontext-Queries zum Einsatz. Zusätzlich können Pattern-Matching-Techniken benutzt werden, um die Queries flexibler und intelligenter zu machen.

Keyword queries bestehen aus ein oder mehreren Wörtern, und bei der Suche werden alle Do-

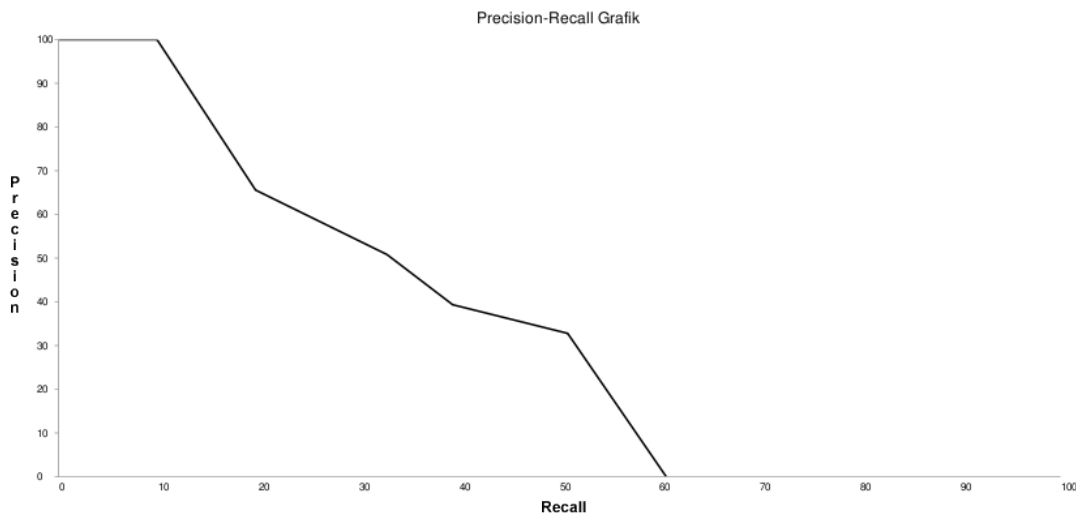


Abbildung 4.10: Beispiel für Precision-Recall-Kurve.

kumente zurückgegeben die eine oder mehrere davon beinhalten. Beim Vektormodell lassen sich die Terme je nach Wichtigkeit anders gewichten.

Phrase queries sind sogenannte Kontext-Queries, die die Suche in einem Kontext erlauben. So werden Terme, die gleichzeitig mit bestimmten anderen Wörtern im Text auftauchen, als relevant berücksichtigt. Die *Phrase-Queries* sind die einfachste Form von solchen Queries. Sie bestehen aus einer Sequenz von Termen, die üblicherweise mit Anführungszeichen begrenzt werden und in der gleichen Reihenfolge auch im Text vorkommen sollen. Trennzeichen und Stoppwörter werden besonders berücksichtigt.

Proximity queries sind auch Kontext-Queries, allerdings weniger streng. Der Benutzer kann die maximale Distanz zwischen den Wörtern einstellen, die entweder in Wörtern oder Buchstaben gemessen wird. Die Reihenfolge der Terme im Text spielt keine Rolle. Zum Beispiel sucht die Query "CPU Speicher"~4 nach Dokumenten, in denen die beide Terme zusammen auftauchen und maximal vier Wörter voneinander entfernt sind.

Boolesche queries bestehen aus Termen und den booleschen Operatoren *AND*, *OR* und *NOT*. Der *AND*-Operator selektiert nur Dokumente, in denen alle Terme im Text vorkommen. Der *OR*-Operator erfordert, dass mindestens einer der Terme im Dokument existiert. Mit dem Operator *NOT* kann man eine Negation ausdrücken und Queries wie CPU NOT Intel bilden. Komplexe Abfragen lassen sich durch das Bilden von Subqueries erstellen, die mit Klammern gruppiert werden, zum Beispiel CPU AND (Intel OR AMD), was nur Dokumente selektiert, in denen CPUs der Marke Intel oder AMD vorkommen. Weil die klassischen booleschen Queries kein Ranking erlauben, existieren auch *Fuzzy-boolesche* Queries, wo die Terme und Operatoren teilweise ignoriert werden.

Regular Expressions sind reguläre Ausdrücke, die mit Hilfe von bestimmten syntaktischen Regeln ein Suchmuster beschreiben [Stu03]. Ihren Ursprung haben sie in der Automaten-theorie und den formalen Sprachen. Die Idee besteht darin, einen Abschnitt im Text zu

finden, der mit dem von den RegExp beschriebenen Text übereinstimmt. Diese Expressions bestehen aus einer Kombination aus normalen und Metazeichen. Die normalen Zeichen müssen im Text vorkommen, während die Metazeichen Regeln beschreiben, wie Position, Anzahl und Art der Zeichen. Eine der einfachsten und meistverwendeten Form von RegExp ist die sogenannte Wildcard, bei der jedes Zeichen beschrieben und üblicherweise mit dem Metazeichen „?“ markiert wird. Um beliebig lange Wildcards zu ermöglichen, benutzt man „*“, zum Beispiel „*.txt“.

Fuzzy queries erlauben die Suche nach ähnlichen Termen im Text. Die Idee ist, eventuelle Unterschiede in der Schreibweise zu berücksichtigen, um Terme mit Schreibfehlern in den Dokumenten zu erlauben. Die Ähnlichkeitsmessung zwischen zwei Termen basiert auf der *Levenstein-Distanz* (auch „Edit Distance“), wobei es auch andere Ansätze gibt. Die „Edit“-Distanz wird anhand der Anzahl der notwendigen „Edit“-Operationen gemessen, um beide Strings identisch zu machen. Solche Operationen sind Einfügen, Entfernen und Ersetzen von Zeichen.

Bei manchen IR-Systemen lassen sich unterschiedliche Query-Typen kombinieren, um komplexe Abfragen zu erstellen und so die Antwortmenge zu präzisieren.

4.5 Information Extraction

Genauso wie Information Retrieval beschäftigt sich Information Extraction (IE) mit der Suche nach relevanten Informationen in großen Textmengen. IE geht sogar einen Schritt weiter. Während sich IR damit beschäftigt, nur die relevanten Textdokumente zu finden, geht es bei der IE um die Extraktion von bestimmten Informationen aus den relevanten Dokumenten [AFG03]. Die Ausgangsdaten für den IE-Prozess sind hauptsächlich in Form von *unstrukturierten* oder *halb-strukturierten* Dokumenten gegeben. Die Aufgabe eines IE-Systems ist es, bestimmte Fakten, die in den Ausgangsdaten enthalten sind, aufzufinden und in eine strukturierte Form zu überführen und evtl. zu speichern. Die Dokumente lassen sich gemäß ihrer Struktur in drei Gruppen aufteilen - *strukturiert*, *unstrukturiert* und *halb-strukturiert*. Als strukturiert gelten Dokumente, bei denen die Positionen der Fakten von Anfang an bekannt sind. Ein Beispiel hierfür ist eine Tabelle in einer Datenbank, bei der die Information in Spalten eingeteilt ist, oder eine XML-Datei, bei der die Daten gesondert mit Tags markiert sind. Bei halb-strukturierten Dokumenten ist bekannt, dass sie bestimmte Daten und Fakten beinhalten, aber nicht, wo die genaue Position im Text zu finden ist. Das ist zum Beispiel der Fall bei einer Produktseite in einem Webshop, die den Produktnamen, die Produktbeschreibung und den Preis beinhaltet, wobei nicht bekannt ist, wo auf der Seite diese Angaben stehen. Unstrukturiert sind alle übrigen Dokumente, die nicht als strukturiert oder halb-strukturiert eingestuft werden können. Das sind Texte in natürlichen Sprachen, für die initial keine Informationen vorliegen. Dies ist zum Beispiel der Fall bei den meisten Webseiten im Internet - Nachrichten, Briefe usw.

4.5.1 Informationsextrahierung aus unstrukturierten Dokumenten

Bei der Informationsgewinnung aus unstrukturierten und halb-strukturierten Dokumenten kommen zum Teil unterschiedliche Techniken zum Einsatz. Nichtsdestotrotz existieren klassische

Grundtechniken, die bei jedem IE-Prozess angewendet werden. Der normale Verlauf eines solchen Prozesses besteht aus mehreren Phasen (Fig. 4.11). Bevor das System einzelne Fakten extrahieren kann, wird eine *lexikalische* Analyse des Dokuments durchgeführt [Gri97]. Nachdem die einzelnen Fakten vorhanden sind, werden durch Inferenz neue Fakten abgeleitet oder die bestehenden erweitert (*Diskursanalyse*). Schließlich werden die gesammelten Fakten in einem vordefinierten Ausgabeformat geschrieben, was als *Template Generierung* bezeichnet wird.

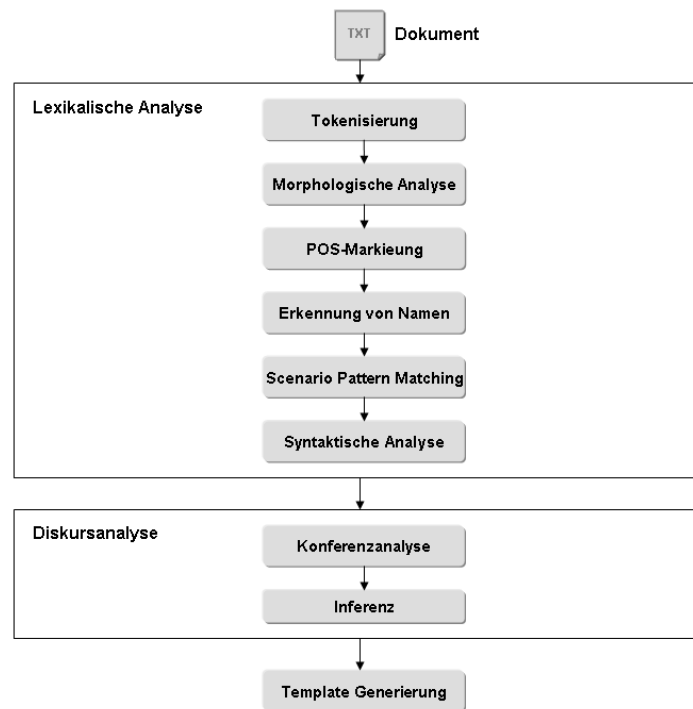


Abbildung 4.11: Struktur des klassischen IE-Prozesses

Die einzelnen Fakten werden im Text mit Hilfe der Mustererkennung erkannt. Weil die Ausgangsdaten normalerweise in natürlicher Sprache vorliegen, ist es infolge der Komplexität der Sprachen nicht üblich, die Mustererkennung direkt anzuwenden. Eine Reihe von Techniken für die Analyse und Strukturierung werden schrittweise im Text ausgeführt. Einige davon stimmen mit den Textoperationen überein, die im Unterkapitel 4.4.2 bereits erläutert wurden; sie werden im Folgenden nur kurz beschrieben:

Tokenisierung Der Text wird in Sätze und einzelne Wörter zerlegt. Für die Worttrennung werden meistens die Leerzeichen und verschiedene Interpunktionszeichen verwendet, wie Punkt oder Fragezeichen für die Satztrennung. Wenn dieses Verfahren problemlos für die meisten europäischen Sprachen funktioniert, müssen Sprachen mit komplexer Wort- und Satzbildung besonders berücksichtigt werden. Die Technik wird detaillierter im Unterkapitel 4.4.2 beschrieben.

Morphologische Analyse Die einzelnen Wörter werden analysiert, um verschiedene Wortformen zu erkennen, die normalerweise durch Flexion entstehen. Das sind Wörter, die sich

von ihrem Stamm syntaktisch unterscheiden, aber semantisch die gleiche Bedeutung haben. Ein Beispiel dafür ist die Veränderung der Verben in Abhängigkeit von Tempus oder Modus, wie *laufen, lief, läuft etc.*. Bei einigen Sprachen lässt sich diese Analyse einfach durch Berücksichtigung von Präfixen oder Suffixen durchführen, bei anderen dagegen (*Deutsch, Russisch, Finnisch*) wird zusätzlich von einem Wörterbuch Gebrauch gemacht, um die Wörter zu *matchen*. Schließlich werden zusammengesetzte Wörter in ihre Einzelteile zerlegt.

Part-Of-Speech-Markierung Die Wörter im Text werden mit der entsprechenden Wortart markiert, wie zum Beispiel Verb, Nomen, Adjektiv etc.

Erkennung von Namen Mit Hilfe dieser Technik werden verschiedene Namen (Personen, Firmen, Orte) und andere Elemente wie Währungen, Datum und Zeitangaben erkannt und markiert. Typischerweise kann in diesem Fall kein Lexikon angewendet werden, weil es unzählige Variationen geben kann. Besonders bei Datum und Zahlformaten lassen sich reguläre Expressions erstellen, die brauchbare Ergebnisse liefern. Auch Namen kann man mit einer Menge von *RegExp* erkennen, zum Beispiel anhand der Anrede, häufiger Vornamen, von Wörtern mit Großbuchstaben oder Suffixen bei Firmen, wie *Google Inc.* oder *Siemens AG*. Außer handcodierten Mustern existieren auch Ansätze, die auf dem maschinellen Lernen und dem *Hidden-Markov-Modell* basieren [BMSW98].

Syntaktische Analyse Bei der Extrahierung von Fakten können Informationen über die syntaktische Struktur der Sätze von Vorteil sein. Besonders Nomen können dazu beitragen, die als Hauptwörter in einem Satz angesehen werden können. Zusätzlich können durch die Kennzeichnung von verbalen Phrasen funktionale Relationen zwischen ihnen erkannt werden. Als Ergebnis generiert das System entsprechende Markups im Text:

[*noun-n1* Volkswagen] [*verb-v1* übernimmt] [*noun-n2* Porsche], [*noun-n3* der bisherige Produktionschef Michael Macht] [*verb-v2* ersetzt] [*noun-n4* Wiedeking]

Der einfachste Ansatz ist das *Full Parsing*, bei dem anhand einer formalen Grammatik die komplette grammatische Struktur des Satzes erkannt wird. Der Nachteil dabei ist die schlechte Performance und die hohe Fehlerquote. Eine bessere Performance bietet dagegen das *Shallow Parsing*. Dieses Verfahren basiert auf Grammatiken mit endlich vielen Zuständen (*finite state grammars*), und im Gegensatz zum *Full Parsing*, bei dem für jeden Satz ein kompletter Parse-Baum erstellt wird, fokussiert es sich hauptsächlich auf die Nomen und Verb-Konstrukte, da sie auch die meiste relevante Information liefern [AFG03].

Scenario Pattern Matching Bei diesem Schritt werden die Ereignisse und die Beziehungen der Fakten, die für das IE-System relevant sind, erkannt. Aus dem Text „*Volkswagen übernimmt Porsche, der bisherige Produktionschef Michael Macht ersetzt Wiedeking*“ können zum Beispiel die zwei Events *Firmenübernahme* und *Positionswechsel* erkannt werden.

Im nächsten Schritt des IE-Prozesses wird die *Konferenzanalyse* durchgeführt. Sehr oft benutzen Autoren unterschiedliche Benennungen von Objekten (zum Beispiel Synonyme), um Wiederholungen im Text zu vermeiden. Das Ziel dieser Analyse ist, solche Elemente im Text aufzuspüren und entsprechend zu markieren. Dabei kann das IE-System den Text „interpretieren“ und die semantisch identischen Elemente bei der Relationenbildung berücksichtigen. Zum Beispiel *Porsche, das Unternehmen, der Sportwagen-Hersteller, der Deutsche Autobauer* usw.

In einigen Texten können Informationen über bestimmte Ereignisse auf mehrere Stellen im ganzen Dokument verteilt sein. Im letzten Schritt des klassischen IE-Prozesses wird versucht, diese Informationen zu kombinieren. Oft lassen sich aus den bestehenden Fakten durch Inferenzregeln weitere relevante Informationen gewinnen. Zum Beispiel kann man aus dem folgenden Text "*Bill Gates war CEO von Microsoft. Nachfolger wird Steve Ballmer*" schlussfolgern, dass 1) Steve Ballmer CEO wird und 2) Bill Gates zurückgetreten ist. Die Schlussfolgerungsregel kann folgendermaßen aussehen:

$$\text{Chef}(\text{Person}X, \text{Firma}Y) \& \text{Nachfolger}(\text{Person}Z, \text{Person}X) \Rightarrow \text{Chef}(\text{Person}Z, \text{Firma}Y)$$

4.5.2 Information Extraction in semi-strukturierten Webdokumenten

Die relevanten Informationen im World Wide Web sind meistens auf unzählige Webseiten verteilt, was die Informationsbeschaffung für den Benutzer sehr mühsam und unübersichtlich macht. Deswegen ist es von größtem Interesse, solche Informationen automatisch aus den Inhalten der Webseiten zu extrahieren. Die im Web verfügbaren Daten werden hauptsächlich individuell gestaltet und haben eine uneinheitliche Quellcode-Struktur. Solche Daten werden *semistrukturiert* genannt und charakterisieren sich durch eine lose und trübe Struktur. Das automatische Abfragen von Informationen ist daher eine komplizierte Aufgabe und erfordert das genaue Kennen der impliziten Dokumentenstruktur.

Die Information Extraction in semi-strukturierten Daten erfordert eine spezielle Modellierung, bei der der Aufbau von Websites, die innere Struktur und die Inhalte der Webseiten berücksichtigt werden. Diese Seiten können sich innerhalb einer Website oder verschiedener Domains befinden. Die innere Struktur der Seiten wird durch hierarchische HTML-Tags repräsentiert, die bei jeder Webseite anders sind. Einige Ansätze basieren auf dem Object Exchange Model (OEM) [Abi97], bei dem die Daten als ein Graph mit beschrifteten Kanten modelliert werden. Die Entitäten (Objekte) werden als Knoten repräsentiert und die Kanten als die Beziehungen zwischen den Objekten. Die Objekte können sowohl einfache als auch komplexe Datentypen beinhalten. Die einfachen können Strings, Zahlen, HTML und Media, wie Bilder und Video, beinhalten. Die komplexen bestehen aus Objektreferenzen, die im Graph als Kanten repräsentiert werden. So können die Daten aus dem Web hierarchisch in einer baumartigen Struktur dargestellt werden. Anhand des OEM-Graph in Abbildung 4.12 wird die uneinheitliche Struktur der Daten sofort sichtbar. Bei jedem der modellierten Restaurant-Objekte ist die Information unterschiedlich strukturiert. Restaurant 1 hat eine detaillierte Adresse mit klaren Angaben zu Stadt, PLZ und Straße. *Restaurant 2* hat zwei grobe Adressen, und zu *Restaurant 3* gibt es gar keine Adressdaten, jedoch Angaben über die Preise.

Die bekanntesten Techniken für das Abfragen von halb-strukturierten Daten sind die Pfadausdrücke und die Mustererkennung. Damit lässt sich in der hierarchischen Struktur des OEM navigieren und bestimmte Objekte lassen sich adressieren. In dem oben genannten Beispiel können mit einem Pfadausdruck wie `"/guide/restaurant/address/zipcode/"` alle PLZ der vorhandenen Restaurants selektiert werden. Die Pfadausdrücke stellen ein Pfadmuster dar, das in dem Datengraph erkannt werden kann. Solche Methoden funktionieren problemlos mit gut strukturierten Daten, weil die Struktur von Objekten gleichen Typs immer identisch ist. Für Daten mit uneinheitlicher Struktur können bessere Ergebnisse erzielt werden, indem der Ansatz mit Wildcards oder regulären Ausdrücken erweitert wird. Wenn wir mit „*“ beliebige Knoten an bestimmter Stelle erlauben, können mit dem Ausdruck `"/guide/*/address/*"` die Adressen von

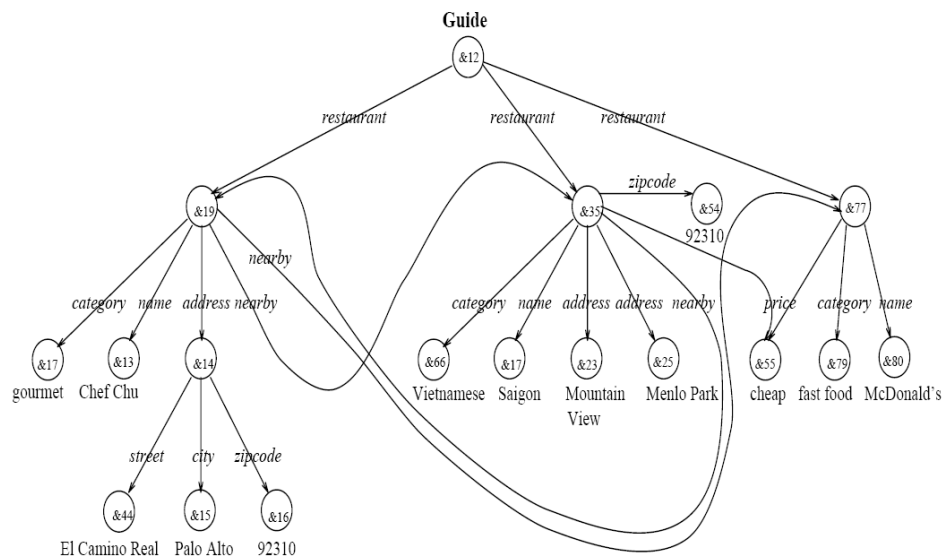


Abbildung 4.12: Struktur einer Webseite für Restaurants, repräsentiert als OEM-Graph (Quelle: [Abi97])

allen Objekten in der Datensammlung abgefragt werden.

4.6 Information Extraction in Webshops

Die vorliegende Diplomarbeit hat zum Ziel, ein Verfahren vorzustellen, das für die Informationsgewinnung aus Webshop-Seiten geeignet ist. Die Hauptaufgabe dabei ist, Daten aus Webseiten zu gewinnen, die, wie bereits in Unterkapitel 4.5.2 erwähnt, zu der Menge der semi-strukturierten Dokumente gehören. Deren Struktur wird durch hierarchische HTML-Tags repräsentiert, die bei jeder Webseite anders sind. Dennoch tendieren Webseiten, die Informationen der gleichen Art liefern (wie zum Beispiel Produktinformationen), dazu, eine ähnliche Struktur aufzuweisen. Theoretisch bilden solche Seiten eine Klasse, und man kann dort einfacher bestimmte Informationen abfragen. In den Onlineshops bilden unter anderem die Produktseiten eine Klasse, in der die Daten jeweils zu einem Produkt präsentiert werden samt detaillierten Produktinformationen und der Kaufmöglichkeit (*Zum Warenkorb*). Der IE-Prozess in Webshops besteht in diesem Fall aus zwei Hauptschritten. Als erstes werden die Webseiten nach Produkt- und Nich-Produkt-Seiten klassifiziert und anschließend werden die Informationen anhand der bereits bekannten Struktur entnommen. Schließlich können die Daten in eine relationale Datenbank geschrieben und weiterverarbeitet werden.

4.6.1 Struktur der Produktseiten

Die Webseiten bestehen aus Daten in diversen Formaten (wie Text, Bilder und *Rich Media*) und Strukturdaten in Form von HTML-Tags. Die meisten Tags bestehen aus einem Start- und einem

End-Tag, die Daten oder weitere Tags einkapseln. Auf diesem Wege entsteht ein Tag-Baum, der die hierarchische Struktur des Dokuments repräsentiert.

In der HTML-Spezifikation existiert eine Vielzahl an Tags, die sich grob in folgende Gruppen einordnen lassen:

- Struktur-Tags - bestimmen die räumliche Anordnung der Seite und schließen meistens weitere Tags mit ein. Ein solcher Tag ist zum Beispiel `<table>`, der die Daten in einer tabellarischen Form anordnet. Oder `` und ``, die die Daten hierarchisch aufzählen. Einige Seiten werden alternativ mit `<div>`-Tags strukturiert, wobei die Positionierung normalerweise durch externe Stylesheets (CSS) gemacht wird. Nichtsdestotrotz lässt sich auch mit `<div>`-Tags eine tabellarische und hierarchische Strukturierung der Daten erreichen.
- Daten-Tags - sie umfassen die eigentlichen Inhalte einer Seite, wie Text und Graphiken, und haben normalerweise keine weiteren Tags. In dem Tag-Baum repräsentieren sie die Blätter, die für die Informationsgewinnung von großem Interesse sind. Das sind Tags wie: `<h1>`,...,`<h6>`,`<p>`,`` und `` .
- Hyperlinks - sie verweisen auf andere Webdokumente und werden mit dem Tag `<a>` markiert. Die mit dem Tag eingeschlossene Texte können auch für den IE-Vorgang relevant sein.
- Formularelemente - das sind Elemente, die für die Interaktivität der Webseiten zuständig sind, wie zum Beispiel *Eingabefelder*, *Comboboxes*, *Checkboxes* usw. Solche Elemente können ebenso keine weiteren Tags beinhalten und werden im Tag-Baum als Blätter repräsentiert.
- Formatierungstags - dienen hauptsächlich der Dekoration und bestimmen meistens Abstände, Schriftgrößen und Farben. Sie können keine Daten beinhalten, liefern nur wenig nützliche Informationen und sind demzufolge für IE uninteressant. Solche Tags sind `<hr>`, `<style>`, `` etc.

Zusätzlich können die Daten-Tags mittels Struktur-Tags thematisch gruppiert werden. Außerdem haben diese Tags eine nach ihrer Wichtigkeit festgelegte Reihenfolge: $H1 > H2 > \dots > H6 > P$. Die H-Tags dienen der Darstellung von Überschriften und Schlagzeilen und werden normalerweise optisch hervorgehoben. Die P-Tags schließen längere Textabschnitte ein und können für detaillierte Informationen, z. B. Produktbeschreibungen, benutzt werden:

```
<html>
<body
...
<div style="color:#00FF00">
  <h3>This is a product name</h3>
  <p>This is a long product description</p>
</div>
...
</body>
</html>
```

Die wichtigsten Tags, die bei dem IE-Prozess unbedingt berücksichtigt werden sollen, sind die Struktur- und Daten-Tags. Im Allgemeinen wird das Dokument durch die Struktur-Tags in separate Regionen unterteilt, die bestimmte Informationen aus einem Bereich beinhalten können. Sie können auch Daten-Tags einschließen, die an sich atomare Datenbestandteile beinhalten. Aus einem anderen Blickwinkel kann man die Daten-Tags als Attribute betrachten, deren Wert einer der gesuchten Informationen entspricht. Die Seiten aus der Klasse der Produktseiten besitzen folgende Attribute: *Produktbezeichnung*, *Produktbeschreibung*, *Preis*, *Produktabbildung*, *Verfügbarkeit* und *eventuell Versandkosten*. Die Daten-Tags sind in der Abbildung 4.13 grau markiert, links im Baum ist die Kategorienavigation und rechts sind die gesuchten Attribute.

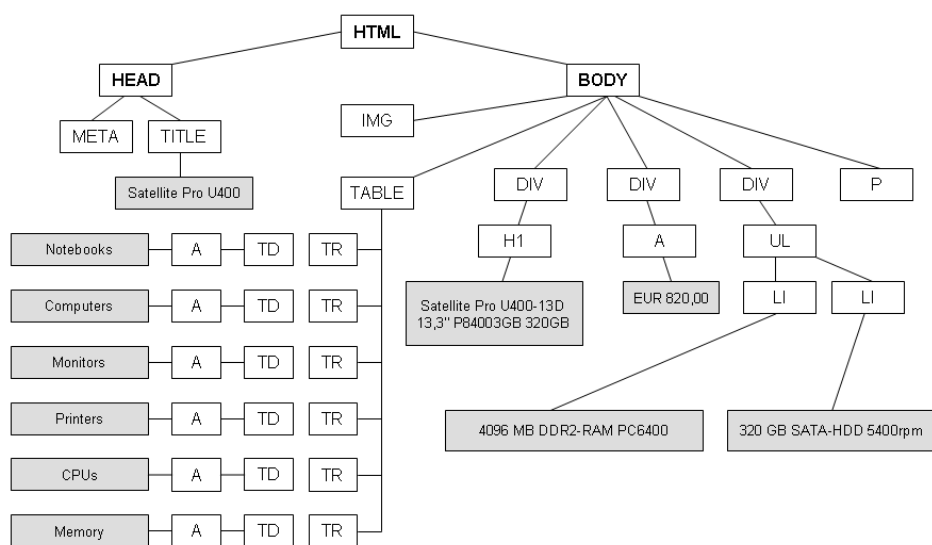


Abbildung 4.13: Baumdarstellung einer Produktseite

4.6.2 Klassifizierung von Webseiten

Webseiten, die sich in einer Klasse befinden, besitzen eine ähnliche Struktur. In Abhängigkeit von dem Typ der Information haben Seiten in einer Klasse bestimmte *Attribute*, die als Eigenschaften in einer relationalen Datenbank betrachtet werden können. Die Attribute können auch dazu dienen, eine Klasse von Webseiten zu beschreiben. Die Klasse der Produktseiten charakterisiert sich zum Beispiel durch Attribute wie *Produktbezeichnung*, *Produktbeschreibung* und *Preis*.

Ferner können Seiten aus einer Klasse von dem gleichen oder von unterschiedlichen Onlineshops stammen. Weil die meisten Shops ihre Produktseiten aus einer Datenbank generieren und die HTML-Seite dazu als Vorlage dient, kann man davon ausgehen, dass die Webseiten

in einer Domain eine fast identische Quellcode-Struktur besitzen. Die Seiten aus verschiedenen Shops haben zwar die gleichen Attribute, aber auf Grund des unterschiedlichen Layouts kann die Quellcode-Struktur komplett anders sein. Das macht das Lokalisieren der Attribute im Quellcode und damit die Klassifizierung der Seiten zu einer sehr komplexen Aufgabe. Aus diesem Grund wird mit dem vorgestellten Ansatz ein alternativer Weg für die Seitenklassifizierung verfolgt. Als Grundlage dafür dient eine Trainingsmenge von Beispiel-Produktseiten aus demselben Webshop, vorausgesetzt die Dokumente innerhalb einer Domain eine ähnliche HTML-Struktur haben.

Trainingsmenge generieren

Damit die Webseiten in einem Onlineshop nach Produkt- und Nicht-Produktseiten richtig klassifiziert werden können, wird eine Menge mit positiven Beispielen für Produktseiten gebraucht, die von dem zu untersuchenden Webshop stammen. Die sogenannte Trainingsmenge wird üblicherweise manuell zusammengestellt. Hierzu muss der Benutzer selbst ein paar Produktseiten samt deren Adresse (URL) und Attribute markieren. Hier wird ein alternatives Verfahren präsentiert, das die Trainingsmenge für einen neuen Shop mit Hilfe von IR-Techniken automatisch generiert. Im Hintergrund steht eine relationale Produktdatenbank, die anfangs manuell aufgebaut wird und durch Extrahieren von bereits untersuchten Webshops erweitert wird. Diese Datenbank beinhaltet die wichtigsten Produktdaten, wie Bezeichnung, Beschreibung und Preis, und dient als Datenquelle für Abfragen in einem neuen und unbekanntem Onlinekatalog. Unser Ansatz basiert auf der Annahme, dass in einem zu extrahierenden Webshop Produkte existieren, die in der Datenbank bereits aufgenommen worden sind. Andernfalls kann nur eine manuelle Zusammenstellung der Trainingsmenge funktionieren.

Als erster Schritt wird der Textinhalt der Seiten indexiert und für Suchanfragen optimiert. Anschließend werden mehrere Queries im Index ausgeführt, die nach bereits bekannten Produkten anhand von Produktname und -beschreibung suchen. Weil die Produktdaten und deren Repräsentation von Shop zu Shop abweichen, müssen unterschiedliche Keywords und deren Reihenfolge im Dokument berücksichtigt werden. Aus diesem Grund ist ein IR-System am besten geeignet, das auf dem Vektorraum-Modell basiert. Anhand des vom IR-System berechneten *Score* für die Dokumente können die relevantesten Seiten selektiert werden. Seiten, die einen bestimmten Score nicht erreicht haben, werden ausgefiltert. Ein Nachteil dabei ist, dass mit diesem Verfahren eventuell auch Nicht-Produktseiten selektiert werden. Fakt ist, dass in einem Onlineshopsystem die Daten eines Produktes nicht nur auf den Produktseiten vorkommen können, sondern unter Umständen auch auf viele andere Seiten verteilt sein, wie Produktaufstellungen, Seiten, die Top-Angebote und verwandte Artikel anzeigen usw. Natürlich finden sich die meisten und die detailliertesten Informationen über ein Produkt auf der Produktseite, die auch im Normalfall einen höheren *Score* bekommt. Damit der IE-Prozess robuster wird und nicht nur das Dokument mit dem besten Score berücksichtigt, können die Top-X-Dokumente in der Trainingsmenge aufgenommen werden. Bei der Klassifizierung kann eine zusätzliche Analyse durchgeführt werden, und die Nicht-Produktseiten können erkannt und ausgefiltert werden.

Mustererkennung in Produktseiten

In diesem Absatz wird ein Verfahren vorgestellt, das mit Hilfe einer Trainingsmenge von Webdokumenten aus der Klasse der Produktseiten alle restlichen Webseiten in dem betreffenden Web-

shop klassifiziert. Wie bereits erwähnt, sind die Seiten innerhalb einer Klasse durch eine ähnliche Struktur und gleiche Attribute charakterisiert. Als semi-strukturierte Dokumente beinhalten die Webseiten einer Klasse die gesuchten Attribute nur implizit. Sie lassen sich ohne ein vorgegebenes Muster im Quellcode nicht identifizieren. Diese Muster können zum Beispiel bestimmte Beschriftungen der Attribute sein. Das ist oft der Fall, wenn die Daten in einer tabellarischen Form repräsentiert werden, bei der die Labels den Spaltenbeschriftungen gleichen. Die Methode erweist sich leider als unzuverlässig: erstens haben nicht alle Webseiten eine tabellarische Struktur und zweitens können die Labels in den verschiedenen Domains zu stark voneinander abweichen.

Wir nutzen den Umstand, dass die Struktur des Quellcodes der Produktseiten innerhalb eines Onlineshops gleich bleibt, und erstellen ein Muster für die Attribute anhand von deren genauer Position im Quellcode. Wenn die hierarchische Quellcodestruktur einer Seite als ein DOM-Baum (Document Object Model) modelliert wird, kann die Position eines Attributes als ein Pfadausdruck betrachtet werden. Der Baum hat für jeden HTML-Tag einen Knoten und die Kanten bilden die Dokumentenhierarchie ab. So beschreibt der Pfadausdruck den Pfad von der Wurzel (Tag <html>) zum Zielknoten. Für das Selektieren der Knoten im DOM-Baum und das Abfragen von Attributinhalt ist die XPath-Sprache gut geeignet. Darüber hinaus unterstützt XPath Wildcards und erlaubt damit das Bilden von generalisierten Pfadausdrücken, die Attribute in einer Vielzahl von Seiten mit leicht abweichender Struktur adressiert. Solche Pfadausdrücke können auch für die Mustererkennung von Attributen in Produktseiten dienen. Das folgende Beispiel zeigt eine XPath-Abfrage, die das Attribut *Produktname* auf einer Produktseite selektiert:

```
/html/body/table[1]/tr[3]/td[1]/table[1]/tr/td[2]/h1
```

Das Ziel unseres Ansatzes für die Klassifizierung von Webshop-Seiten ist, für jedes Attribut einen XPath-Ausdruck zu generieren und damit die vorhandenen Webseiten im Shop nach diesem Muster zu durchsuchen. Falls alle Attribute (wie *Produktname*, *Produktpreis*, *Produktbeschreibung* und *Produktabbildung*) erkannt werden können, liegt die Seite mit großer Wahrscheinlichkeit in der Klasse der Produktseiten. Die Generierung der XPath-Ausdrücke wird anhand der Trainingsmenge von Produktseiten durchgeführt und geschieht in zwei Schritten. Als Erstes wird die genaue Position der Attribute im Sourcecode anhand von deren Werten ermittelt und daraus der XPath erzeugt, der genau diesen Knoten im DOM markiert. Im zweiten Schritt werden die gelieferten Ausdrücke für jedes Attribut generalisiert, so dass möglichst viele Produktseiten mit diesem Muster erkannt werden.

Für den ersten Schritt wird für jedes Attribut ein Wert benötigt, damit der Knoten (bzw. Tag) im Quellcode lokalisiert wird. Für die Trainingsmenge sind die Attributwerte leider nicht explizit angegeben und müssen zuerst auf der Seite lokalisiert werden. Dabei muss berücksichtigt werden, dass die Attributwerte, die für die Generierung der Trainingsmenge benutzt werden, von den tatsächlichen Werten in der Trainingsmenge leicht abweichen können. Der Grund liegt darin, dass unterschiedliche Webshops auch leicht abweichende Attributswerte verwenden, was für das Vektormodel des IR-Systems, mit dem die Beispielseiten gewonnen werden, kein Problem darstellt. Das *Matching* wird mit Hilfe von speziellen *Regular Expressions* gemacht, die für jedes Attribut gesondert angepasst werden:

- Attribut *Produktname* - Hier wird die bekannte Produktbezeichnung in Einzelwörter zerlegt, dabei wird eine Menge von Separator-Symbolen (*Leerzeichen*, *Strich*, *Komma*, *Punkt*,

Querstrich, Unterstrich etc.) berücksichtigt. Für jede Permutation der gebildeten Menge von Einzelwörtern wird eine Regular Expression erzeugt und auf Übereinstimmungen in den Textknoten getestet. Als Ergebnis erhält man einen Pfadausdruck, der den gefundenen Knoten selektiert.

- Attribut *Produktbeschreibung* - Sie enthält normalerweise mehr Text als der Produktname und kann sich auf mehrere Tags erstrecken. Die bekannte Produktbeschreibung wird auf einzelne Tokens verteilt und nach Treffern auf den Seiten in der Trainingsmenge gesucht. Dabei wird die Position jedes Treffers als XPath-Ausdruck gespeichert. Abhängig von dem eingesetzten IR-Algorithmus für die Berechnung der Trainingsmenge können eventuell Techniken wie Stoppwort-Filer, Stemming und Thesaurus angewendet werden. Schließlich werden die gewonnenen XPaths generalisiert und zu einem generalisierten Pfadausdruck kombiniert.
- Attribut *Produktpreis* - Es stellt den Preis eines Produktes dar und ist dadurch charakterisiert, dass der Wert bei unterschiedlichen Webshops abweicht. Unser Ansatz fokussiert auf die Erkennung von diversen Zahlen- und Währungsformaten. Die Wertdifferenz darf dabei eine gewisse Grenze nicht überschreiten, die im Verhältnis zum durchschnittlichen Preis des Produktes gemessen wird.

Wenn die Pfadausdrücke für alle notwendigen Attribute vorhanden sind, kann mit der Mustererkennung in allen Webseiten des Shops begonnen werden. Wie bereits erwähnt, werden die einzelnen Seiten anhand der HTML-Struktur als Baum modelliert, in dem der erste Tag (<html>) die Wurzel ist. Der zu untersuchende Online-Katalog kann als eine Menge von Bäumen betrachtet werden. Damit die Verwaltung der Dokumente und die Suche nach Mustern vereinfacht wird, werden sie in einer geeigneten Datenstruktur gespeichert. Jedem Objekt wird dabei eine laufende Nummer zugewiesen, die für die spätere Identifizierung verwendet werden kann und zusammen mit seiner Internetadresse (URL) als Parameter gespeichert wird. Die einzelnen Bäume lassen sich zu einem gerichteten Graph zusammenfügen, in dem die Verlinkungen zwischen den Seiten als Kanten dargestellt werden. Anhand dieses Modells können die Beziehungen zwischen den einzelnen Seiten analysiert und eventuell Kategorien gebildet werden. Alternativ lassen sich die Dokumente flach als eine Sammlung von Bäumen speichern, was Vorteile bei der Performance bringen kann. Damit bilden die Bäume einen „Wald“ in der Datenstruktur.

Für die Darstellung der hierarchischen Dokumentenstruktur ist der XML-Format gut geeignet. Er besitzt eine implizite Baumstruktur, und der HTML-Quellcode der Seiten lässt sich mit einigen Anpassungen einfach in XML konvertieren. Die Dokumentenverlinkung, die in dem einen Ansatz als Kanten repräsentiert wird, kann mit speziellen Verweisen in Form von XML-Parametern verwirklicht werden. Der Vorteil der XML-Darstellung gegenüber HTML-Darstellung liegt in der Flexibilität, neue Parameter und Knoten zu definieren. Zusätzlich vereinfachen fertige Implementierungen von XML-Datenbanken die Verwaltung der Dokumente. Sie unterstützen meistens die Ausführung von Abfragesprachen wie XQuery und XPath, mit deren Hilfe die Erkennung der Pfadmuster effizient durchgeführt werden kann.

Für jedes der Attribute wird anhand der generalisierten Pfadausdrücke eine Abfrage in der XML-Datenbank gestartet. Die generierten Ergebnisse der Queries bestehen aus den Attributwerten, wie Produktname oder Produktbeschreibung. Damit die Abfrage zusätzlich die komplette

Webseite samt Wurzelknoten liefert, muss sie entsprechend angepasst werden. Anstatt die Knoten mit den Attributwerten auszuwählen, ist der Wurzelknoten (HTML-Tag) zu markieren. Das folgende Beispiel selektiert einen Nachkommen des Kontextknotens „h1“ namens „html“:

```
/catalog/html/body/../../table[id='product']/tr/td[2]/h1/ancestor::html
```

Im letzten Schritt der Klassifizierung werden die Ergebnisse, die bei der Mustererkennung gewonnen werden, zusammengefügt und analysiert. Dabei wird eine Schnittmenge der Ergebnisse gebildet, und die Dokumente, die außerhalb dieser Menge liegen, werden ausgefiltert. Als Erfolgskriterium kann die Relation der Dokumente, die als Produktseiten klassifiziert sind, zu allen anderen Seiten genommen werden. Die Idee besteht darin, die Anzahl an Produktseiten in einem Webshop grob zu schätzen. Falls die Anzahl an Ergebnissen zu gering oder zu groß ist, ist eventuell eine Verfeinerung der Pfadausdrücke notwendig. Die Struktur eines Onlineshops kann als ein vollständiger k -Baum betrachtet werden, dessen Kanten ausgehend von der Startseite bis zu den Produktseiten die Navigation eines Benutzers abbilden. Die Startseite spielt dabei die Rolle der Wurzel, die Produktseiten sind normalerweise die Blätterknoten, und alle restlichen (inneren) Knoten sind Zwischenseiten, wie zum Beispiel Kategorieauflistungen. Die Anzahl an Knoten n in einem vollständigen k -Baum wird folgendermaßen berechnet:

$$n = k^{h+1} - 1$$

wobei h die Höhe des Baumes repräsentiert. Die Anzahl der Blattknoten l wird folgendermaßen berechnet:

$$l = k^h$$

Beim Vergleich beider Funktionen fällt auf, dass die Anzahl der Blätter fast genauso schnell wächst wie die Anzahl der Knoten. In der Praxis ist h mindestens 3 ($h \geq 3$) und k grösser als 10 ($k \geq 10$). Daraus folgt, dass die Anzahl der Produktseiten sich nicht viel von der aller Seiten unterscheidet. Je größer der Katalog, desto geringer wird der Unterschied. In Kenntnis dieser Parameter lässt sich die Produktseitenanzahl besser schätzen.

5 Eigene Implementierungen und Tools

5.1 Software-Architektur

In Zusammenhang mit der vorliegenden Diplomarbeit wurde eine Anwendungssoftware für Informationsgewinnung aus Onlineshops entwickelt. In diesem Kapitel werden die Architektur und die einzelnen Module detailliert beschrieben. Zusätzlich werden auch alle verwendeten „Third-party“-Softwarekomponenten kurz vorgestellt und beschrieben. Im folgenden Diagramm (Abbildung 5.1) wird grob die Architektur des kompletten Systems graphisch dargestellt und anschließend beschrieben:

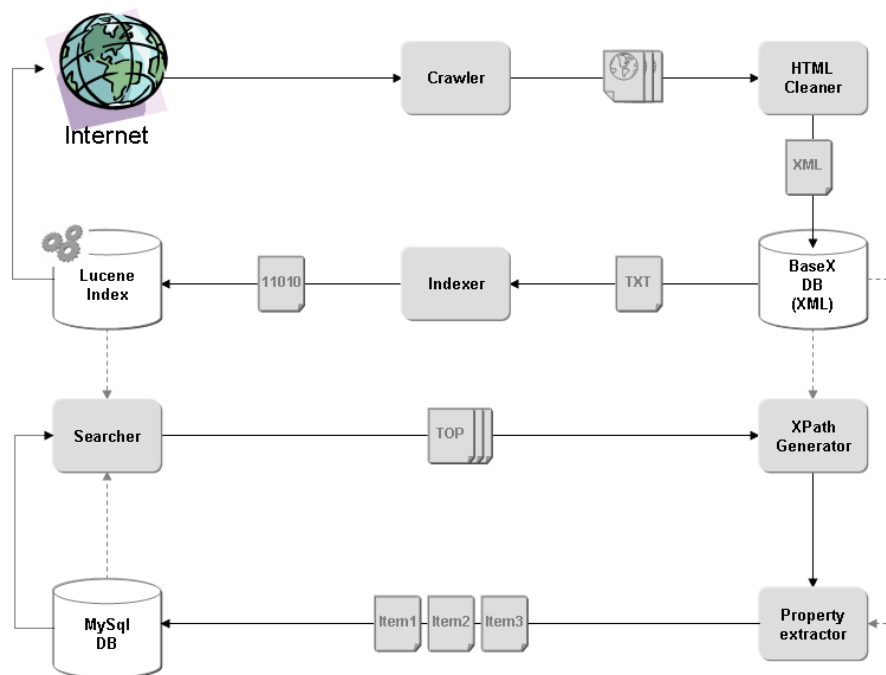


Abbildung 5.1: Architektur der entwickelten Software

Die Kernaufgabe des entwickelten Programms ist die Internetseiten einer vorgegebener Menge von Webshops zu analysieren und automatisch die Daten der online verkauften Artikel zu entnehmen. Bevor die Webseite analysiert werden können, müssen sie lokal auf dem Server gespeichert

werden. Bei der Aufgabe sind mehreren Programmkomponenten beteiligt. Als erstes werden alle erreichbaren Webseiten der Internetshops vom *Crawler* besucht und heruntergeladen. Anschließend werden sie vom *HtmlCleaner* analysiert und in gültigen XML-Dokumente konvertiert. Direkt danach gelangen die Dokumente in einer nativen XML-Datenbank - *BaseX*, um die Verwaltung der einzelnen zu vereinfachen. Um das Durchsuchen der Webseiten zu optimieren werden deren Inhalte mit Hilfe von dem IR-System *Lucene* indexiert. Um die Webseiten in der XML-Datenbank aktuell zu halten wird diese Prozedur in bestimmter Abständen neu angestoßen. Sobald alle Daten vorrätig sind, kann mit der Analyse der Webseiten begonnen werden. Das Hauptziel dabei ist bestimmte Muster zu generieren, anhand deren die Webseiten, die detaillierten Produktangaben beinhalten (Produktseiten), in der XML-Datenbank erkannt werden können. Diese Aufgabe wird von dem *XPath-Generator* übernommen, der eine Menge von potentiellen Produktseiten von dem von dem *Product-Page-Searcher* als Beispiel bekommt. Als Ausgangsbasis für die Generierung dieser Menge potentieller Produktseite werden bereits extrahierten Produktinformation von der *MySQL-Datenbank* verwendet. Sobald die XPath-Expressions erfolgreich generiert erfolgreich worden sind, werden sie von dem *Property-Extractor* verwendet um die Produktinformationen aus der XML-Datenbank zu extrahieren und in die MySQL-Datenbank zu importieren. Dort können sie als Ausgangsdaten für die Entdeckung der Produktseiten in weiteren Webshops verwendet werden.

5.2 Crawler

Der Crawler (auch *Spider* genannt) ist ein Programm, das automatisch Webseiten im Internet besucht und deren Inhalte abspeichert. Er startet auf einer Seite und verfolgt so lange die Verweise zu anderen Seiten (die *Hyperlinks*), bis ein bestimmtes Kriterium erfüllt ist. Im Allgemeinen werden sie von den Suchmaschinen verwendet. Es wird eine Liste verwaltet, die alle bearbeiteten Webdokumente samt URL, Besuchszeit und die Zeit für einen erneuten Besuch beinhaltet. Letzteres wird anhand eines Meta-Parameters auf der Webseite namens *Revisit-After* beeinflusst, wobei nicht alle Bots sich daran halten. Die bekanntesten Anwendungen sind unter anderem der *GoogleBot* und der *Slurp (Yahoo!)*, es existieren aber unzählige Individuallösungen und auch einige Open-Source-Projekte.

Für die Zwecke der Informationsgewinnung aus Onlineshops wurde eine eigene Lösung entwickelt. Das Programm hat die Aufgabe, bei gegebener Startseite eines Shops alle Seiten innerhalb der gleichen Domain zu besuchen und abzuspeichern. Als Programmiersprache wurde wegen der Plattformunabhängigkeit Java (JDK 1.6) vorgezogen. Außerdem lassen sich mit dieser Programmiersprache Multi-Threaded-Applikationen relativ leicht entwickeln. Das Programm bekommt als Eingabe eine Menge von Webshops in Form einer Liste, bestehend aus *Start-URL und Shop-Bezeichnung*, und eine *ID* für eine spätere Identifizierung. Für jeden Webshop wird ein Thread gestartet, der über die Webseiten „krabbelt“ und die bereits besuchten in eine Liste aufnimmt. Dort wird die Zugriffsadresse (URL), eine laufende Nummer und ein Verweis auf die Webseite (*ID*), von welcher der Bot gekommen ist, notiert. Anhand der Liste kann zusätzlich vermieden werden, dass sich der Bot im Kreis dreht. Damit der Crawler den Onlineshop nicht verlässt, wird bei jedem Link analysiert, ob der Hauptdomainname in der URL enthalten ist. Allerdings haben manchmal Werbungen als URL den gleichen Domainnamen und können trotzdem auf fremde Seiten verweisen. Dies wird mit einer internen Umleitung erreicht, hauptsächlich um die Klickrate

(CTR) zu messen. Als zusätzliche Maßnahme eignet sich in diesem Fall die Überprüfung des *HTTP Status Code* für Weiterleitungen auf eine andere Webseite (zum Beispiel *301 Moved Permanently*) [Con04]. Um den Crawler auf Dokumente einzuschränken, die er auch verarbeiten kann (HTML-/Textseiten), muss der Inhaltstyp überprüft werden. Dies geschieht anhand der *Content-Type* -Parameter im *MIME Header* [Bra02]. Für den Bot sind Textinhalte erlaubt, die durch den Wert *text/html* gekennzeichnet sind. Alle übrigen Dokumente sowie Bilder, Videos, PDF und andere binäre Daten können ignoriert werden.

Die wichtigsten Bestandteile unserer Crawler-Implementierung sind der *Http-Client* und die *Crawler-Klasse*. Der *Http-Client* bietet die Möglichkeit, über das *Http-Protokoll* auf Web-Inhalte zuzugreifen. Anstatt die in Java eingebauten Klassen (*java.net*) für die Kommunikation zu verwenden, haben wir stattdessen die *HttpClient-Komponente* der *Apache Software Foundation* gewählt - *Jakarta Commons HttpClient*¹. Dieses Open-Source-Tool bietet viele erweiterte Funktionalitäten, unter anderem *GET-* und *POST-Methoden*, *Cookie-Unterstützung* und *sichere Verbindungen*. Es besitzt eine *openUrl()* *Funktion*, die als Eingabe eine Webadresse bekommt, und liefert den Quellcode der Seiten unter der angegebenen URL. Die *Crawler-Klasse* erbt von *java.lang.Thread*. Diese abstrakte Klasse erlaubt eine *asynchrone bzw. parallele Ausführung* des Codes und wird mit den Funktionen *start()* und *stop()* gesteuert. Für jeden Webshop in der Liste wird ein Objekt vom Typ *Crawler* initialisiert, der dementsprechend eine Instanz der *HttpClient-Klasse* besitzt.

Im Internet sind verschiedene Standards für die Zeichenkodierung verbreitet, die für gleiche Zeichen meistens unterschiedliche Byte-Darstellungen verwenden. Unser Ziel ist es, eine Datenbank aufzubauen, in der die Informationen mit einer einheitlichen Codierung vorliegen. Entsprechend haben wir uns für die meist verbreitete *Unicode-Codierung* entschieden - *UTF-8*. Die Aufgabe des Crawlers ist es, die tatsächliche Codierung auf der Seite anhand der *Charset* Parameter im Header zu erkennen und in *UTF-8* zu konvertieren. Eine andere Problematik, mit der sich der Bot auseinandersetzen muss, sind die *Firewalls* der Websites. Auf Grund der unzähligen und intensiven Seitenaufrufe, die der Crawler produziert, könnten ihn die Serverprogramme als eine *DOS-(Denial of Service-)Attacke* ansehen. Das Ergebnis wäre eine Sperre der IP-Adresse des Clients oder die Generierung einer leeren Seite. Damit unser Programm die Sicherheitsmechanismen der Website nicht aktiviert oder, wenn solche fehlen, sie nicht komplett lahmlegt, werden die *Requests* des *HttpClient* künstlich verzögert. Durch die *Multitasking-Architektur* lassen sich für jeden *Crawler-Thread* bestimmte Pausen (500-1000ms) zwischen den *Requests* definieren. So wird der Server nicht unnötig belastet und in der Zwischenzeit bekommen die anderen laufenden *Crawler* Zugriff auf die Ressourcen. Bei größeren Onlineshops lassen sich alternativ auch mehrere *Crawler-Threads* mittels verschiedener *Proxy-Server* verwenden, was einerseits die Performance verbessert und andererseits die *Firewall-Regel* umgeht.

5.3 HTML Cleaner

Bevor die vom Crawler gelieferten Dokumente gesichert und verarbeitet werden können, müssen die Daten einige Vorbereitungsschritte durchlaufen. Wir verwenden dafür eine *Third-Party-Open-*

¹Jakarta Commons HttpClient - <http://hc.apache.org/httpclient-3.x/>

Source-Software namens *HtmlCleaner*², die sehr stabil und flexibel ist. Dieses Tool ist in Java entwickelt und kann leicht als Bibliothek in das komplette System integriert werden. Die Hauptaufgabe dieses Moduls ist die Konvertierung der HTML-Seiten in XML-Dokumenten. Oft sind die HTML-Seiten im Internet unsauber implementiert und haben eine fehlerhafte Struktur, die einige von den Browsern bei der Generierung der DOM-Objekte ignorieren und trotzdem richtig darstellen. Die XML-Dokumente unterliegen strikten Regeln und können andernfalls nicht geparkt werden. Aus diesem Grund sind HTML-Seiten in dieser Form für eine weitere Verarbeitung nicht geeignet. Der *HtmlCleaner* korrigiert die unsaubere HTML-Struktur und bringt Ordnung in die Tags und Attribute und produziert auf diesem Wege ein wohlgeformtes XML-Dokument. Die häufigste Ursache für das Entstehen einer fehlerhaften HTML-Struktur sind ungeschlossene oder fehlende Tags:

```
<table id=table1 cellspacing=2px
  <h1>CONTENT</h1>
  <td><a href=index.html>1 -> Home Page</a>
  <td><a href=intro.html>2 -> Introduction</a>
```

Der von *HtmlCleaner* produzierte Output sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head />
  <body>
    <h1>CONTENT</h1>
    <table id="table1" cellspacing="2px">
      <tbody>
        <tr>
          <td>
            <a href="index.html">1 -&gt; Home Page</a>
          </td>
          <td>
            <a href="intro.html">2 -&gt; Introduction</a>
          </td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

Zusätzlich können mit Hilfe des *HtmlCleaners* bestimmte Tags ausgefiltert werden, die für die weitere Verarbeitung unwesentlich sind. HTML-Tags und Attribute, die hauptsächlich Design- und Dekorationszwecken dienen, können bedenkenlos entfernt werden. Dies sind zum Beispiel *style*, *script*, *font*, *align*, *hr*, *etc.* . So wird zum einen eine Vereinfachung des Textes erreicht, zum anderen werden die Datenvolumen wesentlich verringert.

5.4 XML-Datenbank

Damit die Mustererkennung in Webseiten funktioniert, werden die Dokumente samt der kompletten HTML-Struktur gebraucht. Nach dem Bereinigungsprozess müssen die vom *HtmlCleaner*

²HtmlCleaner - <http://htmlcleaner.sourceforge.net/>

gelieferten XML-Dokumente in einer Datenstruktur gespeichert und verwaltet werden. Eine einfache Umsetzung wäre das Abspeichern der Dokumente als XML-Dateien auf dem Dateisystem. Kompliziert wird es beim Abfragen von Informationen aus tausenden von Dokumenten und bei der Mustererkennung der Dokumentstruktur. Außerdem ist dieser Ansatz nicht sehr performant. Genauso wenig ist dies das Speichern aller Dokumente in einer einzigen XML-Datei. Als eine flexible und performante Lösung bietet sich der Einsatz einer nativen XML-Datenbank an. Mit ihrer Unterstützung von komplexen Objektstrukturen eignen sich die sogenannten *native-XML*-Datenbanken (siehe Unterkapitel 4.3) hervorragend für die Verwaltung von Daten im DOM-Format.

```

- <root>
- <page id="1" name="" url="http://notebooknet.de/">
- <src>
- <html dir="ltr" lang="de">
+ <head></head>
- <body>
+ <br/>
+ <table width="980" border="0" cellspacing="0" cellpadding="0" align="center"></table>
+ <table width="980" border="0" cellspacing="0" cellpadding="0" align="center"></table>
+ <table class="box_border" width="980" border="0" cellspacing="2" cellpadding="0" align="center"></table>
- <div class="copyright">
+ eCommerce Engine © 2006
+ <a href="http://www.xt-commerce.com" target="_blank">xt:Commerce Shopssoftware</a>
- </div>
- </body>
- </html>
- </src>
- </page>
+ <page id="2" name="MSI WIND U120-1616UXP" url="http://notebooknet.de/product_info.php?info=p3977"></page>
+ <page id="3" name="Sony VAIO VGN-Z41WD/B" url="http://notebooknet.de/product_info.php?info=p4103"></page>
+ <page id="4" name="Sony VAIO VGN-CS31S/P" url="http://notebooknet.de/product_info.php?info=p4097"></page>
+ <page id="5" name="[Letztes>>]" url="http://notebooknet.de/product_info.php?info=p4099"></page>
+ <page id="6" name="[zurück]" url="http://notebooknet.de/product_info.php?info=p4096"></page>
+ <page id="162" name="" url="http://notebooknet.de/shop_content.php?coID=22"></page>
- </root>

```

Abbildung 5.2: Struktur einer Xml-Datenbank für ein Webshop

In unserem Projekt verwenden wir *BaseX*³, eine Open-Source-native-Xml-Datenbankanwendung, die an der Universität Konstanz entwickelt wurde. Das Programm ist in Java geschrieben und kann entweder als externe Jar-Bibliothek oder als Stand-Alone-Programm über die Konsole oder die GUI benutzt werden. Zusätzlich werden standardisierte APIs (wie z. B. XAPI⁴) angeboten, mit der sich Daten in der DB aus jedem Programm abfragen und aktualisieren lassen. Dank der eingebauten Server-Client-Architektur lässt sich die Datenbank bei größeren Projekten auch als ein Server ausführen. So kann zum Beispiel die Datenbank auf einen dedizierten Server ausgelagert werden. *BaseX* besitzt einen eingebauten *XQuery/XPath* -Prozessor und steigert die Performance mit Text-

³BaseX Xml-Datenbank - <http://www.inf.uni-konstanz.de/dbis/basex/>

⁴XAPI - <http://xmldb-org.sourceforge.net/xapi/>

und Attribut-Indizes. Selbst bei großen Datenmengen (mehr als 10000 Dokumenten) verhält sich die XML-Datenbank stabil. Das alles macht die Anwendung für IE aus (halb-)strukturierten Daten in hohem Maße geeignet.

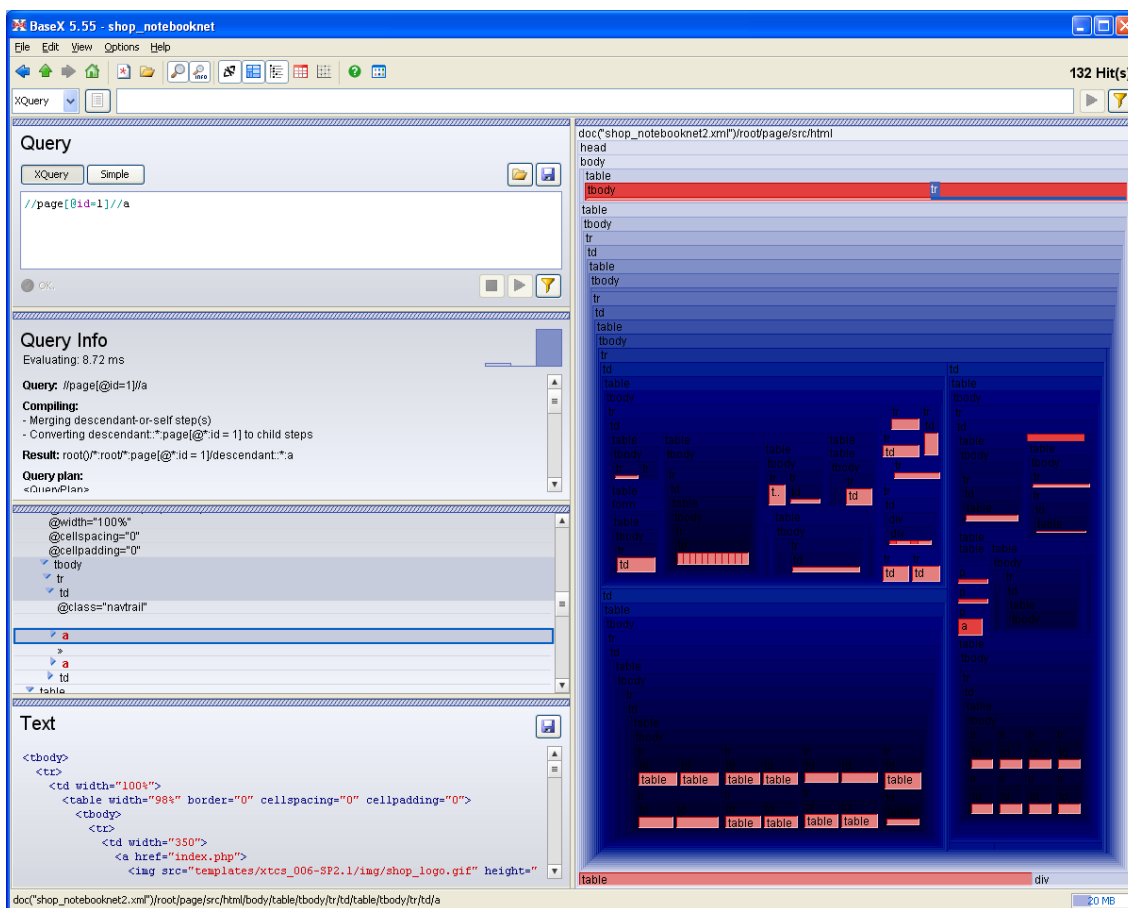


Abbildung 5.3: Die GUI von BaseX

Genauso wie bei den relationalen Datenbanksystemen werden die Daten in BaseX in unterschiedlichen Datenbanken verwaltet. Für Datenbanken lassen sich spezifische Einstellungen, wie Zeichencodierung und Indizes, konfigurieren. Die Kernfunktionen in einer Datenbank sind das Einfügen, Abfragen, Aktualisieren und Entfernen von Inhalten mit Hilfe von XPath-Ausdrücken. In unserem IE-System verwenden wir BaseX für die Verwaltung der Webseiten, die in dem vorherigen Schritt in XML-Dokumente konvertiert worden sind. Für jeden untersuchten Onlineshop wird eine Datenbank erstellt und dort werden alle vom Crawler erreichten Seiten gesichert. Schließlich kann die Datenbank als eine Baumstruktur in XML-Format betrachtet werden. Die einzelnen Seiten sind als Teilbäume gespeichert und können bei Bedarf über den ID-Parameter adressiert werden. Die bereits im vorigen Kapitel beschriebenen Pfadausdrücke für die Seitenklassifizierung können als XPath in der Datenbank ausgeführt werden, indem die komplette Webseite nach bestimmten Mustern durchsucht wird. In Abbildung 5.2 wird die XML-Darstellung eines Webshops dargestellt:

Mit dem Ziel, die Kommunikation und Interaktion mit der XML-Datenbank zu vereinfachen, haben wir eine spezielle Klasse implementiert - den *XMLCatalog*. Er hat alle notwendigen Funktionen für Operationen mit XML-Dokumenten. Der Konstruktor öffnet eine bestehende Datenbank oder erstellt eine neue, falls keine mit dem angegebenen Namen existiert. Mit der Funktion *insert-Document()* wird eine neue Webseite zum Katalog hinzugefügt. Die Funktion *query()* führt eine XPath-Abfrage und liefert eine Menge von XML-Elementen als Ergebnis. Nicht gebrauchte Elemente lassen sich mit der Member-Funktion *delete()* entfernen. Zusätzlich wurde eine Funktion *cleanUpCatalog()* implementiert, die Elemente mit einer gewissen Häufigkeit erkennt und eliminiert. Damit lassen sich beispielsweise HTML-Elemente in der Kopfzeile oder Fußzeile, die auf fast jeder Webseite vorkommen und für die IE nicht von Interesse sind, komplett entfernen. Das vereinfacht zum einen die Analyse und zum anderen verringert es die Datenmengen.

BaseX bietet zusätzlich ein hochinteraktives GUI (Graphical User Interface) mit sehr intuitiven Visualisierungsmöglichkeiten von XML-Dokumenten und Datenbanken. Die sogenannte *TreeMap*-Visualisierung repräsentiert alle XML-Nodes als Vierecke, und je nach Tiefe werden sie dunkler oder heller gefärbt. Die mit einer XPath-Abfrage selektierten Knoten werden rot markiert. Mit Hilfe dieses Interface können bei Bedarf die aufgenommenen Webseiten analysiert und die XPath-Ausdrücke manuell generiert werden, was besonders für die Initialphase unseres IE-Prozesses wichtig ist. In Abbildung 5.3 wird die BaseX-GUI im Einsatz vorgestellt.

5.5 Lucene

Lucene⁵ ist eine performante, voll funktionsfähige Open-Source-Such-Engine, die von der Apache Software Foundation entwickelt wurde und vollständig in Java geschrieben ist [Ima]. Sie erlaubt dem Entwickler, mühelos Suchmöglichkeiten in ihren Anwendungen hinzuzufügen. Wie die meisten IR-Systeme besteht der Retrieval-Prozess mit Lucene aus zwei Hauptschritten. Zunächst wird der gesamte Text aus allen Dokumenten indexiert und anschließend wird mit Hilfe des Indexes gesucht. Beide Operationen sind sehr effizient und selbst mit geringen Ressourcen durchführbar. Zum Beispiel können über 20MB/Minute auf einem Pentium M 1.5Ghz indexiert werden, und dabei lässt sich das Datenvolumen auf unter 30% der ursprünglichen Größe komprimieren. Lucene basiert auf einem modifizierten Vektorraummodell, um mehr Performance zu gewinnen. Im klassischen Vektorraummodell sind sowohl die Abfragen als auch die Dokumente Vektoren in einem n-dimensionalen Raum, wobei n die Anzahl der Keywords ist. Die Ähnlichkeit wird anhand des Winkels zwischen den einzelnen Vektoren gemessen, indem der Query-Vektor allen Dokumenten-Vektoren im Index gegenübergestellt wird, was nicht besonders performant ist. In dem modifizierten Vektormodell von Lucene wird zuerst eine boolesche Suche ausgeführt, um Dokumente, die keines der Keywords beinhalten, auszusortieren. Anschließend wird die Relevanz der gebliebenen Dokumente kalkuliert und so eine Ergebnismenge generiert. Die Implementierung des Indexes in Lucene basiert auf der invertierten Indexierung und steigert damit die Effizienz der Abfragen.

⁵Lucene - <http://lucene.apache.org/>

5.5.1 Indexaufbau

In unserem IE-System wird Lucene als externe Java-Bibliothek (jar) eingebunden. Es wurde eine Klasse mit allen notwendigen Methoden für die Indexierung und die Suche von Dokumenten implementiert, die auch Lucene über die API steuert. Diese Klasse namens *XmlCatalogSearcher* wird jeweils für einen Onlineshop bzw. eine XML-Datenbank in BaseX als eine Instanz erstellt. Im Konstruktor wird eine Referenz des XML-Katalogs in BaseX übergeben, wo anschließend die Dokumente abgefragt werden. Außerdem wird eine Referenz im lokalen Verzeichnis erstellt, in dem der Index verwaltet wird. Falls es noch nicht existiert, wird ein neues Verzeichnis mit leerem Index initialisiert. Nachdem der Crawler alle Webseiten im XML-Format in der BaseX gespeichert hat, kann mit dem Indexieren der Dokumente begonnen werden. Das wird in der Funktion *indexCatalog()* ausgeführt und vom Crawler angestoßen. Als Erstes wird Lucene's *IndexWriter* instanziiert und die Index-Einstellungen werden gesetzt. Die wichtigste unter diesen ist die Wahl des Index-Analysierers (*Analyzer*), der die Dokumentinhalte in Tokens zerlegt und für die Suche vorbereitet. Der Analyseprozess besteht hauptsächlich aus einem *Tokenizer* und eventuell aus einem oder mehreren Filtern. Lucene bietet verschiedene eingebaute *Analyzer*, unter anderem:

- *StandardAnalyzer* - ein universaler Analyser, der besonders gut für die meisten europäische Freitext-Dokumente geeignet ist. Er nutzt den grammatikbasierten *StandardTokenizer*, der eine Vielzahl an Punctuationssymbolen und Separatoren wie Bindestrich etc. berücksichtigt. In der Filterphase werden die Tokens kleingeschrieben (*LowerCaseFilter*), der Text wird normalisiert und eine Reihe von Stoppwörter werden von dem *StopFilter* entfernt.
- *SimpleAnalyzer* - benutzt den *LetterTokenizer*, der die Zerteilung der Texte in Tokens bei solchen Zeichen macht, die der Klasse der Nicht-Buchstaben angehören. Diese Technik ist für Produktseiten wenig geeignet, da Produktnamen meistens aus alphanumerischen Zeichen bestehen. Als Filter wird nur der *LowerCaseFilter* verwendet.
- *SnowballAnalyzer* - basiert auf dem *StandardAnalyzer* und filtert die Wörter anhand eines *Snowball-Stemmers*⁶. Zur Auswahl stehen eine Menge von Stemmern, unter anderem *PorterStemmer* oder *LovinsStemmer*, sowie eine Reihe von sprachspezifischen Implementierungen.
- *PatternAnalyzer* - ein effizienter *Tokenizer*, der die Tokens im Text anhand benutzerdefinierter Regular Expressions bildet. Die Technik bietet mehr Flexibilität und ist einfacher als die eigene Implementierung eines Analyzers.
- *ShingleAnalyzer* - ein *N-Gram*-basierter Analyser, der bei der Tokenisierung bestimmte Wörter berücksichtigt, die im Text häufig zusammen vorkommen. Diese Technik ist besonders bei Produkteigenschaften von Vorteil, die meistens zusammen mit den Maßeinheiten im Text zu finden sind (z. B. 4 GB, 2.2 Ghz, 50 Zoll, 120 Watt etc.).

Für die Indexierung von Webseiten nutzen wir den eingebauten *ShingleAnalyzer*, wobei sich mit eigenen Erweiterungen der Abstraktklasse *Analyzer* definitiv bessere Ergebnisse erzielen lassen.

Nachdem Lucene's Indexer initialisiert wurde, werden die Dokumente iterativ indexiert. Dazu wird eine Abfrage in der XML-Datenbank ausgeführt, die alle vorhandenen Webseiten liefert.

⁶Snowball Stemmer - <http://snowball.tartarus.org/>

Für jede Seite werden die ID und der Quellcode entnommen. Die Seiten-ID wird in einem nicht durchsuchbaren Feld gesichert und dient für die Identifizierung in der XML-Datenbank. In das Default-Feld kommt der eigentliche Textinhalt der Seite, der vorher mit Hilfe von `HtmlCleaner` von dem Quellcode extrahiert und anschließend normalisiert worden ist. Die HTML-Tags sind für die Suche nicht relevant und können vor dem Indexierungsprozess entfernt werden, damit die Datenvolumen im Index reduziert werden.

5.5.2 Suchen mit Lucene

Lucene verfügt über reichhaltige eingebaute Suchoptionen. Eine Abfrage besteht aus Termen und Operatoren, wobei mehrere Terme Phrasen bilden können, indem sie in Anführungszeichen eingeschlossen werden. Ein einzelner Term könnte z. B. *Intel* sein, und eine Phrase sieht folgendermaßen aus - „*Intel Core 2*“. Durch Einsatz von booleschen Operatoren können durch Kombination von Termen und Phrasen komplexe Abfragen entstehen. Beim Suchen nach *Intel Core 2*-CPUs, die aber nicht vier Kerne besitzen, lässt sich folgende Abfrage generieren:

```
("Intel Core Duo" OR "Intel Core 2") NOT Quad
```

Bei der Suche mit einfachen Termen besteht die Möglichkeit, Wildcards und Fuzzy-Logik zu verwenden. Ähnlich wie bei Suche im Dateisystem über die Konsole wird bei der Wildcardsuche ein Fragezeichen als Platzhalter für ein Zeichen verwendet, für mehrere Zeichen kann das Sternchen benutzt werden. Ein Beispiel lässt sich bei der Suche nach bestimmten Namen bilden: `Ma?er` findet nur Namen wie *Maier und Mayer*, `An*` findet dagegen alle Namen, die mit *An* anfangen, wie *Anna, Anton etc.* Die Fuzzy-Suche in Lucene basiert auf dem Levenshtein-Distanz-Algorithmus, der die Anzahl der Editoperationen als Ähnlichkeitskriterium nimmt. Die Fuzzy-Terme werden mit dem Tilde-Symbol markiert (~). Zusätzlich kann der Ähnlichkeitsgrad wunschgemäß mit einem Zahlparameter zwischen 0 und 1 angepasst werden. Diese Technik wird standardgemäß für die Suche in Texten mit Tippfehlern verwendet und kann auch in unserem System eingesetzt werden. Besonders bei kleineren Onlineshops, bei denen die Daten manuell eingepflegt werden, können Tippfehler vorkommen (z. B. *Ericson statt Ericsson*). Nachteilig kann dies besonders für Produkte mit alphanumerischer Bezeichnung sein, bei denen ein falsches Zeichen eine große Rolle spielen kann. Nehmen wir als Beispiel die Produktgruppe der TFT-Monitore; dort steht meist am Anfang des Modellnamens die Displaygröße. Zum Beispiel seien *19WTFG* und *21WTFG* zwei Monitore der gleichen Marke, der eine ist 19 Zoll und der andere 21 Zoll groß. Für Suchabfragen mit Phrasen bietet Lucene eine Distanzsuche, indem nach Wörtern gesucht wird, die eine maximale Distanz voneinander haben, die in Wörtern gemessen wird. Formuliert wird die Suche, indem die Phrase von dem Tilde-Symbol (~) gefolgt wird. Diese Technik wird in unserem Programm für das Aufspüren von Produktbezeichnungen auf einer Webseite benutzt. Angenommen, es wird nach dem Produkt *Acer TravelMate 6593G 500GB 4096MB* gesucht, das auf der zu durchsuchenden Webseite zwar vorhanden ist, deren Bezeichnungsbestandteile dort jedoch eine andere Reihenfolge aufweisen. Zum Beispiel:

```
Acer TravelMate 6593G 4096MB 500GB  
Acer TravelMate 4096MB 500GB 6593G  
Acer TravelMate 15.4" 500GB 4096MB 6593G  
Acer 6593G 4096MB 500GB TravelMate
```

Eine funktionierende Abfrage, die alle 4 Fälle erfasst, sieht folgendermaßen aus:

```
"Acer TravelMate 6593G 500GB 4096MB"\~5
```

wobei die Zahl 5 die Distanz in Wörtern angibt, d. h., wie viele Wörter zwischen jedem Wort der Phrase maximal stehen können. Alternativ kann eine Suchabfrage generiert werden, die über den Produktnamen hinaus zusätzlich auch alle Produktdaten aus der Beschreibung beinhaltet. Weil das Produkt anhand des Produktnamens identifiziert werden kann, ist diese Information viel relevanter als die Beschreibung, dessen Terme auch auf vielen anderen Seiten vorkommen können. In Lucene besteht die Möglichkeit, in einer Query bestimmten Termen (oder Phrasen) eine höhere Gewichtung zu geben. Die wichtigeren Ausdrücke werden mit dem Symbol `^`, gefolgt von einer Zahl (>0), markiert, was die Rolle eines Multiplikationsfaktors spielt. In der Abfrage für das Produkt aus dem vorherigen Beispiel werden die Keywords in der Bezeichnung als zweimal wichtiger eingestuft, so dass sie folgendermaßen aussieht:

```
Acer^2 TravelMate^2 6593G^2 500GB^2 4096MB^2 Intel Core 2Duo T9600  
Arbeitsspeicher 4096MB DDR3 500GB S-ATA Festplatte 15.4" WSXGA+ TFT Display
```

Die Klasse *XmlCatalogSearcher*, die extra für die Interaktion mit Lucene entwickelt wurde, besitzt eine Funktion für die Suche nach Produkten im Katalog. Die Funktion *searchProduct()* bekommt als Eingabe die Produktbezeichnung und die Beschreibung und liefert eine Menge von relevanten Dokumenten - die sogenannten *TopDocuments*. Die Webshops sind so aufgebaut, dass die detailliertesten Informationen über ein Produkt auf der Produktseite zu finden sind, wo das Produkt einzeln repräsentiert wird. Aus diesem Grund wird davon ausgegangen, dass bei gegebenen Produktdaten anhand einer passenden Abfrage mit großer Wahrscheinlichkeit die entsprechende Produktseite gefunden werden kann. Als Ausgangsdaten für die Abfrage dienen der Produktname und die Beschreibung, die an verschiedenen Stellen im Dokument zu finden sind. Der Produktname besteht normalerweise aus ein paar Wörtern, z. B. Hersteller, Modellname und eventuell einige Produktmerkmale. Die Produktbeschreibung kann tabellarisch oder als Freitext vorkommen und variiert, je nach Produkt und Shop, zwischen ein paar Sätzen und mehreren Absätzen. Wie in Unterkapitel 5.1 erwähnt, können die Daten für die Suche aus einer relationalen Datenbank stammen, die entweder manuell oder durch bereits bekannte Onlineshops erstellt wurde. Die Abfragen werden vom Programm automatisch generiert und anschließend von Lucene's *QueryParser* geparkt. Für das Durchsuchen der indexierten Webseiten bieten sich unterschiedliche Arten von Abfragen an:

- Die einfachste Variante, eine Query aufzubauen, ist es, den Text aus Produktname und Beschreibung zu kombinieren und dem QueryParser von Lucene zu übergeben. Der Text wird dann automatisch analysiert und mit Hilfe des ausgewählten Lucene's Analyzer in den Vektorraum übersetzt. Der Nachteil dabei ist, dass die Produktbezeichnung die gleiche Gewichtung erhält wie die Produktdaten, was die Fehlerquote erheblich erhöhen kann.
- Eine andere Möglichkeit ist es, zwei unterschiedliche Abfragen zu erstellen und anschließend eine Schnittmenge der Ergebnisse zu bilden. Zusätzlich zu der Query aus der ersten Variante wird eine neue Abfrage erstellt, die nur den Produktnamen beinhaltet. Der Vorteil gegenüber der ersten Variante liegt darin, dass mit der zweiten Abfrage und mit der Bildung der Schnittmenge den Keywords aus der Produktbezeichnung mehr Relevanz verlieht wird.

- Eine bessere Alternative ist es, eine einzelne Abfrage zu erstellen, in der den Termen des Produktnamens mit Hilfe der Technik *Term-Boosting* mehr Gewicht gegeben wird. Dazu werden die Terme mit dem Symbol \wedge und einer Zahl (Multiplikatorfaktor) für die Relevanz versehen.
- Zusätzlich zum *Term-Boosting* kann von der *Distanzsuche* in Lucene Gebrauch gemacht werden. In den meisten Fällen besteht der Produktname aus zwei bis sechs Wörtern, die in einer Zelle auf der HTML-Seite positioniert sind. Die Ergebnisse lassen sich präzisieren, indem die Abfrage mit Hilfe der Distanzsuche so formuliert wird, dass die Keywords aus dem Produktnamen auf einer Webseite nahe beieinander auftauchen müssen. Eine solche Abfrage kann folgendermaßen aussehen:

```
"Acer TravelMate 6593G 500GB 4096MB"\~5 AND  
Acer^2 TravelMate^2 6593G^2 500GB^2 4096MB^2 Intel Core 2Duo T9600  
Arbeitsspeicher 4096MB DDR3 500GB S-ATA Festplatte 15.4" TFT Display
```

Die Query wird geparkt und dient als Eingabeparameter für die Lucene-Klasse *IndexSearcher*, die die Suche im Index ausführt und, geordnet nach Relevanz, die besten Ergebnisse liefert. In den meisten Fällen bekommt die gesuchte Produktseite den höchsten Score und steht als erste in der Ergebnismenge, sofern das gesuchte Produkt im Katalog existiert. Allerdings muss eine gewisse Fehlertoleranz erlaubt werden und Situationen berücksichtigt werden, in denen entweder das Produkt nicht vorhanden ist oder eine Nicht-Produktseite bessere Scores bekommen hat. Als nächster Schritt werden die Top-*x*-Dokumente analysiert und die potenziellen Nicht-Produktseiten herausgefiltert.

5.6 XPath Generator

Die Kernaufgabe unseres Projektes ist die Extrahierung der Produkteigenschaften aus einem Onlinemarkt, dessen Struktur initial nicht bekannt ist. Fakt ist, dass gleichartige Webseiten innerhalb einer Domain eine nahezu identische Quellcode-Struktur besitzen. Das liegt daran, dass sie automatisch anhand einer Datenbank generiert werden und dass für Dokumente wie die Produktseite ein Template existiert. Drauf basiert das entwickelte Verfahren für die Erkennung der Produktseiten in einem neuen Shop. Das Ziel ist dafür einen bestimmten Muster automatisch zu generieren und auf alle Webseiten anzuwenden, um sie so im neuen Shop nach Produkt- und Nicht-Produktseiten zu klassifizieren. Bei vorhandenen Informationen, wo genau im Quellcode der Produktseiten ein Produktattribut zu finden ist, kann einen Muster in Form von XPath-Ausdruck entnommen werden und anschließend bei Anwendung auf die restliche Dokumenten die Attributswerte aller Produkte gewonnen werden. Auf den folgenden Seiten wird die Implementierung eines Moduls beschrieben, das, mit Hilfe einer Beispielsmenge von Produkten, die XPath-Expressions für die Produktattribute *Bezeichnung*, *Beschreibung* und *Preis* generiert. Da diese Attributen nur auf Produktseiten vorkommen, werden sie für die Klassifizierung der Webseiten in Produkt- und Nicht-Produktseiten verwendet. Der Prozess der XPath-Generierung geht über mehrere Schritte, die vom Modul *XPath-Generator* ausgeführt und gesteuert werden. In Abbildung 5.4 wird grob die Struktur des Moduls anhand einer Klassendiagramm.

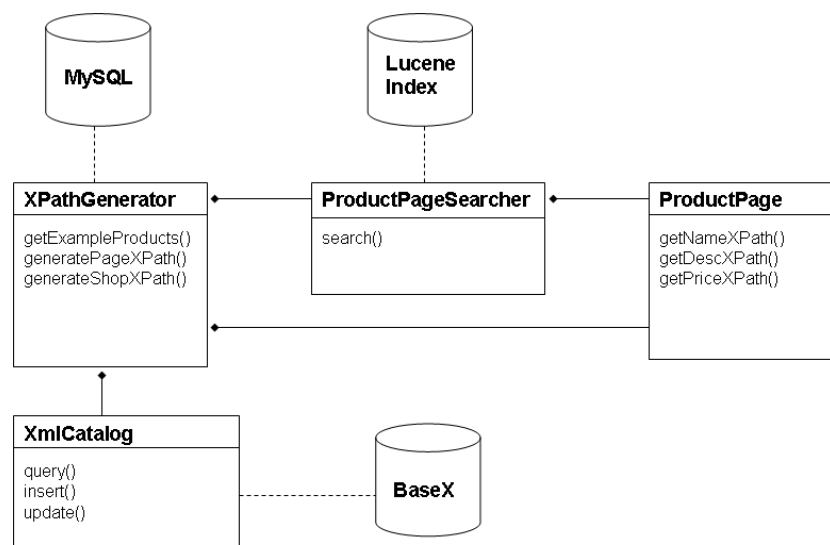


Abbildung 5.4: Struktur des XPath-Generators

Die Klasse *XPathGenerator* ist das Herzstück dieses Projektes und wird pro Onlineshop initialisiert. Nachdem der XML-Katalog und der Lucene-Index vollständig aufgebaut sind, übernimmt *XPathGenerator* die Kontrolle. Das Objekt braucht den Namen des XML-Kataloges für die Initialisierung und erstellt automatisch eine Instanz der *XmlCatalog*-Klasse im Konstruktor. Zusätzlich wird eine Verbindung zu der relationalen Datenbank aufgebaut, wo die bereits extrahierten Produktdaten verwaltet werden. Die Klasse besitzt eine Hauptfunktion *generateShopXPath()*, die sofort nach der Initialisierung aufgerufen werden kann. Als Erstes wird mit der Funktion *getExampleProducts()* eine Abfrage nach zufälligen Datensätze aus der Produktdatenbank ausgeführt, um eine Menge mit Beispielsprodukten zu erstellen. Zusätzlich wird ein *ProductPageSearcher*-Objekt erstellt. Für jedes Produkt aus der Datenbank wird eine Suche im XML-Katalog bzw. im Index mit Hilfe von Lucene ausgeführt. Die Funktion liefert eine Menge potentieller Produktseiten. Anschließend wird die Funktion *generatePageXPath()*, die jeden der Treffer analysiert. Die Funktion hat die Aufgabe, die Position der Produktattribute in einem Dokument zu lokalisieren und gegebenenfalls XPath-Ausdrücke für die Identifizierung im Quellcode zu bilden. Als Eingabe dienen die Werte der Produktattribute aus der Datenbank (Name, Beschreibung und Preis) und der Quellcode der zu analysierenden Webseite, die aus *BaseX* stammt und im XML-Format vorhanden ist. Für die Aufgabe der Erkennung der Produktattribute und die Generierung der XPath-Ausdrücke auf einer Webseite haben wir eine spezielle Klasse entwickelt - *ProductPage*. Für jede zu analysierende Webseite aus der Menge der potentiellen Produktseiten wird ein Objekt von diesem Typ instanziiert. Der Konstruktor bekommt als Inputparameter den Quellcode der Webseite in XML-Form. Da für die Erkennung der verschiedenen Attribute unterschiedliche Strategien notwendig

sind, bietet die Klasse für jedes von ihnen eine *Public-Funktion*, die den Wert aus der MySQL-Datenbank als Eingabe braucht. Momentan werden die Attribute *Produktname*, *Beschreibung* und *Preis* unterstützt, wobei die Klasse mit neuen Funktionen erweitert werden kann.

5.6.1 XPath-Ausdruck für den Produktnamen

Die Expression für den Produktnamen auf einer potenziellen Produktseite wird mit der Funktion *getNameXPath()* gebildet. Als Parameter wird die Produktbezeichnung aus der Datenbank übermittelt. Ähnlich wie bei Lucene wird der Name in Tokens zerlegt, indem sämtliche Symbole (Strich, Komma, Punkt, Querstrich etc.) als Trennzeichen benutzt werden. Einige Sonderfälle müssen dabei berücksichtigt werden, wie zum Beispiel Trennsymbole in Dezimalzahlen, bei denen nicht gesplittet werden darf. Zusätzlich wird ein Stopfilter eingesetzt, um Wörter wie *in*, *mit*, *und*, *new* etc. zu entfernen. Die Produktnamen auf der untersuchten Produktseiten können von dem in der Datenbank hinterlegten Wert in gewissem Maße abweichen. In diesem Projekt werden zwei Fälle berücksichtigt:

- In verschiedenen Webshops können im Produktnamen unterschiedliche Separatoren verwendet werden. Zum Beispiel kann das Produkt „Eizo S2402WH-GY“ auf einer anderen Seite in folgender Form vorkommen: „Eizo - S2402WH/GY“. Um diese Unterschiede zu berücksichtigen, wird aus den Tokens ein regulärer Ausdruck gebildet, der unterschiedliche Trennzeichen berücksichtigt. Zum Beispiel wird ein *RegExp* für das erwähnte Produkt so aussehen:

```
"Eizo[\s\-\_\\\/\.\\,\\;]{0,4}S2402WH[\s\-\_\\\/\.\\,\\;]{0,4}GY"
```

Der Text in den eckigen Klammern definiert die Zeichen, die als Separatoren zu erkennen sind. Weil die meisten davon Meta-Zeichen in *RegExp* sind, werden sie mit einem führenden Querstrich versehen. Die Zahlen in den geschweiften Klammern geben die Länge der Separatoren in Zeichen an. In dem obigen Beispiel kann es zwischen Null und vier Trennzeichen geben.

- Die Reihenfolge der Terme in der Produktbezeichnung kann in verschiedenen Systemen auch unterschiedlich ausfallen. Manchmal können sogar einige von ihnen komplett fehlen. Der Produktname „Speicher für Notebook 1024MB DDR 333Mhz“ kann zum Beispiel in einem anderen Katalog als „Notebook Speicher 1024MB 333MHz DDR“ vorkommen. Bei der Suche mit Lucene werden diese Abweichungen mit Hilfe des Vektormodells berücksichtigt und die Dokumente als relevant gekennzeichnet. Weil Lucene die Position der Terme im Text nicht liefert, wird eine Methode entwickelt, die nach verschiedenen Kombinationen von Tokens sucht. Weil die Länge der Produktbezeichnung nur wenige Tokens beträgt, bietet sich die Möglichkeit an, alle Permutationen der Tokens auszuprobieren. Die Anzahl an möglichen Kombinationen ist durch $n!$ begrenzt, wobei n der Anzahl der Tokens im Produktnamen entspricht. In der Praxis ist n selten größer als 5 und damit die Anzahl der Permutationen kleiner als 120. Damit die Präzision der Suche erhöht wird, werden Mengen, die weniger als zwei Tokens haben, entfernt. Es ist nicht sinnvoll, nach einem Term zu suchen wie „Speicher“ oder „Notebook“. Für den Produktnamen aus dem obigen Beispiel

werden nach dem Entfernen des Stopwortes „für“ und den Mengen mit weniger als zwei Tokens 114 sinnvolle Permutationen erzielt, unter anderem:

1. Speicher, Notebook, 1024MB, DDR, 333Mhz
2. Speicher, Notebook, 1024MB, 333Mhz, DDR
3. Speicher, Notebook, DDR, 333Mhz, 1024MB
- ...
114. Notebook, Speicher

Die Permutationen werden absteigend nach Mengengrößen sortiert und in dieser Reihenfolge ausprobiert. Beim ersten Treffer wird die Suche abgebrochen. Auf diese Weise wird eine Art Distanzsuche implementiert, die erfolgreich abweichende Produktbezeichnungen auf einer Seite aufspürt.

Nachdem die genaue Bezeichnung des gesuchten Produktes auf der untersuchten Produktseite bekannt ist, wird ein XPath-Ausdruck für die Identifizierung der Position im Quellcode generiert. Für diesen Zweck wurde eine spezielle Funktion namens *getXpathAbs()* in der *Utils*-Klasse implementiert, die als Eingabe den Quellcode des XML-Dokuments und den Inhalt eines Knotens bekommt und eine absolute XPath-Expression liefert, die den Weg von der Wurzel (HTML) bis zum Knoten beschreibt. Auf einer Produktseite kann der Produktname eventuell an mehreren Stellen vorkommen, was in mehreren XPath-Ausdrücken resultieren kann. Deswegen stellt es eine Schwierigkeit dar, den richtigen Platz auf der Produktseite zu lokalisieren. Hierzu wird eine Heuristik verwendet, die auf unterschiedlichen Wichtigkeiten der HTML-Tags basiert. Dabei werden alle Tags, die Textinhalte besitzen können, nach Wichtigkeit sortiert:

$h1 > h2 > \dots > h6 > p > strong > b > i > div > td > span > a$

Falls mehrere XPaths erkannt werden, wird derjenige genommen, der einen besser platzierten HTML-Tag markiert. Eine ähnliche Technik wird von den Suchmaschinen für die Berechnung der Gewichtung der Terme auf einer Webseite verwendet.

5.6.2 XPath-Ausdruck für die Produktbeschreibung

Der nächste Schritt ist die Generierung eines XPath-Ausdrucks für die Produktbeschreibung. Sie besteht normalerweise aus mehreren Sätzen und kann in tabellarischer oder in Form eines Fließtextes formuliert werden. Im Gegensatz zum Produktnamen, der komplett in einem HTML-Tag enthalten ist, kann der Text-Inhalt der Beschreibung auf mehrere Tags verstreut sein. Unser Ziel ist es, einen XML-Knoten (HTML-Tag) zu lokalisieren, der die komplette Beschreibung samt Kindknoten beinhaltet. Wir haben dafür eine spezielle Funktion namens *getDescXPath()* in der Klasse *ProductPage* entwickelt, die als Eingabeparameter die originelle Beschreibung aus der Datenbank bekommt und den XPath-Ausdruck für den HTML-Tag zurückgibt. Als Erstes wird, genauso wie beim Produktnamen, die übermittelte Produktbeschreibung in Tokens zerlegt. Zusätzlich können Techniken wie Stopfilter und Stemmer verwendet werden, um möglichst viele der vorhandenen Schlüsselwörter zu entdecken. Für jeden der Tokens aus der vorhandenen Beschreibung wird eine Suche im XML-Dokument ausgeführt. Die Treffer landen in einer Liste, und anschließend wird die Position im Quellcode mit Hilfe absoluter XPath-Ausdrücke bestimmt. Wenn die Struktur des XML-Dokuments als ein Baum betrachtet wird, selektieren die XPaths die Knoten im Baum, die

Teile der Produktbeschreibung beinhalten. Der Algorithmus für die Generierung des XPath der Produktbeschreibung geht wie folgt vor. Für jeden der Knoten wird die Anzahl der Treffer berechnet, die in dem Knoten selbst oder in seinen Nachfolgerknoten enthalten sind. Als Datenstruktur lässt sich zum Beispiel eine Hash-Tabelle verwenden, in der die XPaths die Schlüssel sind und die Anzahl der Treffer als Wert gespeichert wird. Ausgehend von der Wurzel (HTML-Tag) wird rekursiv immer derjenige Kindknoten ausgewählt, der die meisten Treffer besitzt. Die Prozedur wird so lange wiederholt, bis ein Nachfolgeknoten mit weniger als $x\%$ aller Treffer (die Versuche wurden mit $50\% < x < 80\%$ durchgeführt) entdeckt oder ein Blattknoten erreicht wird. Falls der komplette Inhalt in einem Blattknoten enthalten ist, wird seine XPath-Expression als Lokalisierungspfad ausgewählt. Andernfalls wird ein Elternknoten selektiert, dessen Nachfolger mindestens einen bestimmten Teil der Keywords aus der Beschreibung beinhalten.

5.6.3 XPath-Ausdruck für den Produktpreis

Die größte Herausforderung stellt das Aufspüren des Produktpreises auf einer Webseite aus der Beispielsmenge dar. Für diese Aufgabe wurde die Funktion *getPriceXPath()* in der Klasse *Xml-ProductPage* entwickelt. Genauso wie bei der Funktion für die Generierung von Mustern für den Produktnamen und die Beschreibung wird auch in diesem der Produktpreis aus der Datenbank als Eingabeparameter übermittelt. Selbst wenn dieser Wert in der Regel stark in den einzelnen Webshops abweicht, wird es einen Orientierungswert geben, anhand dessen der tatsächliche Preis entdeckt werden kann. Die Idee besteht darin, den kompletten Inhalt der Webseite nach bestimmten Zahlen- beziehungsweise Währungsmustern zu durchsuchen und den Wert mit dem vorhandenen Produktpreis zu vergleichen. Zusätzlich zum bekannten Produktpreis wird als zweiter Eingabeparameter die XPath-Expression für den bereits gefundenen Produktnamen benutzt. Die Produkseite ist so strukturiert, dass der Produktpreis in der Regel relativ nahe beim Produktnamen positioniert ist. Die genaue Position des Produktnamens kann dazu dienen, bei mehreren potenziellen Ergebnissen eine gewisse Gewichtung zu vergeben. Das kann besonders bei Webseiten, die auch andere (ähnliche) Produkte am Ende der Seite präsentieren, von Bedeutung sein. Das Programm kann deren Preise fälschlicherweise als Produktpreis erkennen, obwohl sie vom Layout her in einem anderen Seitenbereich positioniert sind.

Der Algorithmus in der Funktion *getPriceXPath()* funktioniert folgendermaßen. Zunächst werden alle möglichen Zahlenformate definiert, die mit regulären Ausdrücken zu erkennen sind. Als Dezimalkomma bzw. Tausendeseperator ist auf Grund der unterschiedlichen Länderstandards sowohl das Komma als auch der Punkt zu berücksichtigen. Ein Währungssymbol erhöht die Gewichtung bei mehreren möglichen Ergebnissen. Anschließend werden anhand der XML-Struktur die Nachbarknoten der Produktbezeichnung schrittweise nach Zahlenmustern durchsucht. Dazu wird ein XPath-Ausdruck konstruiert, der alle nachfolgenden Blattknoten links und rechts des Produktnamens iterativ ausgibt. Deren Inhalte werden mit Hilfe der bereits definierten RegExp überprüft. Die Ergebnisse werden zusammen mit dem absoluten XPath-Ausdruck und der Entfernung von dem Bezeichnungsknoten, die in der Anzahl der Zwischenknoten gemessen wird, in einer Liste hinzugefügt. Als nächster Schritt werden die Ergebnisse in der Liste analysiert. Dazu werden die Einträge nach Distanz aufsteigend sortiert. Zusätzlich werden alle Zahlen bzw. Preise aussortiert, die von dem bekannten Produktpreis zu stark abweichen. Als Grenzwert wurde ein Unterschied von 50% definiert. Es sei zum Beispiel der Preis eines Artikels 100 EUR, dann dürfen die Ergebnisse unserer Mustererkennung nur in dem Interval 50-150 EUR liegen, sonst werden

sie ignoriert. Als potenzieller Produktpreis wird derjenige Eintrag ausgewählt, dessen Distanz von dem Produktnamen am geringsten ist.

Nicht bei allen Webshopsystemen werden die Artikelpreise im Klartext dargestellt. Manchmal werden Bilder verwendet, um den Preis besonders hervorzuheben oder um ihm ein spezielles Design zu geben. Dabei greifen die Webshop-Entwickler auf eine Technik zurück, bei der jede Ziffer durch ein entsprechendes Bild präsentiert wird. Es lassen sich auch mehrstellige Zahlen anzeigen, indem mehrere nebeneinander positionierte Bilder geladen werden. Auch für das Dezimalkomma und für den Tausenderseparator kann ein entsprechendes Bild existieren. Die Erkennung eines solchen Musters erfordert eine andere Vorgehensweise als bei der bereits beschriebenen Suche nach Klartext-Preisen, die im Folgenden vorgestellt wird. Dabei wird der Umstand ausgenutzt, dass die Dateinamen der Bilder entsprechend den zu repräsentierenden Ziffern benannt werden. Zum Beispiel können die Bilder, die den Preis "345,00" anzeigen, folgendermaßen aussehen:

```
<div id=price>






</div>
```

Alternativ können die Dateinamen anstelle der entsprechenden Ziffern auch die Zahlen als Wörter beinhalten. Zum Beispiel kann die Zahl 1 durch eine Bilddatei namens „one.gif“ oder „eins.gif“ dargestellt werden.

```
<div id=price>






</div>
```

Für das Erkennen von solchen Mustern auf einer Produktseite haben wir spezielle reguläre Ausdrücke entworfen. Mit Hilfe von XPath-Ausdrücken wird die komplette Webseite nach Bildern durchsucht, die eine oder mehrere Ziffern im Dateinamen beinhalten. Danach wird nach gemeinsamen Elterntags gesucht, die ein oder mehrere Tags vom Typ Bild beinhalten. Solche Knoten können den Produktpreis beinhalten und müssen genauer analysiert werden. Alle Kindknoten vom Typ Bild werden anschließend für Zahleninformationen überprüft und gegebenenfalls wird die erkannte Zahl zusammen mit der entsprechenden XPath-Expression des Parentknotens für eine spätere Analyse gespeichert. Im Folgenden wird vorgestellt, wie das konkret umgesetzt wird.

1. Damit die gesuchten Zahlenmuster im Quellcode entdeckt werden können, haben wir eine Art Vokabular definiert, das anhand einer Hash-Funktion für jedes mögliche Schlüsselwort,

das in einem Zahlenmuster vorkommen kann, den entsprechenden Wert zurückliefert. In Java wurde das mit einer Hash-Tabelle realisiert, die folgendermaßen aussieht:

```
public static Hashtable<String,String> digitSigns(){
    Hashtable<String,String> digitSigns = new Hashtable<String,String>();

    digitSigns.put("dot", ".");
    digitSigns.put("punkt", ".");

    digitSigns.put("comma", ",");
    digitSigns.put("komma", ",");

    digitSigns.put("00", "0");
    digitSigns.put("0", "0");
    digitSigns.put("zero", "0");
    digitSigns.put("null", "0");

    digitSigns.put("01", "1");
    digitSigns.put("1", "1");
    digitSigns.put("one", "1");
    digitSigns.put("eins", "1");

    digitSigns.put("02", "2");
    digitSigns.put("2", "2");
    digitSigns.put("two", "2");
    digitSigns.put("zwei", "2");

    digitSigns.put("03", "3");
    digitSigns.put("3", "3");
    digitSigns.put("three", "3");
    digitSigns.put("drei", "3");

    digitSigns.put("04", "4");
    digitSigns.put("4", "4");
    digitSigns.put("four", "4");
    digitSigns.put("vier", "4");

    digitSigns.put("05", "5");
    digitSigns.put("5", "5");
    digitSigns.put("five", "5");
    digitSigns.put("fuenf", "5");
    digitSigns.put("funf", "5");
}
```

Abbildung 5.5: Hashtabelle als Vokabular für unterschiedlichen Zahlenformate

2. Es wird ein XPath-Ausdruck definiert, der alle Bildelemente auf einer Webseite selektiert, die Zahleninformation besitzen. Beispielsweise kann er so aussehen:

```
"//img[
contains(@scr,'0') or contains(@scr,'zero') or
contains(@scr,'1') or contains(@scr,'one') or
...
contains(@src,'9') or contains(@scr,'nine')]"
```

In diesem Fall liefert `@src` den Wert des Parameters `src`. Die Funktion `contains()` bekommt als Eingabe zwei Strings und überprüft, ob der zweite Parameter im ersten enthalten ist.

3. Nachdem die XPath-Abfrage ausgeführt worden ist und die gewünschten Elemente vom Typ „img“ vorhanden sind, kann mit der Suche nach deren Elternknoten weitergemacht werden. Dazu wird eine neue XPath-Abfrage generiert, die anhand des Dateinamens des Bildes die Elternknoten selektiert:

```
"//img[@scr='one.jpg']/.."
```

Die zwei Punkten am Ende des Ausdruckes stehen für den Elternknoten des vorherigen Elementes. Für jeden der gefundenen Bild-Knoten wird so ein XPath-Ausdruck ausgeführt. Anschließend werden mit Hilfe der Funktion *getAbsXPath()* aus der *Tools*-Klasse die absoluten Pfad-Ausdrücke der Elternknoten bestimmt, die potenziell eine XPath-Abfrage für den Produktpreis darstellen. Ein Beispiel für solch einen Ausdruck:

```
"/html/body/div[id='main']/div[id='product']/div[id='price']"
```

4. Wenn alle potenziellen Expressions vorhanden sind, wird deren Inhalt analysiert. Die Bild-Elemente werden dabei iterativ für Zahleninformationen überprüft, indem nach dem Schlüssel aus der Hash-Tabelle von Punkt 1 gesucht wird. Auf diesem Weg können alle Zahlen innerhalb der untersuchten Parentknoten erkannt werden, die mit Bildern repräsentiert werden. Als Ergebnis wird eine assoziative Array erstellt, die in Form einer Hash-Tabelle in Java umgesetzt wird. Als Schlüssel steht die XPath-Expression des Elternknotens und als Wert die erkannte Zahl.
5. Der nächste Schritt besteht darin, die bereits vorhandenen Einträge der Hash-Tabelle zu analysieren. Dabei werden die Inhalte aller XPath-Expressions mit den bereits definierten regulären Ausdrücken überprüft. Diejenigen, die den Test nicht bestehen, werden ausgefiltert. Der Wert wird als eine Dezimalzahl geparkt und dem vorgegebenen Preis aus der Datenbank gegenübergestellt. Genauso wie bei der Erkennung von Klartextpreisen wird als Abweichungsgrenze 50% definiert. Zusätzlich wird die Entfernung der gefundenen Knoten von der Position des Produktnamens mit Hilfe einer entsprechenden Funktion berechnet, die die Anzahl an Zwischenknoten anhand der Baumstruktur des Dokuments berücksichtigt. Als potenzielle Preisposition wird die XPath-Expression ausgegeben, die dem Produktnamen am nächsten liegt und deren Wert sich in den zulässigen Grenzen befindet.

5.6.4 Analyse der Produktseiten und XPath-Generierung

Die Hauptfunktion *generateShopXPath()* generiert eine Menge von XPath-Expressions, die für das Adressieren der Produktseiten in dem Onlineshop dienen. Dieser Prozess kann als eine Klassifizierung der Seiten im XML-Katalog betrachtet werden, welche die Dokumente in zwei disjunktive Mengen einteilt - die Menge der Produktseiten und alle restlichen Seiten. Von Interesse ist in unserem Projekt die Menge der Produktseiten, anhand deren die Produkteigenschaften extrahiert werden (siehe 5.7). Die generierte Menge von potentieller Produktseiten müssen analysiert werden, um eventuelle Nicht-Produktseiten, die fälschlicherweise von dem *ProductPageSearcher* als Produktseiten erkannt worden sein, für die weitere Bearbeitung auszufiltern. Der erste Teil der Analyse geschieht in der Klasse *ProduktPage* bei der Generierung der XPath-Expression für den Produktnamen. Dort wurde der Html-Element der Produktbezeichnung entnommen. Wird sie in einem A-Tag (Hyperlink) auf der Seite dargestellt, so wird diese XPath-Expression ignoriert. Tritt der Produktname nur in Hyperlinks auf der untersuchten Seite, so wird kein XPath-Ausdruck für den Produktnamen generiert und entsprechend die Seite als Nicht-Produktseite erkannt. Das wird aus dem Grund gemacht, dass auf einer Produktseiten der Produktname nicht verlinkt ist.

Der zweite Schritt bei der Analyse der potentiellen Produktseiten (und Auswertung der gefundenen XPaths) besteht in der Ausführung dieser Abfragen im XML-Katalog. Für jede potenzielle

Produktseite existieren drei Pfadausdrücke, die als Muster für die Erkennung weiterer Produktseiten dienen. Das ist die Produktbezeichnung, die Beschreibung und der Produktpreis. Der XPath-Befehl, der in dem XML-Katalog auszuführen ist, wird so formuliert, dass als Ergebnis die Menge der Produktseiten zurückgeliefert wird. Die einzelnen XPaths (Produktname, Beschreibung und Preis) dienen dabei als Auswahlkriterium.

Typ	XPath
Produktname	<code>"/html/body/div[2]/div[5]/div[2]/h2"</code>
Beschreibung	<code>"/html/body/div[2]/div[5]/div[4]"</code>
Produktpreis	<code>"/html/body/div[2]/div[5]/div[2]/div[3]"</code>

Zum Beispiel sieht eine XPath-Abfrage für das Auswählen von Webseiten mit gleicher Struktur in BaseX folgendermaßen aus:

```
"/page/src
[/html/body/div[2]/div[5]/div[2]/h2]
[/html/body/div[2]/div[5]/div[4]]
[/html/body/div[2]/div[5]/div[2]/div[3]]/.."
```

Jeder der einzelnen Ausdrücke wird in eckige Klammern gesetzt und anschließend nebeneinander positioniert. Auf diese Art bilden sie eine *AND* Bedingung. Die Abfrage selektiert also nur XML-Dokumente von BaseX, die jeden der drei Ausdrücke in den eckigen Klammern beinhalten. Mit den zwei Punkten am Ende der Abfrage (..) wird der Elternknoten vom Typ „src“ ausgewählt.

Als Kriterium für den Erfolg einer Abfrage dient die Anzahl der generierten Ergebnisse. Dabei wird sie mit der Anzahl aller vorhandenen Seiten im XML-Katalog verglichen. Wie in Unterkapitel 4.6.2 beschrieben, nähert sich die Anzahl an Produktseiten in einem Webshop der Anzahl aller Seiten im Onlinekatalog. Daher ist eine XPath-Abfrage, die deutlich weniger Ergebnisse als die gesamte Seitenanzahl in BaseX liefert, mit sehr großer Wahrscheinlichkeit fehlerhaft und selektiert keine bzw. nur sehr wenige Produktseiten. In diesem Fall ist die Abfrage entweder falsch generiert und wird entfernt, oder sie ist einfach zu speziell und selektiert nur eine Untermenge der Produktseiten. Um herauszufinden, welcher der beiden Fälle vorliegt, werden die XPath-Ausdrücke, die viele Ergebnisse liefern, mit denjenigen, die wenige Ergebnisse generieren, verglichen. Wenn sich die Ausdrücke an mehr als *i* Stellen (Knoten) unterscheiden, ist höchstwahrscheinlich bei der Berechnung ein Fehler passiert. In diesem Fall wird die XPath mit der geringeren Trefferanzahl entfernt. Andernfalls werden die unterschiedlichen Stellen durch Wildcards ersetzt und die beiden Ausdrücke werden zu einem einzigen zusammengefügt. Die unterschiedlichen Knoten werden in der XPath-Abfrage durch *node()* ersetzt, was jeden Knoten selektiert.

5.6.5 Generalisierung der XPath-Ausdrücke

Im Folgenden wird die Generalisierung der XPath-Expressions beschrieben. Diese Verarbeitungsschritte werden im zweiten Teil der Funktion *generateShopXPath()* durchgeführt. Im besten Fall sind die generierten Ausdrücke für alle gefundenen Produktseiten im Webshop identisch. Dann hat das Programm mit sehr großer Wahrscheinlichkeit die richtigen Muster generiert, die alle

Produktseiten und die Produkteigenschaften erkennen. Allerdings kann es in verschiedenen Onlineshops vorkommen, dass die Produktseiten keine konstante Struktur besitzen, was zu abweichenden XPath-Expressions führt. Um möglichst viele unterschiedliche Produkte des Shops zu entnehmen, müssen die XPaths miteinander verglichen und die Ergebnisse analysiert werden. Anschließend werden sie generalisiert, indem die Knoten, die nicht übereinstimmen, durch Wildcards ersetzt werden.

Um die Fehlerquote bei der Erkennung von Mustern auf der Produktseite zu minimieren, wird die Prozedur mit mehreren Produkten aus der Datenbank wiederholt. Dabei werden bei Funktionsaufrufen mit Produkten, die in dem untersuchten Webshop nicht vorhanden sind, keine Ergebnisse generiert. Der Prozess für die Produktseitenerkennung mit Artikeln aus MySQL wird so lange fortgesetzt, bis mindestens n Produkte im Webshop gefunden worden sind. Dann wird aus allen XPath-Ausdrücken ein generalisierter Ausdruck gebildet, der möglichst viele Produktseiten selektiert und so für den kompletten Webshop gültig ist. Je mehr XPath-Ausdrücke für potenzielle Produktseiten vorhanden sind, desto präziser kann auch die Klassifizierung der Seiten durchgeführt werden und entsprechend besser ist die Informationsgewinnung aus diesen Webseiten.

5.7 Property Extraction

Im vorherigen Schritt wurden XPath-Befehle gebildet, mit denen sich alle Produktseiten adressieren und ausgeben lassen. Für den Aufbau der Produktdatenbank müssen die einzelnen Produktattribute aus dem Quellcode der Produktseiten extrahiert werden. Für diese Aufgabe wurde eine spezielle Klasse entwickelt, die anhand der zuvor gewonnenen XPath-Expressions von jeder erkannten Produktseite die Attribute entnimmt. Die Klasse wird mit einer Instanz der Klasse *XmlCatalog* initialisiert, die für die Kommunikation mit dem XML-Katalog eingesetzt wird. Die notwendigen XPath-Ausdrücke werden mit der Funktion *addExpressions()* der Klasse übergeben. Für das Funktionieren braucht die Klasse die XPath-Abfrage für die Produktseiten sowie die Abfragen für die Produktattribute. Die XPath-Ausdrücke werden als Array der Klasse übergeben, die die folgenden Ausdrücke beinhaltet:

1. XPath für die Produktseite
2. XPath für den Produktnamen
3. XPath für die Produktbeschreibung
4. XPath für den Produktpreis

Für das Funktionieren braucht die Klasse mindestens eine Menge mit den oben aufgelisteten XPath-Befehlen. Da die Klasse *XPathGenerator* auch mehrere Ausdrücke für unterschiedlich strukturierte Produktseiten liefern kann, kann die Funktion mehrmals aufgerufen werden. Die unterschiedlichen Expressions werden in einer neuen Array gespeichert, die als Parameter der Funktion *addExpressions()* gesetzt wird. In der Klasse intern werden die unterschiedlichen Expressions in einer Array-Liste verwaltet. Zum Beispiel:

XPath-Expressions 1	
Typ	XPath
Produktseite	"/page/src [html/body/div[2]/div[5]/div[2]/h2] [html/body/div[2]/div[5]/div[4]] [html/body/div[2]/div[5]/div[2]/div[3]]/../@id"
Produktname	"/html/body/div[2]/div[5]/div[2]/h2"
Beschreibung	"/html/body/div[2]/div[5]/div[4]"
Produktpreis	"/html/body/div[2]/div[5]/div[2]/div[3]"
XPath-Expressions 2	
Typ	XPath
Produktseite	"/page/src [html/body/div[1]/table[2]/tr[2]/td[1]] [html/body/div[1]/table[2]/tr[3]] [html/body/div[1]/table[2]/tr[2]/td[2]]/../@id"
Produktname	"/html/body/div[1]/table[2]/tr[2]/td[1]"
Beschreibung	"/html/body/div[1]/table[2]/tr[3]"
Produktpreis	"/html/body/div[1]/table[2]/tr[2]/td[2]"

Nachdem alle vorhandenen XPath-Ausdrücke in die Klasse *PropertyExtractor* eingespeist worden sind, kann die Extrahierung der Produkteigenschaften gestartet werden, was durch die Funktion *extractProperties()* angestoßen wird. Dort wird die Array-Liste durchlaufen und für jedes Element wird eine Suche im XML-Katalog gestartet. Als Erstes nimmt sich das Programm die Expression für die Produktseite. Anhand dieser werden alle entsprechenden Produktseiten in BaseX abgefragt. Anstatt den Quellcode lokal zu speichern und zu analysieren, bietet es sich an, nur die ID jeder Seite abzufragen. Anhand der ID lassen sich die Seiten in dem XML-Katalog eindeutig identifizieren und mit einer einfachen XPath-Abfrage adressieren:

```
"/page[@id=x]"
```

wobei *x* als Platzhalter für die Seiten-ID steht. Für jede Produktseite aus der Ergebnismenge wird die ID benutzt, um Abfragen für die einzelnen Produktattribute zu generieren. Die XPath-Ausdrücke für die drei Produktattribute Produktname, Produktbeschreibung und Preis sind bereits vorhanden und wurden als Muster für die Erkennung der Produktseite im Webshop verwendet. Diese XPath-Ausdrücke werden mit dem Ausdruck für die jeweilige Produktseite kombiniert, um die Werte der auf diesen Produktseiten dargestellten Produkte zu entnehmen. Auf diesem Weg entstehen für jedes potenzielle Produkt jeweils drei XPath-Abfragen. Mit den Daten aus dem obigen Beispiel entstehen so die folgenden XPath-Abfragen:

XPath für die Attribute eines Produkts mit ID <i>x</i>	
Attribut	XPath
Produktname	"/page[@id=x]/src/html/body/div[1]/table[2]/tr[2]/td[1]"
Beschreibung	"/page[@id=x]/src/html/body/div[1]/table[2]/tr[3]"
Produktpreis	"/page[@id=x]/src/html/body/div[1]/table[2]/tr[2]/td[2]"

Nach diesem Schritt sind die gesuchten Daten der jeweiligen Produkte vorhanden. Der nächste logische Schritt ist die Aufnahme dieser Daten in einer relationalen Datenbank, was im Unterkapitel 5.8 genauer beschrieben wird. Dies erledigt eine spezielle Funktion namens *insertProduct()* aus der Klasse *ProductImporter*. Dazu müssen die drei Attribute als Parameter übergeben werden. Bevor dieser Schritt ausgeführt werden kann, müssen die Daten entsprechend vorbereitet werden, weil sie direkt der Webseite im Rohformat entnommen werden. Die Produktnamen und die Beschreibungen müssen normalisiert werden, indem unnötige Leerzeichen und Zeilenumbrüche entfernt werden. Außerdem müssen gegebenenfalls die Inhalte der inneren XML-Knoten entnommen und die Markups entfernt werden. Beim Produktpreis ist die Aufgabe etwas komplizierter, weil die Rohdaten in Dezimalzahlen zu konvertieren sind. Als Erstes wird versucht, Preise zu extrahieren, die im Klartext vorhanden sind. Dazu werden die gleichen regulären Ausdrücke verwendet, die bereits bei der Erkennung von Zahlenformaten auf der Produktseite verwendet worden sind.

```
" ([0-9]{1,2} [.] [0-9]{3} [, ] [0-9]{2} (?!\\d) ) "
```

Ein Beispiel für reguläre Ausdrücke, die das Zahlenformat #.###,## erkennen.

Falls die regulären Ausdrücke keine Treffer liefern, liegt höchstwahrscheinlich ein Produktpreis im graphischen Format vor, d. h., die Zahlen sind durch Bilder repräsentiert. In diesem Fall wird der Inhalt des XML-Knotens, der den Produktpreis beinhalten soll, für Bilder untersucht. Dazu wird eine neue XPath-Abfrage erstellt, die alle Kindknoten vom Typ „img“ zurückliefert. Anschließend werden die Dateinamen der eingebundenen Bilder iterativ nach Zahleninformationen durchsucht. Genauso wie bei der Erkennung von Preisen auf der Produktseite werden Keywords bei der Suche benutzt, die normalerweise in den Dateinamen vorkommen können. Dazu wird die gleiche Hash-Tabelle verwendet, wie in Tabelle 5.5 angezeigt. Um den Produktpreis zu bekommen, werden die erkannten Ziffern und Symbole (wie zum Beispiel Dezimalkomma) nebeneinander positioniert und anschließend ins Dezimalformat konvertiert.

5.8 Datenbankimport

Das Ziel dieses Projekts ist es, eine übergreifende Produktdatenbank mit Daten aus verschiedenen Onlineshops aufzubauen. Die Produkte sollen dort in einer einheitlichen Form präsent sein. Die Produktdaten, die sich mit dem vorgestellten Verfahren aus Webshops extrahieren und in der Datenbank speichern lassen, sind die folgenden:

1. Produktbezeichnung - die Marktbezeichnung eines Produkts, die meistens aus der Marke und dem Modell besteht
2. Produktbeschreibung - detaillierte Beschreibung eines Produkts, die normalerweise auf der Produktseite zu finden ist und die wichtigsten Spezifikationen und Eigenschaften beinhaltet
3. Produktpreis - der Preis im jeweiligen Onlineshop

Für das Verwalten dieser Daten eignet sich am besten eine Tabelle mit den entsprechenden Spalten. Für die ersten zwei Attribute werden Datenbankfelder vom Typ *String* definiert. Der Produktpreis wird in einer Spalte im Dezimalformat gespeichert, damit er sortierbar und vergleichbar ist. Zusätzlich wird eine Referenz auf den Webshop, wo das Produkt entdeckt worden ist, erstellt. Dafür wird ein Feld für die Webshop-ID eingeführt. Damit das Produkt in der Produkttabelle bei einem neuen IE-Prozess wiedererkannt werden kann, wird ein weiteres Feld für die Internetadresse (URL) der Produktseite definiert. Zusätzlich zur Produkttabelle wird in der Datenbank eine Tabelle mit allen bekannten Onlineshops verwaltet. Dort werden Initial alle Shops eingepflegt, die mit Hilfe der entwickelten Software zu bearbeiten sind. Für jeden Shop wird eine laufende Nummer (ID) vergeben, die auch der Identifizierung in der Produkttabelle dient. Neben der ID und dem Shopnamen wird eine Spalte für die Internetadresse (URL) benutzt, mit der die Startseite aufrufbar ist. Zusätzlich wird ein Feld eingeführt, in dem das Datum der letzten Bearbeitung bzw. des Einlesens der Produktdaten gespeichert wird. In Abbildung 5.6 wird die komplette Datenbankstruktur dargestellt:

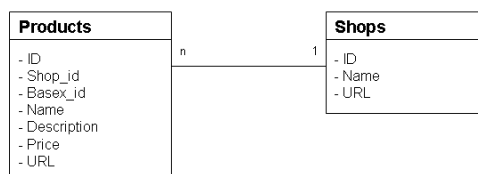


Abbildung 5.6: Struktur der Produktdatenbank in MySQL

Als Datenbanksystem wird die Open-Source-Software *MySQL*⁷ eingesetzt. Sie ist das bekannteste relationale Open-Source-Datenbanksystem, sehr stabil und performant und wird besonders bei webbasierter Software vorgezogen. Einer der Hauptvorteile gegenüber vielen kommerziellen Anwendungen ist die einfache Installation und Konfiguration. Es werden eine Reihe von Betriebssystemen unterstützt, was auch die Plattformunabhängigkeit unseres Systems nicht beeinträchtigt. Für die Administration der Datenbank eignet sich am besten eine webbasierte Open-Source-Software namens *phpMyAdmin*⁸, diese erfordert allerdings die Skriptsprache *PHP*⁹. Alternativ können die von MySQL mitgelieferten Konsolenprogramme oder GUI-Anwendungen, wie *MySQL Query Browser*, verwendet werden. Die MySQL-Datenbank wird als Serverprogramm ausgeführt und je nach Auslastung entweder auf dem gleichen Computer, mit dem die Anwendung ausgeführt wird, oder auf einem dedizierten Server installiert. Die Client-Anwendungen kommunizieren mit Hilfe von programmiersprachspezifischen APIs. Java-Applikationen verwenden für die Kommunikation mit verschiedenen Datenbanksystemen eine *JDBC*¹⁰-API, die eine datenbanksystemunabhängige Konnektivität ermöglicht.

⁷MySQL - <http://www.mysql.com>

⁸phpMyAdmin - <http://www.phpmyadmin.net/>

⁹Php - <http://www.php.net/>

¹⁰Java Database Connectivity (JDBC) - <http://java.sun.com/docs/books/tutorial/jdbc/index.html>

Die Kommunikation mit dem Datenbankserver wird in unserem Programm von der Klasse *ProductImporter* übernommen. Bei der Instanziierung der Klasse wird eine Verbindung mit der Produktdatenbank auf dem MySQL-Server aufgebaut. Zusätzlich wird die ID des untersuchten Onlineshops als Parameter übermittelt, damit alle Produkte dem entsprechenden Webshop in der Datenbank zugeordnet werden können. Die Produktdaten werden mittels einer Funktion *saveProduct()* in die Datenbank geschrieben. Als Eingabeparameter werden die drei Produktattribute (Produktname, Produktbeschreibung und Preis) verwendet, die bereits im vorherigen Schritt extrahiert worden sind. Für jedes der Produkte, die mit Hilfe der in Unterkapitel 5.7 beschriebenen Technik aus dem XML-Katalog extrahiert worden sind, wird die Funktion *saveProduct()* aufgerufen. Zusätzlich wird die Internetadresse (URL) der Produktseite als Parameter übermittelt, die für jedes Produkt in dem betreffenden Webshop unikal ist und für seine Identifizierung verwendet werden kann. Wenn ein Produkt mit der URL bereits in der Produkttabelle existiert, wird der Datensatz aktualisiert. Andernfalls wird ein neuer Datensatz angelegt.

6 Experimente

6.1 Ziel und Ablauf der Experimente

Um das Verhalten der in diesem Projekt entwickelten Software in einer produktiven Umgebung zu testen, wurden eine Reihe von Experimenten durchgeführt. Dazu wurden echte Daten aus ausgewählten Internetshops verwendet, um möglichst reale Bedingungen zu simulieren. Das Ziel der Experimente ist es in erster Linie, die Richtigkeit und die Zuverlässigkeit des implementierten Programms auf die Probe zu stellen. Zusätzlich wird die Stabilität der eingesetzten Fremdsoftware (Tools) getestet. Das entwickelte System muss in der Lage sein, automatisch das gesamte Produktangebot eines Webshops auszulesen und in einer relationalen Datenbank zu speichern. Bevor die Software für die automatische Produktextrahierung in dieser Form entstand, wurden verschiedene meist erfolglose Versuche in dieser Richtung gemacht. Die meisten davon werden in Unterkapitel 6.2 vorgestellt. Die dort entwickelten Verfahren und Tools konnten sich leider nicht als fehlerfrei und zuverlässig erweisen. Nichtsdestotrotz haben sie für die Entwicklung einer besseren Version beigetragen und aus diesem Grund können die Versuche als eine Vorbereitungsphase betrachtet werden. Nach der Auswertung der vielen Nachteile der ersten Version, wurde die Strategie der Informationsextrahierung aus Internetshops grundlegend geändert. Besonders die Methoden für Sammeln der Rohdaten, wurde komplett überdacht und so optimiert, dass möglichst viele und verschiedene Webshops ausgelesen werden konnten. Auch die Methoden für die Mustererkennung in neuen und noch nicht bekannten Webshops wurden präziser und robuster gemacht. Im vorliegenden Kapitel wird anschließend die Funktionsweise der neuen Version des Systems vorgestellt und hinsichtlich ihrer Richtigkeit und Effizienz ausgewertet.

Der Prozess der automatischen Produktextrahierung aus Webshops besteht aus verschiedenen Schritten und ist ein Zusammenspiel der einzelnen Programmmodule, deren Implementierung im vorherigen Kapitel ausführlich beschrieben wurde. Er lässt sich grob in drei Hauptphasen aufteilen (siehe Abbildung 6.1). Der erste Schritt ist die Datenaufbereitung, indem für eine vorgegebene Menge von Internetshops alle Webseiten lokal auf dem Server zu speichern sind. Zu diesem Zweck wird der selbst entwickelte Webcrawler eingesetzt, der mit Hilfe eines Http-Client alle erreichbaren Webseiten besucht und deren Quellcode entnimmt. Anschließend werden die Seiten ins XML-Format konvertiert und in der XML-Datenbank für weitere Analysen gesichert. Die genauen Einzelschritte dazu werden im Unterkapitel 6.3 erläutert und ausgewertet.

Die zweite Phase des Extrahierungsprozesses, der direkt nach der Datenaufbereitung folgt, ist das Initialisieren einer relationalen Datenbank mit einer Menge von Produktdaten. Da zu diesem Zeitpunkt noch keine Erkenntnisse über die Webshops und ihre Seiten, die sich in der XML-Datenbank befinden, vorliegen, ist ein manuelles Eingreifen notwendig. Diese Phase kann als ein Vorbereitungsschritt für den automatischen Aufbau der shopübergreifenden Produkttabelle betrachtet werden. Anstatt eine Liste mit Produktinformationen per Hand zusammenzustellen, wird das Modul *PropertyExtractor* (siehe Unterkapitel 5.7) verwendet, um Daten aus der XML-

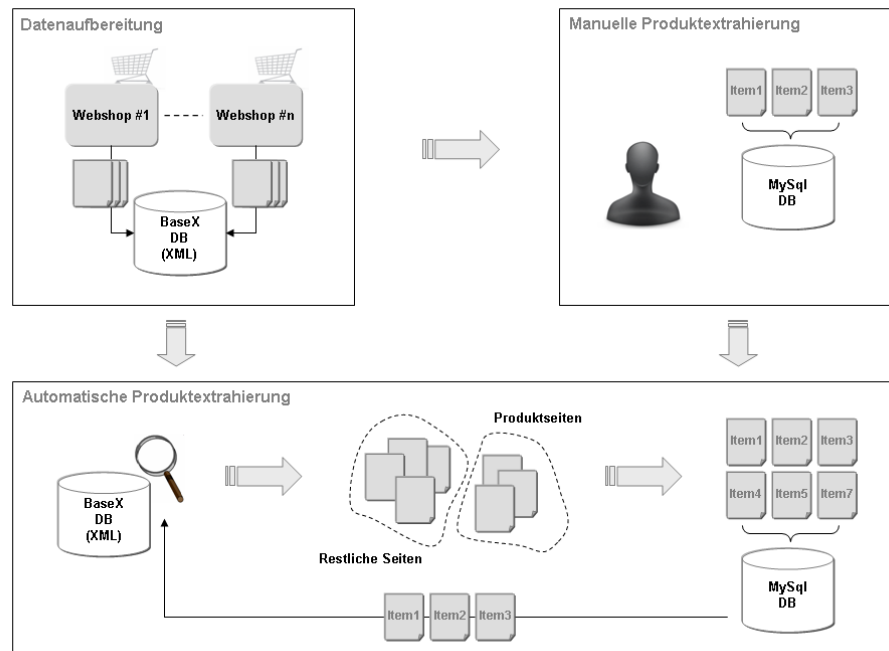


Abbildung 6.1: Ablauf der automatischen Produktextrahierung

Datenbank zu gewinnen. Dafür werden die Muster, die für die Erkennung der Produkteigenschaften notwendig sind, manuell entworfen und bei einigen der Webshops angewendet, die bereits vom Crawler besucht worden sind. Das Ziel ist dabei, die bereits aufgenommenen Produktdaten als eine Trainingsmenge für das automatische Verfahren zu verwenden. In Unterkapitel 6.4 wird diese Vorgehensweise detailliert beschrieben und anhand eines Versuches mit unterschiedlichen Internetshops erläutert.

Die letzte und die wichtigste Phase ist die automatische Extrahierung der Produktangebote von Webshops, die bereits vom Crawler ausgelesen sind und sich in der XML-Datenbank befinden. Die Funktionsweise und die durchgeführten Tests und Experimente werden detailliert in Unterkapitel 6.5 präsentiert. Das Hauptziel ist die automatische Generierung von Mustern, anhand deren die vorhandenen Produktdaten in einem nicht bekannten Webshop erkannt und extrahiert werden. Da die Rohdaten in XML-Format vorliegen, werden diese Muster als XPath-Ausdrücke formuliert, die bestimmte Positionen im Quellcode adressieren. Der erste Schritt in Richtung Erkennung von Produktdaten im Webshop ist die Erkennung der Webseiten, die ein Produkt detailliert präsentieren (Produktseiten). Gewünscht ist eine Klassifizierung der vorhandenen Webseiten in zwei Klassen - *Produktseiten* und *restliche Seiten*. Der erste Versuch die Webseiten zu Klassifizieren wurde mit *RapidMinder* gemacht und ist in Abschnitt 6.5.1 beschrieben. Nach dem erfolglosen Experiment wurde ein alternativer Ansatz für die Erkennung von Produktseiten verfolgt. Die Produktseiten sollen mit Hilfe von gezielter Suche nach Produktinformationen auf den Webseiten gefunden werden. Dafür wird ein Verfahren implementiert, das mit Hilfe von *Lucene* die Textinhalte der Webseiten nach bereits bekannten Produkten durchsucht. Die genaue Funktionsweise und die durchgeführten

Experimente werden im Abschnitt 6.5.2 vorgestellt. Die Treffer werden anschließend analysiert, um die potenziellen Nicht-Produktseiten auszufiltern. Danach werden von den Produktseiten bestimmte Muster in Form von XPath-Ausdrücke entnommen, um alle Produktseiten in der XML-Datenbank zu adressieren. Im Endeffekt wird auf dieser Weise die Klassifizierung aller Dokumente nach Produkt- und Nicht-Produktseiten erreicht. Diese Methode ist im Abschnitt 6.5.3 zusammen mit einigen Experimenten präsentiert. Der letzte Schritt von dem Verfahren ist die Extrahierung der Produktdaten aus den erkannten Produktseiten. Dafür werden die gleichen Muster verwendet, die bei der Klassifizierung der Webseiten eingesetzt wurden. Das sind die XPath-Expression für die Adressierung der Produktnamen, Beschreibungen und Produktpreise. Im Abschnitt 6.5.4 wird, neben der Beschreibung des Verfahrens, ein Experiment für die Extrahierung der Produktdaten aus einem Shop mit der Hilfe der automatisch generierten Erkennungsmuster.

6.2 Erste Experimente

6.2.1 Informationsextrahierung mit Webharvest

Die ersten Versuche wurden mit einem Open-Source Tool für Extrahierung von Informationen aus Webseiten gemacht, namens *Web-Harvest*¹. Das Programm besitzt einen eingebauten Http-Client, der bestimmte Webseiten direkt vom Internet abrufen und aus ihnen Informationen entnimmt. Für jede Webseite wird eine Konfigurationsdatei erstellt, wo ähnlich wie mit einer funktionalen Programmiersprache, das Verhalten der Software definiert ist. Dort wird festgelegt, welche Seiten öffnet der Web-Client und welche Informationen zu extrahieren sind, die mit Hilfe von XPath-Abfragen im Quellcode lokalisiert werden. Das Programm wurde erfolgreich mit einigen Internetshops getestet. Mit Hilfe der Konfigurationsdatei wurde das Programm so eingestellt, dass es wie ein anwendungsspezifischer Webcrawler funktioniert, der bestimmte Links auf den Webshopseiten verfolgt und gezielt Informationen aus den besuchten Seiten entnimmt. Zuerst öffnet der Web-Client die Links der Hauptkategorien und danach besucht er rekursiv alle Unterkategorien. Wenn sich der Crawler in der letzten Unterkategorie befindet, entnimmt er gezielt die Produktdaten wie Produktname und Preis, die dort in einer tabellarischen Form aufgelistet sind. Manchmal passen nicht alle Produkte auf einer Seite und es muss gegebenenfalls die Blätterfunktion benutzt werden, um auf die weiteren Seiten zu gelangen und die dort aufgelisteten Produkte auszulesen. Schließlich werden die gefundenen Produktdaten in einer XML-Datei geschrieben. In Abbildung 6.2 wird grob der Ablauf des Datenextrahierungsprozesses mit Web-Harvest skizziert. Damit das Tool die richtige Links und Produktdaten findet, werden als Eingabe speziell konstruierten XPath-Ausdrücken übergeben, die auf der Webseiten die entsprechenden Stellen im Quellcode lokalisieren. Zusätzlich werden in der Konfigurationsdatei rekursive Funktionen für das Auslesen der Kategorien im Webshop definiert.

Das Tool ist schon sehr praktisch und für kleinere Datenextrahierungsaufgaben sicherlich sehr nützlich. Für einen produktiven Einsatz eignet sich dieses Programm auf Grund der extrem vielen Seiten in Webshops leider nicht, da oft die Stabilität nicht gewährleistet ist. Einer der Nachteile, der gegen eine Automatisierung spricht, ist dass für jeden Webshop eine neue Konfigurationsdatei manuell zu erstellen ist. Außerdem ist die Konfigurationssprache nicht sehr flexibel und erlaubt keine Wiederverwendbarkeit vom Code, was eine automatische Generierung der Konfigurationsdatei

¹Web-Harvest - <http://web-harvest.sourceforge.net/>

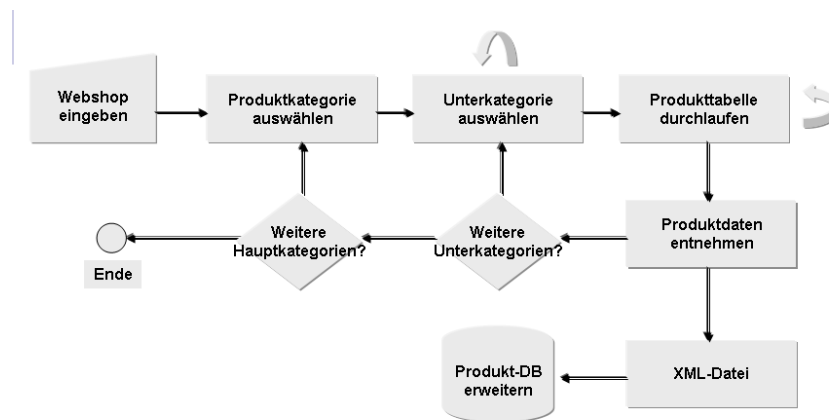


Abbildung 6.2: Produktextrahierung in einem Webshop mit Web-Harvest

praktisch unmöglich macht. Dennoch wurden viele Ideen und einige Modulen von *Web-Harvest* bei der Entwicklung einer ähnlicher Software übernommen, die mehr Stabilität und Flexibilität liefern sollte. Auf dieser Weise konnte eine volle Kontrolle über das Verhalten des Web-Clients erreicht werden, ohne die Einschränkungen der Konfigurationsdatei.

6.2.2 Erste Version eines Programms für Produktextrahierung

Die Grundidee war, eine Applikation zu erstellen, die völlig selbständig die Produktdaten aus jedem neuen Webshop ausliest. Das Programm sollte auf ähnlicher Weise wie Web-Harvest Informationen aus Webseiten extrahieren, mit dem Unterschied, dass es die Webshops analysiert und automatisch die notwendigen XPath-Expression generiert. Die automatische Analyse besteht aus mehreren Schritten und kann als eine „Bottom-up“-Analyse betrachtet werden. Ihr Ablauf ist auf Abbildung 6.3 schematisch dargestellt.

Als erstes wird die Hauptseite eines Onlineshops nach einem Suchformular durchsucht, indem mit Hilfe von regulären Ausdrücke nach üblichen Schlüsselwörtern (*search, find, such etc.*) in den Inputfeldern und Html-Formularen gesucht wird. Sobald etwas gefunden wird entdeckt, wird eine Suche nach einem bekannten Produkt mit der Hilfe des Web-Clients ausgeführt. Als Quelle für die Produktnamen, die bei der Suche verwendet werden, wird eine manuell zusammengestellte Produkttable verwendet, die auch nach erfolgreicher Produktextrahierung in einem neuen Webshop erweitert wird. Nachdem der Http-Client die Suche im Onlineshop ausführt, untersucht das Programm die vom Server zurückgelieferten Ergebnisseite. Normalerweise kommen verschiedene Artikel in tabellarischer Form, die dem Suchparameter entsprechen. Wird das gesuchte Produkt nicht entdeckt, wird eine neue Suche mit einem anderen Produktname gestartet. Die Prozedur wird so lange wiederholt, bis ein Treffer erkannt wird. Der Produktname sollte in der Ergebnisstabelle als ein Link zu der entsprechenden Produktseite vorkommen, der vom Http-Client gefolgt wird. Dabei muss berücksichtigt werden, dass der gesuchte Produktname in dem zu untersuchenden Webshop abweichen kann. Als nächstes analysiert das Programm die Folgeseite und merkt sich die Seitenstruktur, anhand derer später andere Produktseiten in diesem Webshop erkannt werden. Zusätzlich werden alle Links auf der untersuchten Produktseite und deren Folgeseiten der Reihe

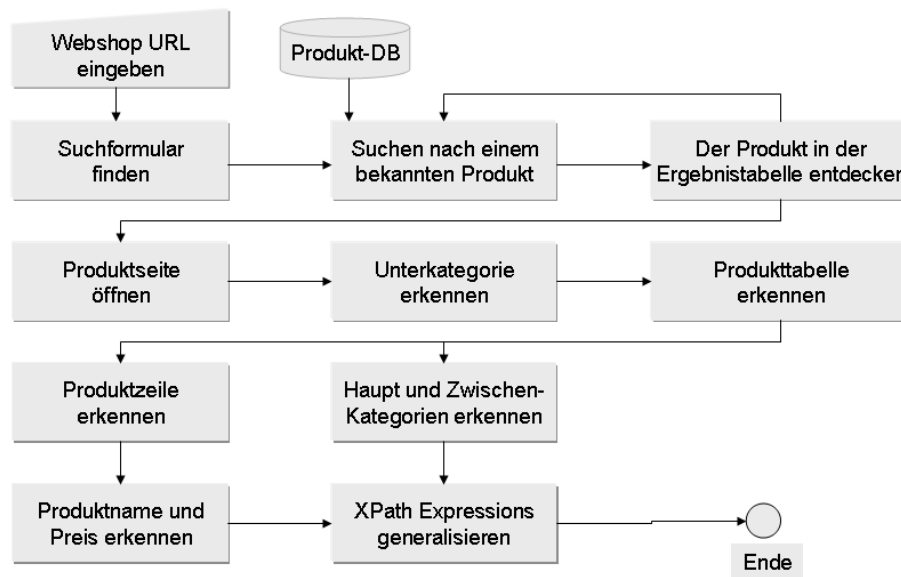


Abbildung 6.3: Ablauf der automatischen Generierung von XPath-Ausdrücke

nach untersucht. Das Ziel dieses Schrittes ist, den Link der Kategorie bzw. Unterkategorie zu finden, zu dem das Produkt auf der untersuchten Seite gehört. Dieser Ansatz basiert auf der Annahme, dass in den meisten Webshops eine Navigation über die Kategorien vorhanden ist. Meistens wird sie anhand eines Baumes dargestellt, so wie in Abbildung 6.4. Wenn die Produktseite geöffnet ist, wird mindestens ein Link zu der Kategorie angezeigt, zu der das ausgewählte Produkt gehört. Wird dieser Link angeklickt (in der Abbildung links mit Pfeil markiert), landet der Besucher auf die Kategorieauflistung. Auf der Seite werden alle Produkte in der ausgewählten Kategorie in einer Tabelle aufgelistet. Das Programm muss erkennen, dass es sich genau um die Produkttafel der Unterkategorie handelt, zu der das Produkt auf der Produktseite gehört. Das ist der Fall, wenn in der Produkttafel ein Link zu der vorher besuchten Produktseite existiert (Siehe Abbildung 6.4, das Link ist mit einem Pfeil markiert). In diesem Fall merkt sich das Programm die Position des Linkes zu der Unterkategorie, so wie die Position des Produktnamen in der Produkttafel. Diese Daten können als Basis für die Erkennung weiterer Produkten und deren Eigenschaften in der Produkttafel dienen, so wie für die Erkennung der Haupt- und gegebenenfalls Zwischenkategorien. Als erstes wird beschrieben, wie die XPath-Expressions für die Produktname, Beschreibung und Preis generiert werden. Anschließend wird erläutert, wie die Ausdrücke für die Produktkategorien in dem Webshop konstruiert werden.

Das Ziel ist, ein generalisierter XPath-Ausdruck automatisch zu bilden, der alle Elemente in der Produkttafel adressiert. Dafür wird zuerst der Name des gefundenen Produkts in dieser Tabelle lokalisiert (in Abbildung 6.4 rechts mit Pfeil markiert) und daraus einen absoluten XPath-Ausdruck gebildet, der Link-Knoten dieses Produktes adressiert. Der kann zum Beispiel folgendermaßen aussehen:

```
/html/body/table[2]/tbody/tr[3]/td/table/tbody/tr[2]/td[3]/a
```

The image shows two side-by-side screenshots of an e-commerce website. The left screenshot displays a detailed product page for a Sony Vaio laptop. The page features a navigation menu on the left with categories like 'Notebooks', 'Hardware', and 'Software'. The main content area includes a promotional banner for NEXOC, a product image of the laptop, and technical specifications such as 'Display: 13.3" (1.280x800)', 'CPU: INTEL Core 2 Duo T6400 2x 2.00GHz', and 'RAM: 4096MB (2x2GB)'. The price is listed as 1149,00 € with a discount of 8,00 €. The right screenshot shows a list of products under the heading 'Mobile Kraftpakete'. It features a navigation menu on the left and a list of products with their respective specifications and prices. The products listed include Sony Vaio VGN-SR41M/P, Sony Vaio VGN-SR41M/W, and Sony Vaio VGN-SR41M/N.

Abbildung 6.4: Beispiel für Produktseite mit Kategorienavigation (links) und Produkttable (rechts). (Quelle: hoh.de)

Da in der Produkttable mehrere Zeilen mit gleicher Struktur existieren, wird der absolute XPath-Ausdruck so modifiziert, dass er alle Zeilen in dieser Tabelle adressiert. Das wird durch Wegnahme der Prädikaten bei bestimmten Lokalisierungsschritten von dem Pfadausdruck realisiert. Das Programm liest von rechts nach links, entfernt jeweils eine der Positionen und analysiert dabei die Ergebnisse der XPath-Abfrage. Am Ende wird die Position entfernt, deren Wegnahme die meisten Treffer erzielt hat. Konkret wird bei dem XPath-Ausdruck von dem Beispiel die Position bei dem letzten TR-Tag entfernt:

```
/html/body/table[2]/tbody/tr[3]/td/table/tbody/tr/td[3]/a
```

Bei dem nächsten Schritt versucht das Programm einen XPath-Ausdruck zu bilden, der nicht nur den Produktnamen in der Tabelle adressiert, sondern die komplette Zeile samt Photo, Beschreibung und Produktpreis. Dazu muss der XPath-Ausdruck wieder modifiziert werden, indem die Lokalisierungsschritte von rechts nach links schrittweise entfernt werden. Dabei werden sie so lange entfernt, bis der letzte Schritt erreicht wird, bei dem der XPath mehr oder gleich viele Ergebnisse liefert wie am Anfang. Die entfernten Elemente werden dabei als Bedingung gesetzt. Das wird mit dem Ziel gemacht möglichst viele Elemente von den einzelnen Produktzeilen zu markieren. In dem Beispiel von oben werden nur die letzten zwei Schritte entfernt.

```
/html/body/table[2]/tbody/tr[3]/td/table/tbody/tr/td[3]/a]
```

Zusätzlich werden die einzelnen Zeilen aus der Produkttable analysiert um die Beschreibung und der Preis zu lokalisieren und anschließend relative XPath-Ausdrücke automatisch zu konstruieren. Der Preis kann mit Hilfe von RegExps erkannt werden, und die Beschreibung anhand des in der Produktdatenbank vorhandenen Textes.

Als nächstes werden von dem Programm die XPath-Ausdrücke für die Kategorien automatisch generiert. Als Ausgangspunkt dient dazu der Link von der erkannten Unterkategorie (siehe Abbildung 6.4). Wenn die Website als ein Graph betrachtet wird, wo die Webseiten die Knoten sind, wird der kürzeste Weg von dem Knoten *Hauptseite* zum Knoten *Unterkategorie-X*, wobei *Unterkategorie-X* die vorher erkannte Unterkategorie ist. Der Ansatz basiert auf einer Art *Bruteforce-Methode* und überprüft alle Links ausgehend von der Hauptseite und merkt sich den kürzesten Pfad, der gefunden wurde. Die Länge dieses Pfades wird zusätzlich begrenzt, damit der Crawler nicht zu tief in den Webshop-Seiten herum sucht. Der ausgerechnete Pfad, der aus URLs der gesuchten Kategorien besteht, dient als Eingabe für die Methode, die die XPath-Ausdrücke der Kategorien generiert. Zum Beispiel kann der Pfad folgendermaßen aussehen:

```
[http://shop.com/EDV]->[http://shop.com/Monitore]->[http://shop.com/22Zoll]
```

Dabei wird für jede Kategorieebene eine eindeutige Expression gebildet. Der Algorithmus dafür funktioniert ähnlich wie bei der Bildung der Expression für die Produktnamen in der Produkttabelle. Als erstes wird der Link von dem ersten Pfadelement, das als die Hauptkategorie betrachtet werden kann, auf der Hauptseite lokalisiert und einen absoluten XPath-Ausdruck generiert. Danach wird der Ausdruck so generalisiert, dass auch die anderen Hauptkategorien von ihm adressiert werden. Das gleiche wird auch für die weiteren Unterkategorien gemacht.

Die Software wurde erfolgreich mit einigen Webshops wie *T-Online Shop*² und ³ getestet und hat mit einigen gezielten Eingriffen im Code auch funktioniert. Der größte Vorteil dieses Ansatzes ist, dass nicht alle Webseiten in dem Internetshop lokal für die Analyse vorhanden sein müssen. Nach der Generierung der XPath-Ausdrücke können alle Produkte aus der Produkttabellen ausgelesen werden, die leicht über die Kategorienavigation zu erreichen sind. So werden nicht unnötig viele Seiten besucht und dadurch eine gute Performanz erreicht. Leider hat dieses Verfahren mehr Nachteile als Vorteile. Der größte davon ist die fehlende Flexibilität durch das Voraussetzen einer bestimmten Webshop-Struktur. Damit das Programm die XPath-Ausdrücke richtig erkennen kann, muss die Website über einer Vielzahl an Funktionalitäten verfügen, unter anderem: Suchformular auf der ersten Seite, Kategorienavigation, die auf allen Seiten sichtbar ist, und verlinkte Produktnamen in der Produkttabelle. Darüber hinaus sollten keine Links auf der Hauptseite existieren, wie zum Beispiel *Top-Angebote* oder weitere *Shortcuts*, die den Algorithmus für die Errechnung des Kategoriepfades negativ beeinflussen. Im Allgemeinen besteht das beschriebene Verfahren aus sehr vielen Einzelschritten und demzufolge ist es extrem Fehleranfällig. Aus diesem Grund wurde an einer alternativen Lösung gearbeitet, die flexibel genug und für die meisten und bekanntesten Webshops geeignet ist.

6.3 Datenaufbereitung

In der Phase der Datenaufbereitung spielt der selbst entwickelte Webcrawler die Hauptrolle (siehe Unterkapitel 5.2). Als Eingabe dienen die Internetadressen der Onlineshops, die für die Experimente ausgewählt wurden. Ausgehend von diesen Seiten wird der Quellcode gespeichert und nach Hyperlinks durchsucht, die als Ausgangspunkt für das Einlesen weiterer Webseiten auf der

²T-Online Shop - <http://www.t-online-shop.de>

³Redcoon - <http://www.redcoon.de>

besuchten Website dienen. Damit das Programm keine externen Verweise verfolgt und nur Webseiten innerhalb der gleichen Domain aufruft, werden die Adressen der Links analysiert und nur solche verwendet, die auch den Domainnamen beinhalten. Um die Anzahl der nicht relevanten Seiten und gleichzeitig die Datenvolumen auf dem Server zu reduzieren, wurde der Crawler zusätzlich dahingehend erweitert, dass er die Link-Parameter (*rel=nofollow*) für die Suchmaschinen interpretiert und so entsprechende Seiten überspringt. Alle Experimente mit dem Crawler-Modul wurden im Rechenzentrum der Universität Dortmund durchgeführt, das über eine fast uneingeschränkte Internetverbindung verfügt, was die Performanz der Anwendung deutlich steigerte. Der eingesetzte Server verfügt über 4GB RAM und einen AMD64-Dual-Core-Prozessor, die Plattform ist Linux.

Wie bereits in Unterkapitel 5.2 gezeigt, wurde dieses Modul als eine Multi-Threaded-Applikation entwickelt, was das gleichzeitige Bearbeiten von mehreren Websites ermöglicht. Dieses Vorgehen hat den besonderen Vorteil, dass der Crawler nicht von dem niedrigen Durchsatz eines Webshops beeinträchtigt wird. Der Crawler ist so konzipiert, dass für jede auszulesende Website ein Thread gestartet wird. Es wurde auch ein Versuch mit mehreren Threads pro Webshop gestartet mit dem Ziel, die Performance zu optimieren. Allerdings hat dies genau das Gegenteil bewirkt, denn die Firewalls der meisten Websites haben dieses Vorgehen auf Grund der vielen Requests als eine Attacke gewertet und verschiedene Schutzmechanismen aktiviert. Einige Webshops reagierten mit künstlich verzögerten Serverantworten, andere lieferten Leerseiten oder Fehler-Responses wie *HTTP Error 500 (Internal server error)*. Das gleiche Verhalten wurde in einigen Fällen sogar mit einem Thread pro Website beobachtet. Angesichts des Servers, auf dem die Experimente mit dem Crawler durchgeführt wurden, können Limits bei der Bandbreite und die Verbindungszeiten vernachlässigt werden, da die Universität Dortmund über eine sehr gute Internetkonnektivität verfügt. Das hat höchstwahrscheinlich dazu geführt, dass der Crawler mehr Http-Requests zum Webshop gestartet hat, als der Webserver erlaubt. Um dieses Problem zu umgehen, wurden die Requests künstlich auf zwei pro Sekunde gedrosselt. Da die Webserver von einigen Websites zudem die Requests pro Client und Minute begrenzen, wurden bei den Threads zusätzlich in bestimmten Abständen eine Pause eingelegt, was bei allen getesteten Shops das Problem gelöst hat.

Webshop	Seiten	Größe	Zeit	Seiten/Min	Codierung
T-Online Shop	8465	568 MB	6:10 Std.	≈ 23	utf-8
Notebooksbilliger	1613	239 MB	1:22 Std.	≈ 11	iso-8859
Power Netshop	6863	235 MB	4:05 Std.	≈ 28	iso-8859
HOH.de	1349	174 MB	2:10 Std.	≈ 10	utf-8
CSV-Direkt	5423	464 MB	2:30 Std.	≈ 36	iso-8859
Redcoon.de	663	41 MB	0:23 Std.	≈ 29	utf-8
PC-Planet	3592	1500 MB	5:30 Std.	≈ 11	iso-8859
NotebookNet.de	1468	66 MB	1:55 Std.	≈ 13	iso-8859

Tabelle 6.1: Liste der getesteten Internetshops inklusive Auswertung

Für die Versuche wurde eine Reihe von Internetshops ausgewählt, die ein ähnliches Produktangebot aufweisen, aber von unterschiedlicher Größe sind. In Tabelle 6.1 werden alle Webshops aufgelistet sowie die Anzahl der ausgelesenen Seiten und die Zeit, die diese Phase in Anspruch

genommen hat. Die Anzahl der Seiten entspricht nicht immer der tatsächlichen Anzahl der im Internet verfügbaren Seiten des jeweiligen Internetshops. Dieser Wert gibt vielmehr an, wie viele Seiten der Crawler tatsächlich erreicht und gespeichert hat. Um das Datenvolumen in Grenzen zu halten, wurden zusätzliche Einschränkungen festgelegt. Zum einen wurde die maximale Anzahl an Links auf einer Seite auf 200 begrenzt, zum anderen die Tiefe der Seiten (Entfernung von der Hauptseite) auf 10 Seiten limitiert. Bei einigen der Webshops wurde nach einer gewissen Zeit trotz der eingeführten Drosselung der Requests der Crawler blockiert.

Zusätzlich wurde das automatische Erkennen der Textcodierung vorgenommen, die auf den vom Server gelieferten Header-Informationen basiert. In allen Fällen wurde sie richtig erkannt, wie Tabelle 6.1 zeigt. Damit die Produktdaten später einheitlich gespeichert werden können und leicht zu vergleichen sind, wird jede Webseite sofort nach dem Auslesen in *Unicode* (UTF-8) konvertiert. Im folgenden Bearbeitungsschritt werden dann die Webseiten, die im Rohformat (HTML) vom Crawler geliefert wurden, in XML konvertiert. Dabei wird der manchmal unsaubere HTML-Code in ein gültiges und wohlgeformtes XML-Format überführt. Diese Aufgabe wird von einem externen Tool namens *HtmlCleaner* übernommen, das in Unterkapitel 5.3 detailliert beschrieben wird. Es hat die Aufgabe, alle HTML-Tags gemäß den XML-Spezifikationen in die richtige Form und Reihenfolge zu bringen und gegebenenfalls die in XML ungültigen Zeichen zu berücksichtigen. Zusätzlich wird das Tool verwendet, um unnötige Quellcode-Fragmente von den Seiten zu entfernen, unter anderem Style- und Dekorationsinformationen, Java-Scripte und Kommentare. Im Allgemeinen hat sich der *HtmlCleaner* als sehr zuverlässig und stabil herausgestellt und kein einziges Problem bei der Verarbeitung der ca. 30.000 untersuchten Webseiten verursacht, sondern stets gültige XML-Dokumente generiert.

Nachdem die Webseiten in das XML-Format konvertiert worden sind, werden sie in der in Unterkapitel 5.4 vorgestellten XML-Datenbank namens *BaseX* gesichert. Dabei wird zusammen mit dem ersten XML-Dokument automatisch eine neue Datenbank für den untersuchten Onlineshop angelegt. Wie beschrieben, kann die Kommunikation mit *BaseX* über die eingebauten API oder alternativ im Server-Client-Modus erfolgen, indem diese Applikation auf einen bestimmten Server ausgelagert wird. Bei den hier beschriebenen Experimenten haben wir uns aus Zeitgründen für das einfachere Modell entschieden, so dass die Datenbank auf dem gleichen Rechner installiert wird, auf dem auch der Crawler ausgeführt wird. Die Struktur einer XML-Datenbank kann als ein Baum repräsentiert werden, der initial nur einen Knoten besitzt - die Wurzel *root*. Die Webseiten, wie jedes andere XML-Dokument, besitzen auch eine Baumstruktur, wobei der Knoten *html* die Rolle der Wurzel spielt. Beim Hinzufügen einer neuen Seite zu *BaseX* wird für sie eine laufende Nummer (ID) vergeben, die zusammen mit der URL der Seite als Parameter eines neuen Wurzelknotens angegeben und anschließend als ein Teilbaum in der XML-Datenbank angehängt werden kann. In unserem Projekt wurden zwei verschiedene Ansätze für die Verwaltung der Webseiten in *BaseX* und für die innere Struktur der Dokumente evaluiert. Es besteht die Möglichkeit, die XML-Datenbank hierarchisch aufzubauen, indem die neuen Dokumente als Unterbaum an dasjenige Dokument anzuhängen sind, von dem aus das neue Dokument vom Crawler erreicht wurde. In dem Beispielgraph in Abbildung 6.5 repräsentiert *Page0* die Hauptseite, und die Seiten *Page1*, *Page190* sowie *Page456* wurden direkt von *Page0* erreicht. Dieses Vorgehen hat den Vorteil, dass die Seiten in der XML-Datenbank verschiedene Ebenen bilden, anhand deren bei einer späteren Analyse gleichartige Seiten erkannt werden können. Zusätzlich können die hierarchischen Beziehungen dazu beitragen, auch thematische Zusammenhänge zwischen den Seiten, die sich in dem

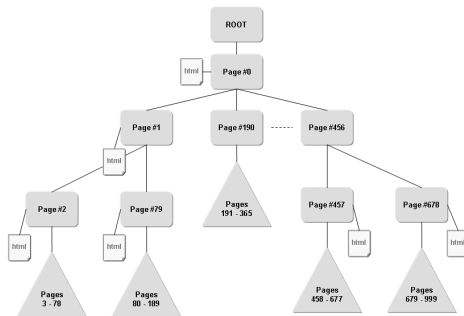


Abbildung 6.5: Hierarchische
Dokumentenstruktur.

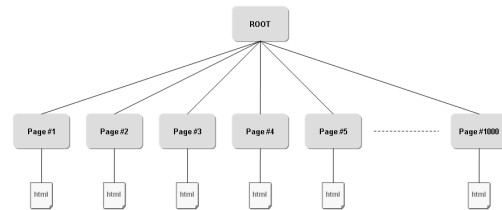


Abbildung 6.6: Flache Dokumentenstruktur.

gleichen Teilbaum befinden, zu erkennen und so die Dokumente in Kategorien zu ordnen.

Neben den Vorteilen haben sich während der Versuche aber auch eine Reihe von Nachteilen ergeben. Zum einen wächst die Tiefe des XML-Baumes sehr schnell, was die BaseX-Anwendung instabil zu machen und für eine schlechtere Performance zu sorgen scheint. Zum anderen machen die hierarchischen Beziehungen zwischen den Seiten die Struktur für einige der verwendeten XPath-Abfragen eher ungeeignet. Aus diesen Gründen haben wir uns für eine alternative Lösung entschieden, bei der die XML-Dokumente flach gespeichert werden. Anstatt die Seiten an deren Referenzseiten im XML-Baum anzuhängen, bekommen sie alle die Wurzel der Datenbank als Elternknoten. Abbildung 6.6 stellt die Struktur anschaulich dar. Damit die Informationen über die Referenzseite nicht verloren gehen, werden deren IDs als Parameter in dem *page*-Knoten angegeben. Mit diesem Modell wurde eine merkbare Steigerung der Performance erreicht, wobei bestimmte XPath-Abfragen mehrfach schneller ausgeführt wurden. Auch die Fehlerquote beim Hinzufügen von neuen Seiten wurde minimiert.

Nachdem alle erreichbaren Webseiten ins XML-Format konvertiert und in der BaseX-Datenbank gespeichert wurden, werden die Webseiten analysiert und auf häufig vorkommende Elemente hin durchsucht. Bestimmte Teilbereiche der Quellcodes, die bei jeder Webseite an der gleichen Stelle zu finden sind, liefern keine interessante bzw. relevante Informationen und können entfernt werden. Ein Beispiel für solche Elemente sind dekorative Html-Tags, wie Bilder oder Tabellen. Manchmal können größere Teile der Seite ohne Informationsverlust entfernt werden, wie zum Beispiel Kopf- und Fußzeilen. Das Hauptziel ist hier die Vereinfachung des XML-Baumes in der Datenbank sowie die Reduzierung des Datenvolumens.

Der letzte Schritt in der Datenaufbereitungsphase ist das Indexieren der Seiteninhalte. Das Ziel ist es, eine schnelle und leistungsfähige Suchmöglichkeit für die Textinhalte in der XML-Datenbank zu ermöglichen. In unserem Projekt wird diese Aufgabe von dem Open-Source-IR-System *Lucene* übernommen, das in Unterkapitel 5.5 vorgestellt wurde. Für jeden Webshop in der BaseX-Datenbank wird eine Index-Datenstruktur mit Hilfe der Lucene-API erstellt und verwaltet. Nachdem alle Webseiten erfolgreich in der XML-Datenbank aufgenommen wurden, werden

mittels einer XPath-Abfrage die Textinhalte jeder Seite extrahiert und als Dokument in einer Indexdatenstruktur aufgenommen. Das hat einige Vorteile gegenüber dem direkten Suchen in der BaseX-Datenbank. Erstens ist die Datenmenge im Lucene-Index viel geringer, weil dort nicht der komplette Quellcode der Webseite, sondern nur die Textinhalte vorhanden sind. Das optimiert die ohnehin schon performanten Suchfunktionalitäten mit Lucene und macht zusätzlich die Suche durch das Entfernen der HTML-Tags noch präziser. Zweitens bietet Lucene verschiedene Suchmöglichkeiten, die von BaseX nicht unterstützt werden, wie *Fuzzy*- und *Distanzsuche*. In Tabelle 6.2 werden die Ergebnisse der durchgeführten Versuche mit den ausgewählten Webshops präsentiert. Durchschnittlich wird eine Komprimierungsrate von 90 % erreicht, wobei sich die Indexierungsfunktionen von Lucene mit durchschnittlich 40 MB/Sek. als sehr performant erwiesen haben.

Webshop	Seiten	Größe XML	Größe Index	Zeit	MB/s
T-Online Shop	8465	568 MB	≈ 57 MB	10 Sek.	≈ 56
Notebooksbilliger	1613	239 MB	≈ 24 MB	6 Sek.	≈ 40
Power Netshop	6863	235 MB	≈ 21 MB	8 Sek.	≈ 29
HOH.de	1349	174 MB	≈ 9 MB	5 Sek.	≈ 34
CSV-Direkt	5423	464 MB	≈ 48 MB	14 Sek.	≈ 36
Redcoon.de	663	41 MB	≈ 7 MB	2 Sek.	≈ 20
PC-Planet	3592	1500 MB	≈ 51 MB	19 Sek.	≈ 78
NotebookNet.de	1468	66 MB	≈ 6 MB	2 Sek.	≈ 33

Tabelle 6.2: Auswertung der Ergebnisse aus der Indexierung der Internetshops mit Lucene

6.4 Manuelle Produktextrahierung

Wie bereits am Anfang des Kapitels erwähnt, wird für die automatische Produktextrahierung eine Trainingsmenge von Beispieldaten notwendig. Anstatt die Produkte per Hand in der MySQL-Datenbank einzugeben oder eine Produktliste zu importieren, werden die Beispielsprodukte aus den bereits in der XML-Datenbank befindlichen Webshops entnommen. Dafür müssen die Internetshops zuerst manuell analysiert werden. Der Benutzer navigiert so lange auf den Seiten des Shops, bis er auf eine Produktseite stößt. Dort wird in den meisten Fällen auf einer Seite ein einzelnes Produkt detailliert samt Informationen wie Beschreibung und Preis vorgestellt. Anhand der URL-Adresse wird die entsprechende Seite in der XML-Datenbank aufgerufen. Der Benutzer analysiert die Quellcode-Struktur und lokalisiert die Stellen im Code, wo die unterschiedlichen Produktdaten positioniert sind. Dann wird ein XPath-Ausdruck konstruiert, der die Position des Produktnamens auf der Seite adressiert. Dieser Pfad wird manuell durch Einsetzen von Wildcards so modifiziert, dass damit möglichst viele Produktbezeichnungen in der XML-Datenbank ausgewählt werden können. Im Endeffekt kann der konstruierte XPath als Muster für die Erkennung und Adressierung aller Produktseiten der entsprechenden XML-Datenbank dienen. Es wird anschließend für die Erstellung eines speziellen *Produktseiten*-XPath-Ausdrucks verwendet, der alle *page*-Knoten der Produktseiten liefert. Der *Produktname*-XPath-Ausdruck wird dabei als Prädikat eingesetzt, so dass der *Produktseiten*-XPath alle XML-Teilbäume (Dokumente) selektiert, für die der *Produktname*-XPath gilt. So kann er verwendet werden, um die XML-Dokumente in der Datenbank nach Produktseiten und Nicht-Produktseiten zu klassifizieren und den Quellcode von allen

Produktseiten zu extrahieren. Mit Hilfe der *ProductImporter*-Module werden dann die Produktdaten aus den XML-Dokumenten entnommen und in der relationalen Datenbank angelegt. Wie in Unterkapitel 5.8 beschrieben, werden in der Produkttabelle drei Attribute pro Produkt eingefügt - der Produktname, die Beschreibung und der Preis. Für jeden Parameter wird ein XPath-Ausdruck erstellt, anhand dessen die Position des entsprechenden Attributs auf der Produktseite adressiert wird. Für den Produktnamen wurde bereits im vorherigen Schritt eine Expression konstruiert, die für die Erkennung der Produktseiten verwendet wird. Es bleibt jeweils ein XPath-Ausdruck für die Position der Beschreibung und des Produktpreises zu konstruieren, was analog zu der Bildung der Produktname-XPath-Expression abläuft. Schließlich werden die per Hand konstruierten XPath-Ausdrücke als Parameter an das *ProductImporter*-Modul übergeben, das den Rest erledigt. Im Erfolgsfall werden die Daten aller Produkte samt deren Attributen in der Produkttabelle der MySQL-Datenbank gespeichert.

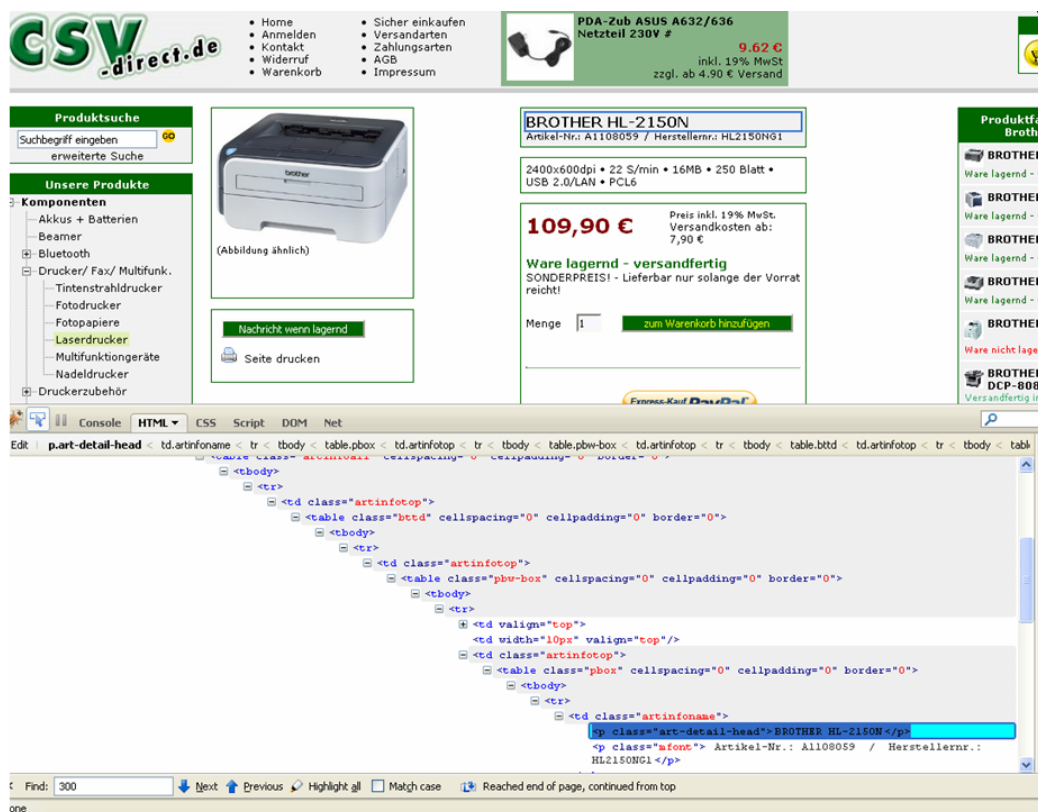


Abbildung 6.7: Manuelle Konstruierung von einem XPath-Ausdruck für den Produktname

Im Folgenden wird der Prozess der manuellen Produktextrahierung anhand eines Beispiels erläutert. Für den Versuch eignet sich der bereits ausgelesene Webshop *CSV-Direkt*⁴ auf Grund der hohen Anzahl gecrawlter Seiten in der XML-Datenbank. Bei der Suche nach einer Produktseite habe ich meine Auswahl zufällig auf einer Seite gestoppt, die folgende Produktdaten präsentiert:

⁴CSV-Direkt - <http://www.csv-direct.de/>

Produktname	BROTHER HL-2150N
Produktpreis	109,90
Beschreibung	Bis zu 22 Seiten pro Minute... (<i>verkürzt</i>)

Abbildung 6.7 zeigt einen Screenshot der Webseite, wo die Stelle des Produktnamens markiert ist. Unterhalb der Seite wird anschaulich einen Auszug aus dem Quellcode der Webseite angezeigt, der eine Idee von der Konstruktion der XPath-Ausdrücke vermittelt, die in der folgenden Tabelle aufgelistet sind:

Produktseite	<code>//page[//td[@class='artinfoname']/p[@class='art-detail-head']]</code>
Produktname	<code>//td[@class='artinfoname']/p[@class='art-detail-head']</code>
Produktpreis	<code>//p[@class='article-price']</code>
Beschreibung	<code>//div[@id='tabcontent1']/div[@class='artinfotech']</code>

Die fertigen XPath-Ausdrücke werden dem Modul *ProductImporter* als Parameter übergeben. Es hat die Aufgabe, anhand der *Produktseiten*-XPath-Expression den Quellcode aller Produktseiten von der XML-Datenbank abzufragen. Auf jede einzelne Webseite, die als XML-Dokument repräsentiert wird, werden die XPaths für die einzelnen Attribute angewendet. Da die Beschreibung auf mehrere Textknoten verteilt werden kann, wird der letzte Lokalisierungsschritt so angepasst, dass der Textinhalt aller Nachkommenknoten geliefert wird. Bei der Extrahierung des Preis-Attributs gibt es die Besonderheit, dass der XPath-Ausdruck einen XML- beziehungsweise HTML-Code liefert, in dem der Preis erkannt werden muss. Das geschieht mit Hilfe von regulären Ausdrücken und wird in Unterkapitel 5.7 beschrieben. Wenn damit keine Zahlen im Währungsformat erkannt werden können, wird vom *ProductImporter* die Prozedur zur Erkennung von Preisen durchgeführt, die mit nebeneinander positionierten Bildern dargestellt werden. Tabelle 6.3 gibt eine Übersicht über die Anzahl der Produkte, die mit diesem Ansatz in den untersuchten Webshops entdeckt worden sind.

Webshop	Alle Seiten	Produktseiten
T-Online Shop	8465	3377
Notebooksbilliger	1613	531
Power Netshop	6863	2517
HOH.de	1349	745
CSV-Direkt	5423	4379
Redcoon.de	663	317
PC-Planet	3592	1469
NotebookNet.de	1468	82

Tabelle 6.3: Auswertung der Ergebnisse aus der manuellen Produktextrahierung

6.5 Automatische Produktextrahierung

Das entwickelte automatische Verfahren für Produktextrahierung ist ähnlich strukturiert wie das manuelle und besteht grundsätzlich aus denselben Bearbeitungsschritten, die allerdings automatisiert sind. Das Hauptziel besteht darin, die Muster für die Erkennung der Produktseiten und Produktdaten in einem noch nicht untersuchten Webshop automatisch zu generieren. Dafür müssen zuerst die lokal gespeicherten Webseiten nach Produkt und Nicht-Produktseiten klassifiziert

werden, da nur die Produktseiten die meist detaillierten Produktinformationen beinhalten. Um das zu erreichen wurden in dieser Arbeit zwei verschiedene Ansätze verfolgt.

Der erste Versuch in dieser Richtung wurde mit *RapidMiner*⁵ gemacht, ein Open-Source *Data Mining* System, das ursprünglich am Lehrstuhl für Künstliche Intelligenz der Universität Dortmund entwickelt wurde. Die Idee hinter diesem Experiment ist, das in RapidMiner eingebaute *SVM-Modul*⁶ für die Klassifizierung der Webseiten in einem neuen Webshop einzusetzen. Die genaue Vorgehensweise und die Ergebnisse werden in Abschnitt 6.5.1 genauer beschrieben und erläutert.

Der zweite und deutlich erfolgreichere Versuch geht in eine andere Richtung um die Produktseiten zu erkennen. Der Ablauf dieser Phase wird grob in Abbildung 6.1 (automatische Produktextrahierung) skizziert. Die Experimente basieren auf dem im Kapitel 4.6 vorgestellten Ansatz und verwendet Quellcode Fragmente aus der Produktseiten als Muster für die Seitenklassifizierung. Um diese Ausdrücke zu konstruieren, braucht das System eine Menge von Produktseiten, die es selbst generiert. Dazu wird gezielt nach bekannten Produkten in dem untersuchten Webshop gesucht. Als Quelle für die Produktdaten dient die Produkttabelle in MySQL. Im Unterkapitel 6.5.2 werden Experimente mit verschiedenen Abfragearten und Webshops beschrieben und ausgewertet. Jeder der Treffer, dessen Score höher als eine definierte Untergrenze ist, wird als eine potenzielle Produktseite betrachtet. Anschließend werden die gelieferten Seiten analysiert, um die Nicht-Produktseiten auszufiltern. Danach werden die Muster aus den Produktseiten entnommen und auf alle Seiten in der XML-Datenbank angewendet, um sie nach Produkt- bzw. Nicht-Produktseiten zu klassifizieren. Diese Methode wird anhand eines ausführlichen Experiments mit einem neuen Webshop in Unterkapitel 6.5.3 vorgestellt. Schließlich wird in Unterkapitel 6.5.4 ein Versuch gemacht, die automatisch-generierten XPath-Expressions für die Extrahierung und Importierung der Daten in der MySQL-Datenbank auf dem zuvor untersuchten Webshop anzuwenden. Danach wird die Menge der manuell gewonnenen Produktdaten mit der Menge der automatisch berechneten Produktdaten gegenübergestellt.

6.5.1 Klassifizierung der Webseiten mit RapidMiner

RapidMiner ist eine Umgebung für die Durchführung von Experimente im Bereich des Maschinellen Lernens und Data Mining [MWK⁺06]. Das System unterstützt eine Vielzahl an Methoden, die in den erwähnten Bereichen zum Einsatz kommen. Für jede der unterstützten Methode ist ein Operator in RapidMiner definiert, der in den einzelnen Experimenten instanziiert und konfiguriert werden kann. Sie können leicht vom Benutzer mit Hilfe der RapidMiner-GUI zusammengestellt werden (siehe Abbildung 6.8) und können aus beliebig vielen Operatoren aufgebaut sein, die die einzelnen Bearbeitungsschritte darstellen. Einige der Operatoren lassen sich auch als Subroutinen in anderen Operatoren einbetten und gruppieren. Auf dieser Weise wird das Experiment hierarchisch als ein Baum dargestellt.

Das Ziel des Experimentes ist, die Webseiten in einem neuen Internetshop, die als XML-Dokumente in der XML-Datenbank vorliegen, zu analysieren in zwei Klassen einzuordnen - *Produktseiten* und *restliche Seiten*. Als Ausgangsbasis für die Analyse dient eine Menge von bereits klassifizierten

⁵RapidMiner - <http://rapid-i.com/>

⁶Support Vector Machine

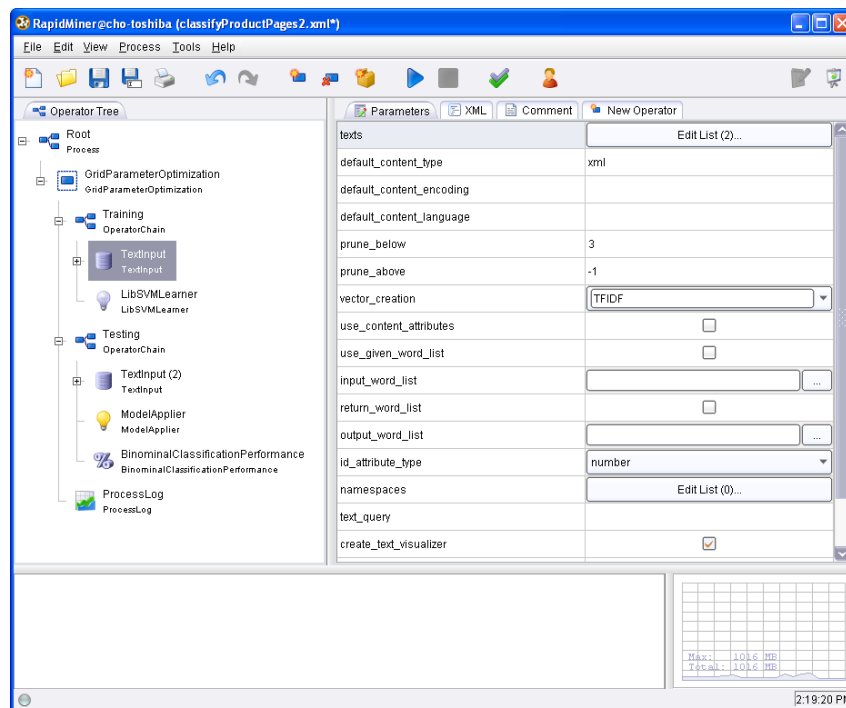


Abbildung 6.8: Beispielsperiment für die Klassifizierung der Seiten innerhalb eines Webshops

Webseiten stammend von einem anderen Webshop, die als Trainingsmenge zu betrachten ist. Das Experiment besteht aus zwei Hauptschritte (Training und Testing) und mehreren Unterschritten, die im Folgenden zusammen mit den entsprechenden Operatoren erläutert werden:

- *TextInput(1)* - Mit diesem Operator werden die bereits manuell klassifizierten Webseiten vom RapidMiner eingelesen. Die Daten wurden vorher als HTML-Dateien in zwei getrennten Verzeichnissen gespeichert, wobei alle Produktseiten in dem einen und alle restlichen Seiten in dem anderem Verzeichnis eingeteilt werden. Anschließend wurden die Dokumente in Vektorraum überführt, indem jedes Dokument als Vektor präsentiert wird. Vorher wurde der Text für die Vektorerstellung anhand verschiedener Textoperationen optimiert. Als erstes wurde der Text tokenisiert und danach durch ein Stoppwort-Filter und Tokenlänge-Filter gefiltert. Zusätzlich werden N-Gramme aus dem Text generiert mit der Hilfe von dem Operator *TermNGramGenerator*. Als Schema für die Vektorraumdarstellung wurde die *TF/IDF* ausgewählt, die in Unterkapitel 4.4.4 vorgestellt wurde.
- *SVMLEARNER* - In diesem Schritt wird ein SVM-Model generiert. Als Ausgangsbasis wird die vom ersten *TextInput* Operator generierte Trainingsmenge verwendet. Die grundlegende Idee hinter der *Support Vector Machines* ist die mathematische Berechnung einer Hyperebene im Vektorraum, die die einzelnen Vektoren aus der Trainingsmenge in zwei Mengen aufteilt. Auf dieser Weise wird eine „Grenze“ definiert, die die Trainingsbeispiele in zwei Mengen aufteilt. Nach der Trainingsphase wird die berechnete Ebene für die Klassifizierung von neuen Dokumenten bzw. Vektoren verwendet, die zum Vektorraum nachträglich hinzugefügt werden. Dieser Operator ist der Letzte aus der Trainingsphase. In den nächsten

Schritten wird die Ausführung und Evaluierung der SVM-Methode beschrieben.

- *TextInput(2)* - Der zweite *TextInput* Operator markiert den Anfang der Testphase in diesem Experiment. Hier wurden die Webseiten aus dem neuen Webshop eingelesen, die von dem SVM-Model zu klassifizieren sind. Die Dokumente wurden vorher durch manuell erstellten Muster in der XML-Datenbank klassifiziert und anschließend als HTML-Dateien exportiert. Genau so wie bei der Trainingsphase wurden die Dateien in zwei Verzeichnisse eingeteilt, zwecks einer Auswertung der SVM-Methode. Vor der Überführung im Vektorraum wurden die gleichen Textoperationen verwendet, wie bei *TextInput(1)*.
- *ModelApplier* - In diesem Schritt wird der generierte SVM-Model auf die Webseiten von dem neuen Internetshop angewendet. Als Eingabe bekommt der Operator *ModelApplier* das SVM-Model und die im vorherigen Schritt ausgelesenen Webseiten als Vektoren. Nach der Anwendung des SVM-Moduls wird diese Beispielsmenge von Vektoren in zwei Mengen klassifiziert. Im Erfolgsfall werden die Produktseiten und die restlichen Seiten aus dem neuen Webshop in unterschiedlichen Mengen eingeteilt.
- *BinomicalClassificatiionPerformance* - Mit diesem Operator wird die *Performance* der Klassifizierungsaufgabe ausgewertet. Als Eingabe bekommt der Operator eine Beispielsmenge, wo neben der berechneten auch die richtige Klassifizierung vorhanden ist. In diesem Experiment wird die richtige Klassifizierung von dem Operator *TextInput(2)* geliefert und mit der vom *ModelApplier* berechneten gegenübergestellt.

Bei diesem Experiment wurden echte Daten aus zwei Internetshops verwendet. Als Trainingsmenge wurden 775 Webseiten aus dem Webshop *Notebooksbilliger.de* verwendet, davon 275 Produktseiten. Als Eingabe für die Klassifizierungsaufgabe wurden 2436 Webseiten aus dem Webshop *Redcoon.de* verwendet, davon 1578 Produktseiten. Anschließend wurde ein neuer Versuch gestartet, wo die Seiten von *Redcoon.de* als Trainingsmenge dienen. Die Auswertung der Ergebnisse des Experimentes werden in der Tabelle 6.4 dargestellt. Mit der SVM-Methode konnten im ersten Versuch nur 120 der 1578 Produktseiten erkannt werden, in dem zweiten Versuch hat RapidMiner alle Seiten in die Klasse der Produktseiten eingestuft. Das kann als ein deutlicher Misserfolg betrachtet werden. Der Grund dafür liegt höchstwahrscheinlich darin, dass die Webseiten in den verschiedenen Webshops meistens unterschiedliche Quellcode-Struktur und Produktangebot haben, was die Klassifizierung ohne weiteres unmöglich macht.

Trainingsmenge	Zu klassifizieren	Precision	Recall
Notebooksbilliger	Redcoon	7.60%	99.17%
Redcoon	Notebooksbilliger	35.48%	100.00%

Tabelle 6.4: Ergebnisse von der Klassifizierung der Webseiten mit RapidMiner

6.5.2 Suche nach Produktseiten mit Lucene

Die mit der Methode der automatischen Produktextrahierung durchgeführten Versuche basieren auf einem in Abbildung 6.1 dargestellten Ablauf. In diesem Unterkapitel wird das Verfahren für das Generieren einer Menge aus potentiellen Produktseiten vorgestellt und ausgewertet. Die Grundidee besteht darin, nach Produktdaten aus der Trainingsmenge in dem Textinhalt der Webseiten

eines neuen Webshops zu suchen. Als IR-System kommt die Open-Source Software *Lucene* zum Einsatz. Für dieses Ziel wurden die Textinhalte bei der Datenaufbereitung in einer optimierten Datenstruktur indexiert. Anschließend wird der Index abgefragt und mit Hilfe eines Verweises das entsprechende XML-Dokument in der Datenbank ausgewählt. In einer produktiven Umgebung wird die Suche nur mit zufällig ausgewählten Produkten ausgeführt. Um die Genauigkeit dieses Verfahrens zu unter Probe zu stellen, werden dagegen in diesem Experiment alle verfügbaren Produktdaten in der MySQL-Datenbank als Quelldaten für die Suche verwendet. Das Experiment besteht aus zwei Versuchen. Bei dem ersten geht es darum, eine passende Abfrage zu konstruieren, die für die Suche in den meisten getesteten Webshops die besten Ergebnisse liefert. Dafür werden die manuell extrahierten Produkte in dem gleichen Webshop abgefragt. In dem zweiten Versuch wird die ausgewählte Abfrage auf einer großen Menge von Produkten ausprobiert, die von anderen Webshops stammen.

Konstruktion einer Abfrage für Lucene

Für diesen Versuch werden zwei unterschiedliche Lucene-Abfragetypen entworfen. In der ersten Abfrage wird eine reine Keyword-Suche vorgenommen, wobei den Keywords im Produktnamen mehr Gewicht zugesprochen wird (siehe Unterkapitel 5.5.2 - *Keyword-Boosting*). Die andere Abfrage verwendet für die Keywords im Produktnamen eine Distanzsuche und kombiniert sie mit einer gewöhnlichen Keywordsuche für die Beschreibung. Im Folgenden werden als Beispiel die zwei unterschiedlichen Abfragen für die Suche nach der Produktseite eines Produktes wiedergegeben:

Lucene-Abfrage mit Keyword-Boosting:

```
LCD^ TV^2 Panasonic^2 TX^2 L42S10E^2 107cm^2  
Full HD Diagonale 42\" 107cm Auflösung 1920x1080 Format 16:9  
Kontrast 50.000:1 dynamisch DVB-T Farbe Piano Black
```

Lucene-Abfrage mit Distanzsuche:

```
"LCD TV Panasonic TX L42S10E 107cm"~5 AND  
Full HD, Diagonale 42\" 107cm Auflösung 1920x1080 Format 16:9  
Kontrast 50.000:1 dynamisch DVB-T Farbe Piano Black
```

Um festzustellen, mit welcher Abfrageart die Suche bessere Ergebnisse erzielt, werden die beiden in einem Test allen indexierten Webshops gegenübergestellt. Jeder Datensatz aus der Produkttabelle wird im Lucene-Index abgefragt. Im Erfolgsfall wird die ID der gefundenen Seite im Datensatz gespeichert, um sie danach mit der Produktseite, auf der das Produkt ursprünglich bei der manuellen Informationsextrahierung gefunden wurde, zu vergleichen. In der Abbildung 6.9 wird die Auswertung der Suchergebnisse klar. Zu jedem Produkt existiert ein Verweis auf das Dokument in *BaseX*. In der Spalte *id-baseX-found* wird die BaseX-ID der gefundenen Produktseite gespeichert.

Der Versuch wird zuerst mit dem ersten Abfragetyp und dann mit dem zweiten durchgeführt, schließlich werden die Ergebnisse analysiert. Tabelle 6.5 zeigt eine Statistik über die Ergebnisse, die mit der *Keyword-Boosting* Abfrage erzielt wurden. Die Auswertung der Ergebnisse der Abfrage mit Distanzsuche werden in Tabelle 6.5 präsentiert. Die Spalte *P-Seiten* zeigt die Anzahl der Produktseiten, die mit dem manuellen Verfahren gefunden worden sind, während die Spalte

6 Experimente

id	id_shop	name	description	price	id_basex	id_basex_found
52635	104	Exquisit / GGV EGS 101 Weiss Einbau-Gefrierschrank - Neuware	·Farbe weiß·Energieeffizienzklasse A·En...	199.00	3239	NULL
52632	104	Beko HSA 37540 Gefriertruhe, 350 l Nutzinhalt - Neuware	·· Innenbeleuchtung - 4 Gefrierkörbe - Sü...	0.00	3229	3229
52633	104	Siemens GS 24 NV 21 Weiss - Neuware	·Verbrauchs-Eigenschaften·Energieeffiz...	0.00	3232	3232
52637	104	Exquisit / GGV UKS 140 RV Weiss Unterbau Kühlschr...	·Energieeffizienzklasse: A·Energieeffizie...	249.00	3245	NULL
52636	104	Exquisit / GGV UKS 130 Weiss Unterbau Kühlschr...	·Energieeffizienzklasse: A·Energieeffizie...	249.00	3242	NULL
52629	104	Beko FS 251 Weiss - Neuware	·· Freistehend - Transparente Fronten - B...	279.00	3220	3220
52630	104	Bosch GSD 26 N 10 Weiss - Neuware	·Verbrauchs-Eigenschaften·0,73 kWh En...	0.00	3223	3223
52631	104	Beko HSA 29520 - Neuware	·· Innenbeleuchtung - 3 Gefrierkörbe - Sü...	0.00	3226	3226
52626	104	Exquisit / GGV GS 235 A+ Weiss Gefrierschrank - Neuware	·Farbe: weiss·Energieeffizienzklasse: A+...	249.00	3211	3211
52625	104	Severin KS 9890 A+ Tischgefrierschrank - Neuware	·Energieeffizienzklasse: A+· - Energieeffiz...	183.00	3209	3209
52627	104	Bosch GSD 11 V 21 Weiss - Neuware	·Allgemeine Information·Sottline-Design...	259.00	3214	3214
52628	104	Exquisit / GGV GS 235 A+ Silber Gefrierschrank - Neuware	·Farbe: Silber·Energieeffizienzklasse: A+...	269.00	3217	3217
52623	104	Exquisit / GGV GS 115 S Weiss Gefrierschrank - Neuware	·Energieeffizienzklasse A·Energieeffizie...	179.00	3203	3203
52624	104	Exquisit / GGV GS 12 A+ Weiss Gefrierschrank, Türanschlag we...	·Farbe weiß·Energieeffizienzklasse A+· in...	179.95	3206	NULL
52619	104	WAECO T25 Dc Mobicool Kühlbox - Neuware	·Ausführung: Kühlbox·Leistungsmerkmale...	39.99	3189	NULL
52620	104	CWM 1350 Mini Cooler / Warmer rot - Neuware	·· 13,5 Liter - max. Kühlleistung - 20°C - ...	45.00	3192	NULL
52621	104	Exquisit / GGV GB 40 Weiss Gefrierwürfel-box - Neuware	·4 Sterne Gefrierschrank·Energieeffizien...	125.00	3197	NULL
52622	104	Severin KS 9804 Weiss - Neuware	·Energieeffizienzklasse A, 182 kWh/Jahr...	129.00	3200	3200

Abbildung 6.9: Auszug aus der Produkttabelle in MySQL

Abgefragt die Anzahl an Produktdaten angibt, die für die Abfrage passend waren. Für das automatische Abfragen werden nur Produktdaten verwendet, bei denen der Produktname eine Länge von 60 Zeichen nicht übersteigt und die Beschreibung mindestens 250 Zeichen umfasst. In den *Treffer*-Spalten wird die Anzahl der gelieferten Ergebnisse (#), ihr Anteil an allen Abfragen in Prozent (%) und der durchschnittliche Score (Scr) angezeigt. Analog wird die Information in den nächsten Spalten präsentiert, wobei unter *Richtig* die Menge der richtig erkannten Produktseiten und unter *Falsch* diejenigen Produkte stehen, für welche die Abfrage ein falsches Ergebnis geliefert hat.

Webshop	P-Seiten	Abgefragt	Treffer			Richtig			Falsch		
			#	%	Scr	#	%	Scr	#	%	Scr
T-Online Shop	3377	1460	1460	100	1.81	599	41.0	1.93	861	59.0	1.63
Notebooksbilliger	488	488	488	100	0.84	0	0.0	0.00	488	100	0.84
Power Netshop	2517	1536	1536	100	2.39	867	56.4	1.65	669	43.5	2.36
HOH.de	745	496	496	100	1.85	393	79.2	1.89	103	20.7	1.71
CSV-Direkt	4379	3513	3513	100	1.30	2590	73.7	1.43	923	26.3	0.93
Redcoon.de	317	312	312	100	0.15	1	0.3	0.02	311	99.7	0.15
PC-Planet	1469	306	306	100	1.22	260	85.0	1.14	46	12.8	1.68
NotebookNet.de	82	77	77	100	0.98	0	0.0	0.00	77	100	0.98

Tabelle 6.5: Auswertung der Ergebnisse für die Abfrage mit *Keyword-Boosting* im Lucene-Index

Im Allgemeinen sind beide Abfragetypen nicht fehlerfrei. Die *Keyword-Boosting*-Variante liefert für einige Shops sehr gute Ergebnisse, für andere ist die Fehlerquote allerdings nicht akzeptabel, wie zum Beispiel bei *Notebooksbilliger* und *Redcoon*. Da die ausgeführte Suche auf dem Vektormodel basiert, wird ein partielles Matching der Dokumente ermöglicht. Aus diesem Grund ist die Anzahl der Treffer immer 100% von der abgefragten Menge. Der Anteil der richtigen Treffer ist leider nicht stabil und im Durchschnitt deutlich unter 50%. Die *Distanzsuche*-Abfrage zeigt eine geringere Fehlerhäufigkeit und hat bei allen getesteten Webshops richtige Ergebnisse geliefert. Bei genauerer Betrachtung der Ergebnisse, wird es klar, dass die *Distanzsuche* Abfrage weniger Treffer als die *Keyword* Suche erzielt, aber die Präzision der Treffer in den meisten Fällen 50% übersteigt. In drei der acht getesteten Webshops wurde sogar eine fast hundertprozentige Genauigkeit erreicht. Aus diesem Grund wird sie gegenüber der *Keyword-Boosting* Suche in den nächsten Experimenten vorgezogen.

Webshop	P-Seiten	Abgefragt	Treffer			Richtig			Falsch		
			#	%	Scr	#	%	Scr	#	%	Scr
T-Online Shop	3377	1460	887	60.7	2.85	425	47.9	2.95	462	52.1	2.75
Notebooksbilliger	488	488	306	62.7	1.54	283	92.5	1.53	23	7.5	1.65
Power Netshop	2517	1536	1000	65.1	1.60	598	59.8	1.65	402	40.2	1.54
HOH.de	745	496	272	54.8	1.28	151	55.5	1.29	121	44.5	1.28
CSV-Direkt	4379	3513	1883	53.6	2.02	1675	88.6	2.01	208	11.4	2.09
Redcoon.de	317	312	193	61.6	1.44	135	69.9	1.50	58	30.1	1.29
PC-Planet	1469	306	147	48.0	2.11	141	95.9	2.13	6	4.1	1.71
NotebookNet.de	82	77	24	31.2	0.56	5	20.8	0.70	19	79.2	0.53

Tabelle 6.6: Auswertung der Ergebnisse für die Distanzsuche im Lucene-Index

Erkennung potentieller Produktseiten in unbekanntem Webshops

Bei diesem Experiment wird versucht, verschiedene Produkte in einem Webshop abzufragen, die aus unterschiedlichen Internetshops stammen. Dazu werden drei Webshops ausgewählt, die ein ähnliches Produktangebot haben. Zum Beispiel verkaufen die Webshops *HOH.de*, *CSV-Direct.de* und *Notebooksbilliger.de* Hardware-Artikel. In unserem Versuch werden *CSV-Direct* und *Notebooksbilliger* als Ausgangsdaten für die Abfragen benutzt, die in *HOH.de* ausgeführt werden. Der Versuch soll zeigen, wie gut das entwickelte Suchverfahren Produktseiten in einem neu indextierten Webshop findet. Laut der manuell durchgeführten Produktextraktion befinden sich in den beiden Quell-Webshops insgesamt 4834 Produkten. Davon können 4001 abgefragt werden, auf Grund der Begrenzung für die Länge der Produktnamen und Beschreibungen, die in der Abfrage benutzt werden können (Tabelle 6.6). In dem Versuchsworkshop *HOH.de* befinden sich 745 Produktseiten, von denen das Programm initial nichts kennt. Anschließend wird der gleiche Versuch mit anderen Webshops wiederholt. Als Versuchsshop wird *T-Online-Shop* ausgewählt, und als Ausgangsbasis für die Produktdaten die Webshops *Redcoon* und *CSV-Direct*. Insgesamt wurden bei diesem Versuch 3825 Abfragen nach Produktseiten durchgeführt.

Die Artikel, die der Webshop *HOH.de* gemeinsam mit den beiden Quell-Webshops hat, ist nicht bekannt, denn sie müssen eventuell manuell abgeglichen werden. Eine kurze manuelle Vergleichsanalyse in der MySQL Datenbank hat gezeigt, dass die für die Experimente ausgesuchten Webshops ziemlich wenige Produkte gemeinsam haben. Deswegen ist es schwierig den Anteil der gefundenen gegenüber der nicht gefundenen Produktseiten zu errechnen. Dieser Versuch hat das Ziel zu zeigen, ob in einem unbekanntem Webshop Produktseiten anhand Produktdaten aus unterschiedlichen Webshops überhaupt gefunden werden können. Die Ergebnisse dieses Experimentes werden in Tabelle 6.7 und Abbildung 6.10 präsentiert. Das Programm hat insgesamt 17 Produktseiten richtig erkannt, zwei Seiten aus *HOH.de* wurden fälschlicherweise als Produktseiten eingestuft. Die Ergebnisse sind nicht wirklich optimal. Im ersten Versuch konnten aus 4001 nur 7 Treffer geliefert werden und bei dem zweiten Versuch 10 Treffer aus 3825 Abfragen. Allerdings ist davon auszugehen, dass mit einer größeren Produktdatenbank, die als Quelle für die Abfragen dienen kann, die Trefferquote optimiert werden kann. Als positives Ergebnis kann die geringe Fehlerquote betrachtet werden. Aus 17 Treffern wurden nur zwei fälschlicherweise als Produktseiten erkannt.

Quell-Shops	Abfragen	Ziel-Shop	Produktseiten	Richtig	Falsch
CSV-Direct, Notebooksbilliger	4001	HOH.de	745	7	2
CSV-Direct, Redcoon	3825	T-Online-Shop	3377	10	0

Tabelle 6.7: Auswertung der Ergebnisse aus der Produktseitenerkennung in unbekanntem Webshops

6.5.3 Klassifizierung der Webseiten mit XPath

Das wichtigste Experiment aus der Phase der automatischen Produktextrahierung ist die Klassifizierung der Seiten in einem Webshop nach Produkt- bzw. Nicht-Produktseiten. In diesem Experiment wird die vorher generierte Menge von potentiellen Produktseiten analysiert und gegebenenfalls Nicht-Produktseiten aussortiert. Im Gegensatz zum vorherigen Versuch wird nicht nur das Dokument mit dem besten Score genommen, sondern die besten drei Webseiten der Ergebnismenge. Damit wird die Wahrscheinlichkeit reduziert, dass Lucene fälschlicherweise eine nicht relevante Seite als *Top-Document* einstuft. Als Untergrenze für den Score der Ergebnisse wurde 0.3 verwendet, damit die wenig relevanten Dokumente von der Analyse ausgeschlossen bleiben. Bei der Analyse (beschrieben in Unterkapitel 5.6) sucht das System in jedem relevanten Dokument nach Mustern, die auf das gesuchte Produkt hindeuten. Als Erstes wird eine modifizierte Distanzsuche nach dem bekannten Produktnamen in dem Dokument ausgeführt. Im Erfolgsfall werden alle Positionen im Quellcode lokalisiert und es wird ein XPath-Ausdruck generiert, der diese Position adressiert. Anschließend werden diese XPath-Abfragen analysiert und anhand des letzten Elements (Html-Tag) priorisiert. Einige werden gegebenenfalls entfernt, wenn zum Beispiel der gesuchte Text in einem Link-Tag vorkommt (<a>). In diesem Fall handelt es sich bei dem untersuchten Dokument höchstwahrscheinlich nicht um die gesuchte Produktseite, sondern um eine Seite, die zu der Produktseite verweist. Die generierten XPath-Ausdrücke für den Produktnamen werden als Muster für die Erkennung der Produktseiten verwendet. Für jeden davon wird eine Abfrage in der XML-Datenbank ausgeführt und die Ergebnisse werden analysiert. Dabei ist das Kriterium die Anzahl der Dokumente, die diesen XPath-Ausdruck als Muster beinhalten. Als zusätzliches Kriterium wird die durchschnittliche Länge der Ergebnisse berücksichtigt, die eine bestimmte Länge nicht überschreiten darf. So wird sichergestellt, dass der XPath-Ausdruck nicht auf die Produktbeschreibung zeigt, in welcher der Produktname selbst durchaus vorkommen könnte. Als nächster Schritt wird ein XPath-Ausdruck für die Produktbeschreibung generiert. Dabei wird nach Keywords aus der bekannten Beschreibung im Dokument gesucht und mit dem in Unterkapitel 5.6 beschriebenen Algorithmus ein Knoten ausgewählt, für den die XPath-Expression generiert wird. Ein dritter XPath-Ausdruck ist für die Erkennung der Position des Produktpreises notwendig. Dafür wird die komplette Webseite nach Währungs- bzw. Zahlenmuster durchsucht, die von dem bekannten Produktpreis nur bis zu einem gewissen Grad abweichen. In diesem Versuch wurde eine maximale Abweichung von 50% vom dem bekannten Produktpreis definiert. In einer produktiven Umgebung können alle drei XPath-Ausdrücke als Muster für die Erkennung und Klassifizierung von Produktseiten verwendet werden. Wir beschränken uns bei unserem Versuch auf die XPath-Expression für den Produktnamen, um die Ausdrücke zu vereinfachen.

Aus dem vorherigen Experiment (Abschnitt 6.5.2) ist bekannt, dass die Webshops *CSV-Direct*, *Notebooksbilliger* und *HOH.de* teilweise die gleichen Produkte anbieten. Entsprechend werden bei unserem Test die manuell extrahierten Produktdaten aus den Webshops *CSV-Direct* und *No-*

6 Experimente

Quell-Shop	Produktname	Preis	Ziel-Shop	Produktname	Preis
CSV-Direct	TFT 19" AOC 919P2	150,76 €	HOH	AOC 919P2, 19" TFT, DVI/VGA, schwarz	152,90 €
CSV-Direct	CANON Pixma iP2600	42,85 €	HOH	Canon PIXMA iP2600 (2435B009)	42,90 €
CSV-Direct	TFT 19" ASUS VH196D	106,68 €	HOH	ASUS VH196D, 19" TFT, VGA, schwarz	114,90 €
NB	Logitech Cordless Desktop 660 schwarz	24,90 €	HOH	Toshiba Satellite P300-23E (PSPCCE-05203HGR), Demo	1.199,90 €
NB	Apple Mini DisplayPort auf-DVI-Adapter	28,90 €	HOH	Apple MacBook Air 13,3" (ctoMB543ZH), Messedemo	1.549,90 €
NB	Logitech ClearChat PC Wireless	73,90 €	HOH	Logitech ClearChat PC Wireless (981-000069), Demo	64,90 €
NB	SAMSUNG SyncMaster 2333HD	229,90 €	HOH	Samsung SyncMaster 2333HD, 23" TFT, schw.	232,90 €
Redcoon	Beurer MG 190	99,27 €	T-Online	Beurer MG 190 Shiatsu-Sitzaufgabe	85,90 €
Redcoon	Garmin Edge 705 HR	320,95 €	T-Online	Garmin Edge 705 HR mit Brustgurt Fahrrad-Navi + Computer + Höhenmessung	338,00 €
Redcoon	Garmin Edge 305 Bundle	246,95 €	T-Online	Garmin Edge 305 inkl. Brustgurt Fahrradcomputer mit GPS	264,90 €
Redcoon	Polar RS300X Run orange	153,95 €	T-Online	Polar RS300X RUN Orange Multisportcomputer mit S1-Laufsensor	149,90 €
CSV-Direct	NAVIGON 8310	365,50 €	T-Online	Navigon 8310 - Europa mit TMCpro High-End Edel-Navigations-System	348,95 €
CSV-Direct	LowePro Nova 140 AW Schultertasche	38,25 €	T-Online	LowePro Nova 140 AW, schwarz Schultertasche f. SLR & Equipment	39,99 €
CSV-Direct	LowePro Nova 160 AW Schultertasche	43,26 €	T-Online	LowePro Nova 160 AW, schwarz Schultertasche f. SLR & Equipment	49,99 €
CSV-Direct	Nokia 6500 slide silver ohne Vertrag	217,19 €	T-Online	Nokia 6500 slide silver Mobiltelefon - Ohne Vertrag!	199,99 €
CSV-Direct	Nokia 6500 classic bronze ohne Vertrag	183,61 €	T-Online	Nokia 6500 classic mocca / bronze Mobiltelefon - Ohne Vertrag!	184,99 €
CSV-Direct	TomTom GO 940 LIVE	365,17 €	T-Online	TomTom GO 740 LIVE BT, HD Traffic, 4,3 Zoll	298,90 €

Abbildung 6.10: Auszug aus der Datenbank. Die ersten drei Spalten zeigen die Beispielsprodukt-daten, die letzten drei - die Daten aus der erkannten Produktseiten. Die falsch erkannten Produkten sind gelb markiert.

tebooksbilliger als Ausgangsdaten verwendet. Dabei sei über den Shop *HOH.de* bisher nichts bekannt. Das System hat die Aufgabe, die Positionen der Produktattribute zu markieren und XPath-Ausdrücke zu generieren. Eine Beispielsproduktseite zeigt Abbildung 6.11, wobei zusätzlich ein Auszug aus dem Quellcode dargestellt wird. Genau so wie bei dem Experiment vom Abschnitt 6.5.2, werden alle bekannten Produkte von *CSV-Direct* und *Notebooksbilliger* im Webshop *HOH.de* abgefragt. In der Tabelle 6.8 werden alle gelieferten Treffer in diesem Versuch aufgelistet. Wenn die Tabelle genauer untersucht wird, fällt auf, dass die zwei von Lucene fälschlicherweise erkannten Produkte bei der Analyse vom System ausgefiltert wurden. Tatsächlich handelte es sich bei diesen beiden Treffern um Kategorienseiten, die auf die entsprechenden Produktseiten verweisen. Das Programm konnte den Text auf diesen Seiten zwar lokalisieren, er stand aber innerhalb eines Link-Tags (<a>). Zusätzlich wurde eine neue Produktseite für *Office 2007 Small Business* erkannt, die bei der Suche mit Lucene nicht gefunden wurde. Der Grund liegt darin, dass das entwickelte System nicht nur den besten Treffer herausgreift, sondern auch die beiden nachfolgenden Treffer in der Treffermenge berücksichtigt.

Im Folgenden werden die Ergebnisse des Experimentes anhand eines Auszugs aus der Program-

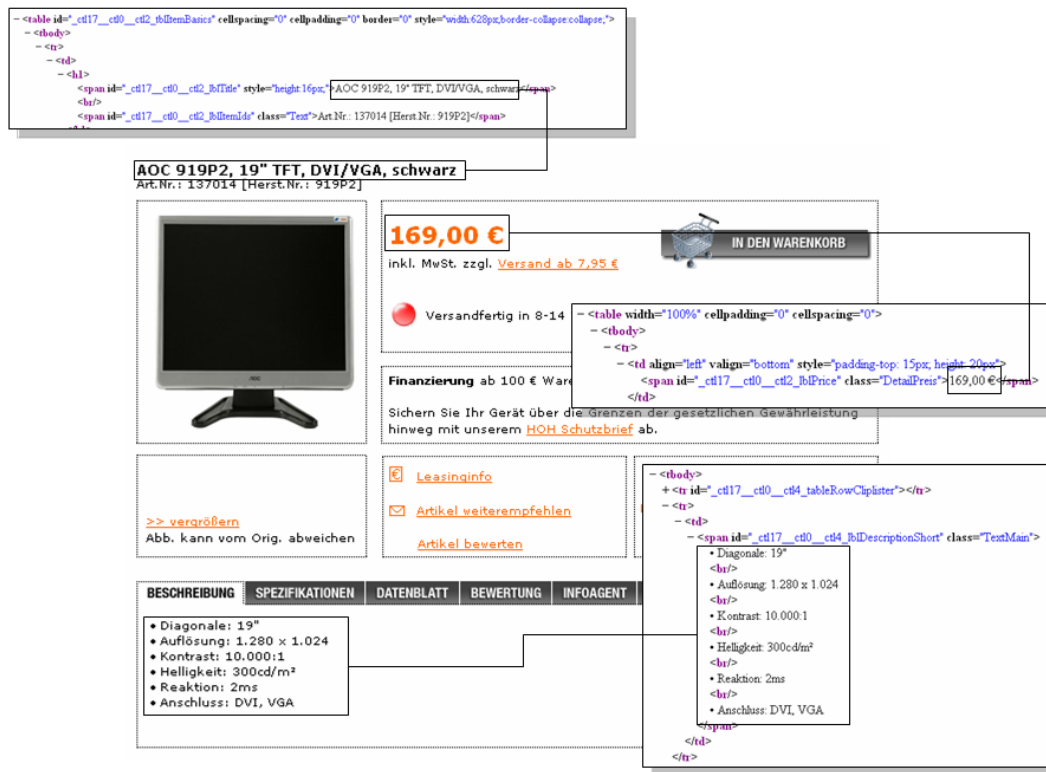


Abbildung 6.11: Beispiel für Produktseite in hoh.de, die Positionen der Produktattributen werden im Quellcode markiert.

Treffer	Produktseite	XPath generiert	Ähnliche Seiten	Alle Produktseiten
CANON Pixma iP2600	ja	ja	745	745
TFT 19" AOC 919P2	ja	ja	745	745
TFT 19" ASUS VH196D	ja	ja	745	745
Logitech ClearChat PC Wireless	ja	ja	745	745
Office 2007 Small Business	ja	ja	745	745
SAMSUNG SyncMaster 2333HD	ja	ja	745	745
Logitech Cordless Desktop 660 schwarz	nein	nein	745	745
Apple Mini DisplayPort auf DVI Adapter	nein	nein	745	745

Tabelle 6.8: Ergebnisse von der Klassifizierung der Webseiten

ausgabe präsentiert. Die generierten Ausdrücke sind bis auf die Beschreibung für alle Produkte identisch, und mit ihnen konnten 745 Produktseiten in der XML-Datenbank erkannt werden - genauso viele, wie bei der manuellen Produktextrahierung entdeckt worden sind. Fazit ist, die sechs richtig erkannten Produktseiten haben ausgereicht, um die Webseiten in dem unbekanntem Shop *HOH.de* erfolgreich zu klassifizieren.

XPath-Generierung für Webshop: HOH (www.hoh.de)

6 Experimente

Suche nach Produkt: CANON Pixma iP2600
Dokumente gefunden: 3
Produktseite gefunden: ja (score: 2.13)
XPath-Produktname:

```
html/body[1]/form[1]/div[3]/table[3]/tbody[1]/tr[1]/td[1]/table[1]/tbody[1]/tr[1]/td[1]  
/h1[1]/span[1]
```

XPath-Produktbeschreibung:
html/body[1]/form[1]/div[3]/div[1]/div[2]/span[1]/table[1]/tbody[1]/tr[4]/td[1]/p[1]

XPath-Produktpreis:
html/body[1]/form[1]/div[3]/table[3]/tbody[1]/tr[1]/td[1]/table[1]/tbody[1]/tr[1]/td[1]
/table[1]/tbody[1]/tr[1]/td[3]/table[1]/tbody[1]/tr[1]/td[1]/span[1]

XPath-Produktseite:
//page[src/html/body[1]/form[1]/div[3]/table[3]/tbody[1]/tr[1]/td[1]/table[1]/tbody[1]/
tr[1]/td[1]/h1[1]/span[1]]

Anzahl erkannte Produktseiten: 745

Suche nach Produkt: TFT 19" AOC 919P2
Dokumente gefunden: 1
Produktseite gefunden: ja (score: 0.46)
XPath-Produktname: (gleich)

XPath-Produktbeschreibung:
html/body[1]/form[1]/div[3]/div[1]/div[2]/span[1]/table[1]/tbody[1]

XPath-Produktpreis: (gleich)

XPath-Produktseite: (gleich)

Anzahl erkannte Produktseiten: 745

Suche nach Produkt: TFT 19" ASUS VH196D
Dokumente gefunden: 1

Produktseite gefunden: ja (score: 1.78)

XPath-Produktname: (gleich)

XPath-Produktbeschreibung:

```
html/body[1]/form[1]/div[3]/div[1]/div[2]/span[1]/table[1]/tbody[1]
```

XPath-Produktpreis: (gleich)

XPath-Produktseite: (gleich)

Anzahl erkannte Produktseiten: 745

Suche nach Produkt: Logitech ClearChat PC Wireless

Dokumente gefunden: 2

Produktseite gefunden: ja (score: 0.74)

XPath-Produktname: (gleich)

XPath-Produktbeschreibung:

```
html/body[1]/form[1]/div[3]/div[1]/div[2]/span[1]
```

XPath-Produktpreis: (gleich)

XPath-Produktseite: (gleich)

Anzahl erkannte Produktseiten: 745

Suche nach Produkt: Office 2007 Small Business

Dokumente gefunden: 3

Produktseite gefunden: ja (score: 0.61)

XPath-Produktname: (gleich)

XPath-Produktbeschreibung:

```
html/body[1]/form[1]/div[3]/div[1]/div[2]/span[1]
```

XPath-Produktpreis: (gleich)

XPath-Produktseite: (gleich)

Anzahl erkannte Produktseiten: 745

Suche nach Produkt: SAMSUNG SyncMaster 2333HD

Dokumente gefunden: 1

Produktseite gefunden: ja (score: 0.65)

XPath-Produktname: (gleich)

```
XPath-Produktbeschreibung:  
html/body[1]/form[1]/div[3]/div[1]/div[2]/span[1]/table[1]/tbody[1]  
XPath-Produktpreis: (gleich)  
XPath-Produktseite: (gleich)  
Anzahl erkannte Produktseiten: 745
```

```
Suche nach Produkt: Logitech Cordless Desktop 660 schwarz  
Dokumente gefunden: 3  
Produktseite gefunden: nein
```

```
Suche nach Produkt: Apple Mini DisplayPort auf DVI Adapter  
Dokumente gefunden: 1  
Produktseite gefunden: nein
```

6.5.4 Datenextrahierung und Datenbankimport

Dieser Versuch hat das Ziel, die Extrahierung der Produkteigenschaften *Produktname*, *Beschreibung* und *Preis* aus der bereits als Produktseiten klassifizierten Dokumente in dem unbekanntem Webshop *HOH.de* auszuwerten. Dafür werden die in Unterkapitel 6.5.3 generierten XPath-Ausdrücke verwendet, die die oben genannten Produkteigenschaften in der Webseite adressieren. Da die Produktdaten aus diesem Shop bereits bei der manuellen Produktextrahierung (Unterkapitel 6.4) in die MySQL geschrieben wurden, mussten sie bei diesem Versuch einem neuen Webshop zugeordnet werden. Schließlich wurden die Daten aus der manuellen Produktextrahierung und die Produktdaten, gewonnen anhand der automatisch generierten Ausdrücke, gegenübergestellt. Zum Vergleich werden hier die manuell konstruierten XPath-Ausdrücke aufgelistet:

```
XPath-Produktname:  
//form[id="Form1"]/div/table/tbody/tr/td/table/tbody/tr/td/h1/span[1]/text()  
XPath-Produktbeschreibung:  
//div[@id='hohTabPane']/div[3]  
XPath-Produktpreis:  
//span[@class='DetailPreis']  
XPath-Produktseite:  
//page[//form[id="Form1"]/div/table/tbody/tr/td/table/tbody/tr/td/h1/span[1]/text()]
```

Wenn man die Ausdrücke genauer anschaut, wird klar, dass sie grundsätzlich gleich sind. Die manuell erstellten sind relativ, die automatisch erstellten dagegen absolut. Im Endeffekt adressieren sie die gleichen Knoten in der XML-Datenbank. Für die Extrahierung aller Produktdaten wird das Modul *ProductImporter* verwendet, das die Daten schließlich in die MySQL-Datenbank anlegt. In Abbildung 6.12 ist der Quellcode abgebildet, mit dem die Klasse instanziiert und der Datenbank-Import angestoßen wird. Die Methode importiert alle Produktdaten mit einer neuen Webshop-ID, in diesem Fall *110*. Zusätzlich wird in Abbildung 6.13 einen Screenshot von der Produkttabelle gezeigt. Dort werden die Produkte, die anhand der manuell erstellten XPath-Expressions gewonnen wurden, an die mit dem automatischen Verfahren extrahierten Produkten gegenübergestellt. Es wurde eine SQL-Abfrage erstellt, die alle Produkte mit gleichen Namen in der Produkttabelle selektiert. Damit kann erkannt werden, dass die mit dem automatischen Verfahren generierten Muster genau die gleichen Produktdaten extrahieren, wie die manuell erstellten.

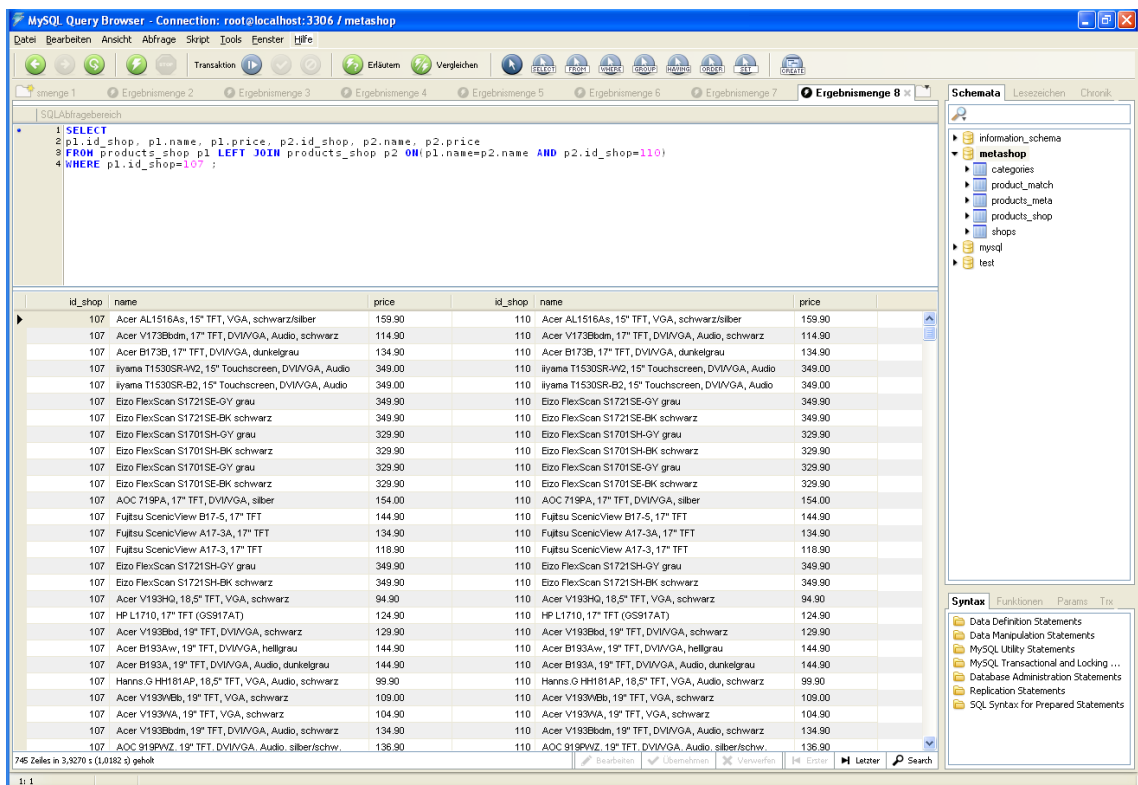
6 Experimente

```
/**
 * @param args
 */
public static void main(String[] args) {
    // Test ProductImporter

    ProductImporter pxi = new ProductImporter(110, "shop_hoh");
    String xpathPage = "//page[src/html/body[1]/form[1]/div[3]/table[3]/tbody[1]/tr[1]/td[1]/table[1]/tbody[1]/tr[1]/td[1]/h1[1]/span[1]";
    String xpathName = "//html/body[1]/form[1]/div[3]/table[3]/tbody[1]/tr[1]/td[1]/table[1]/tbody[1]/tr[1]/td[1]/h1[1]/span[1]/text()";
    String xpathDesc = "//html/body[1]/form[1]/div[3]/div[1]/div[2]/span[1]/table[1]/tbody[1]/tr[4]/td[1]/p[1]";
    String xpathPrice = "//html/body[1]/form[1]/div[3]/table[3]/tbody[1]/tr[1]/td[1]/table[1]/tbody[1]/tr[1]/td[1]/table[1]/tbody[1]/tr[1]/td[3]/table[1]/tbody[1]/tr[1]/td[1]/span[1]";

    pxi.writeProductsToDB(xpathPage, xpathName, xpathDesc, xpathPrice);
}
```

Abbildung 6.12: Quellcode für die Ausführung des Datenbankports



The screenshot shows the MySQL Query Browser interface. The SQL query is:

```
1 | SELECT
2 | p1.id_shop, p1.name, p1.price, p2.id_shop, p2.name, p2.price
3 | FROM products_shop p1 LEFT JOIN products_shop p2 ON p1.name=p2.name AND p2.id_shop=110
4 | WHERE p1.id_shop=107 ;
```

The results are displayed in a table with the following columns: id_shop, name, price, id_shop, name, price. The table contains 24 rows of product data, including items from Acer, Eizo, Fujitsu, and Hannes G.

id_shop	name	price	id_shop	name	price
107	Acer AL1516As, 15" TFT, VGA, schwarz/silber	159.90	110	Acer AL1516As, 15" TFT, VGA, schwarz/silber	159.90
107	Acer Y173Bdm, 17" TFT, DVIVGA, Audio, schwarz	114.90	110	Acer Y173Bdm, 17" TFT, DVIVGA, Audio, schwarz	114.90
107	Acer B173B, 17" TFT, DVIVGA, dunkelgrau	134.90	110	Acer B173B, 17" TFT, DVIVGA, dunkelgrau	134.90
107	iiyama T1530SR-M2, 15" Touchscreen, DVIVGA, Audio	349.00	110	iiyama T1530SR-M2, 15" Touchscreen, DVIVGA, Audio	349.00
107	iiyama T1530SR-B2, 15" Touchscreen, DVIVGA, Audio	349.00	110	iiyama T1530SR-B2, 15" Touchscreen, DVIVGA, Audio	349.00
107	Eizo FlexScan S1721SE-GY grau	349.90	110	Eizo FlexScan S1721SE-GY grau	349.90
107	Eizo FlexScan S1721SE-BK schwarz	349.90	110	Eizo FlexScan S1721SE-BK schwarz	349.90
107	Eizo FlexScan S1701SH-GY grau	329.90	110	Eizo FlexScan S1701SH-GY grau	329.90
107	Eizo FlexScan S1701SH-BK schwarz	329.90	110	Eizo FlexScan S1701SH-BK schwarz	329.90
107	Eizo FlexScan S1701SE-GY grau	329.90	110	Eizo FlexScan S1701SE-GY grau	329.90
107	Eizo FlexScan S1701SE-BK schwarz	329.90	110	Eizo FlexScan S1701SE-BK schwarz	329.90
107	AOC 719PA, 17" TFT, DVIVGA, silber	154.00	110	AOC 719PA, 17" TFT, DVIVGA, silber	154.00
107	Fujitsu ScenicView B17-5, 17" TFT	144.90	110	Fujitsu ScenicView B17-5, 17" TFT	144.90
107	Fujitsu ScenicView A17-3A, 17" TFT	134.90	110	Fujitsu ScenicView A17-3A, 17" TFT	134.90
107	Fujitsu ScenicView A17-3, 17" TFT	118.90	110	Fujitsu ScenicView A17-3, 17" TFT	118.90
107	Eizo FlexScan S1721SH-GY grau	349.90	110	Eizo FlexScan S1721SH-GY grau	349.90
107	Eizo FlexScan S1721SH-BK schwarz	349.90	110	Eizo FlexScan S1721SH-BK schwarz	349.90
107	Acer V193HQ, 18.5" TFT, VGA, schwarz	94.90	110	Acer V193HQ, 18.5" TFT, VGA, schwarz	94.90
107	HP L1710, 17" TFT (G5917AT)	124.90	110	HP L1710, 17" TFT (G5917AT)	124.90
107	Acer V193Bbd, 19" TFT, DVIVGA, schwarz	129.90	110	Acer V193Bbd, 19" TFT, DVIVGA, schwarz	129.90
107	Acer B193Aw, 19" TFT, DVIVGA, hellgrau	144.90	110	Acer B193Aw, 19" TFT, DVIVGA, hellgrau	144.90
107	Acer B193A, 19" TFT, DVIVGA, Audio, dunkelgrau	144.90	110	Acer B193A, 19" TFT, DVIVGA, Audio, dunkelgrau	144.90
107	Hannes G HH181AP, 18.5" TFT, VGA, Audio, schwarz	99.90	110	Hannes G HH181AP, 18.5" TFT, VGA, Audio, schwarz	99.90
107	Acer V193MB, 19" TFT, VGA, schwarz	109.00	110	Acer V193MB, 19" TFT, VGA, schwarz	109.00
107	Acer V193WA, 19" TFT, VGA, schwarz	104.90	110	Acer V193WA, 19" TFT, VGA, schwarz	104.90
107	Acer V193Bdm, 19" TFT, DVIVGA, Audio, schwarz	134.90	110	Acer V193Bdm, 19" TFT, DVIVGA, Audio, schwarz	134.90
107	AOC 919PWZ, 19" TFT, DVIVGA, Audio, silber/schw.	136.90	110	AOC 919PWZ, 19" TFT, DVIVGA, Audio, silber/schw.	136.90

Abbildung 6.13: Screenshot von MySQL Query Browser. Auszug aus der Produkttabelle.

7 Zusammenfassung und Ausblick

Die grundlegende Idee hinter dieser Diplomarbeit ist die automatische Auslesung des gesamten Produktangebots eines Webshops, samt aller Produktpreise und Eigenschaften zum Zeitpunkt der Analyse. Dabei sollte Initial keine Informationen über die Webseite vorhanden sein, lediglich die Internetadresse. Diese Aufgabe hat sich als eine große Herausforderung dargestellt, da die meisten Internetshops eine unikale Struktur besitzen, wird die Information immer anders präsentiert. Die Entwicklung einer Lösung für diesen Zweck war zeitintensiv und lief über mehrere Phasen. Als erstes war eine grundlegende Einarbeitung im Bereich der *Information Retrieval* und *Informations Extraktion* notwendig. Gleichzeitig wurden erste Versuche mit dem Open-Source Tool für *Web Mining* namens *WebHarvest*¹ gemacht, Produktinformationen aus einer Reihe von bekannten Webshops zu entnehmen. Anhand einer Konfigurationsdatei, die für jede Website manuell zu entwerfen ist, wurden sehr gute Ergebnisse erreicht. Um diesen Prozess völlig zu automatisieren sind spezielle Muster zu generieren, um die Positionen der Produkteigenschaften auf den verschiedenen Seiten zu erkennen. Aufgrund der Komplexität der Konfigurationssprache und der Instabilität bei größeren Webshops konnte selbst bei vorhandenen Mustern eine automatisierte Generierung der Konfigurationsdatei, und damit des ganzen Extrahierungsprozesses, nicht erreicht werden. Anschließend wurde zu diesem Zweck ein Tool entwickelt, das mit Hilfe eines eingebauten WebClients die Webseiten der Onlineshops analysiert und anhand einer Beispielsmenge von bekannten Produktdaten, die notwendigen Muster für die vollständige Auslesung der Produkte selbst konstruiert. Das Programm nutzt das Suchformular der Websites, um an die Ergebnisliste zu kommen, in der die Produkte und ihre wichtigsten Informationen in einer tabellarischen Form präsentiert sind. Von da aus werden spezielle Muster gewonnen, um auf alle Produkttabellen im Onlineshop zuzugreifen und die dort aufgelisteten Produktdaten zu extrahieren. Trotz einiger Einschränkungen hat das Programm funktioniert und wurde erfolgreich mit einigen großen Internetshops getestet. Um richtig zu funktionieren setzt das Verfahren voraus, dass gewisse Funktionalitäten bei dem untersuchten Webshop vorhanden sind. Außerdem besteht dieses Verfahren aus sehr vielen Einzelschritten und ist sehr fehleranfällig. Aus diesem Grund wurde anschließend die Strategie grundlegend überdacht und an einer alternativen Lösung gearbeitet, die flexibel genug und für die meisten und bekanntesten Webshops geeignet ist.

Die Aufgabe der Software ist, die Startseite des Internetshops aufzurufen und von da aus alle über Links erreichbaren Webseiten lokal in einer Dokumentdatenbank zu speichern. Dann werden die Seiten analysiert und diejenigen, die detaillierte Produktinformationen beinhalten, ausgewählt. Dort werden die Positionen, wo die Produktdaten auftauchen, automatisch erkannt und anschließend alle Produkteigenschaften aus diesen Seiten entnommen. Dabei dient der Fakt als Grundlage, dass die so genannten Produktseiten innerhalb eines Webshops die gleiche Struktur haben und demzufolge die gleichartigen Daten an denselben Stellen zu finden sind. Um diese Positionen in einem unbekanntem Onlineshop zu lokalisieren werden bereits vorhanden Produktdaten benutzt,

¹WebHarvest - <http://web-harvest.sourceforge.net/>

die in dem zu analysierenden Onlineshop zu finden sind. Ist eine Produktinformation bekannt, wird anhand der Position und des Seiten-Quellcodes spezielle Muster in Form von XPath-Ausdrücken generiert, die dazu dienen, alle Produktseiten zu erkennen und alle restlichen aussortieren. Danach werden die Seiten ausgewählt und die dort vorhandenen Produktinformationen, wie Produktname, Preis und Beschreibung, anhand der vorher generierten Muster einzeln entnommen. Bei einer erfolgreichen Extrahierung der Produktdaten werden die gewonnenen Daten in einer Datenbank aufgenommen und können als Grundlage für das Untersuchen weiterer unbekannter Internetshops verwendet werden. Im Endeffekt werden die Produktdaten aus den Webshops, wo sie in einer unstrukturierten Form auftauchen, in eine strukturierte Form überführt. Als Ergebnis liefert die Software eine Produkttabelle, wo die Produkte aus allen ausgelesenen Webshop einheitlich verwaltet werden. Für die wichtigsten Produkteigenschaften, die mit diesem System extrahiert werden können, existiert dort eine Spalte. Momentan ist die Anzahl der Produktattribute beschränkt auf drei: - der Produktname, die Produktbeschreibung inklusive aller Spezifikationen und das wichtigste Attribut - der Produktpreis. Das Programm kann bei Bedarf mit weiteren Attributen erweitert werden, wie Verfügbarkeit und Lieferzeit, vorausgesetzt, die Informationen ist auf den Webseiten zu finden.

Um die Richtigkeit des Verfahrens, und die Zuverlässigkeit der entwickelten Software unter Beweis zu stellen, wurden verschiedene Versuche mit Echtdateien durchgeführt. Die Versuchsdaten stammen von acht bekannten Internetshops aus Deutschland, wobei die Anzahl der insgesamt vom Crawler ausgelesenen Webseiten ca. 30.000 beträgt. Hierzu muss erwähnt werden, dass bei einigen der Webshops nicht alle erreichbaren Seiten besucht werden konnten, auf Grund der Firewalls und der Sicherheitsmechanismen des Webserver. Im Allgemeinen besteht Verbesserungsbedarf in diesem Prozess, um möglichst viele Internetseiten in den Shops zu besuchen zu können und herunterzuladen. Als nächstes Experiment wurde mit *RapidMiner*² versucht, die Webseiten in einem unbekanntem Webshop nach Produkt und nicht nach Produktseiten zu klassifizieren. Als Grundlage dienten bereits klassifizierte Webseiten aus anderen Webshops, die als Trainingsmenge für den *SVM* verwendet worden sind. Auf Grund der unterschiedlichen Daten und Quellcode-Struktur in den Seiten der verschiedenen Internetshops konnte dieses Verfahren nicht überzeugen. Die weiteren Experimente bestanden darin, das Verfahren für die Erkennung von Produktseiten, das auf *Lucene*³ basiert, und anschließend die Klassifizierung der Webseiten und die Extrahierung der Produktdaten auszuwerten. Für die Suche nach potentiellen Produktseiten mit *Lucene* wurden zwei verschiedene Abfragearten ausprobiert und nach der Auswertung der Fehlerquote wurde eine Keywordsuche kombiniert mit Distanzsuche vorgezogen. Allerdings hat das Verfahren nicht hundertprozentig fehlerfrei funktioniert und die Anzahl an Produktseiten, die nicht gefunden werden konnten lassen vermuten dass es ein gewisses Verbesserungspotenzial gibt. Mit gezielten Optimierungen bei der lexikalischen Analyse der Webseiten könnte eventuell eine Verbesserung der Fehlerquote erreicht werden. Als nächstes wurde ein Experiment durchgeführt, Produktseiten und deren Eigenschaften in einem unbekanntem Webshop anhand der bekannten Produktdaten aus anderen Shops zu erkennen. Selbst wenn bei dem Versuch nur ganz wenige Produktseiten von der *Lucene*-Suche erkannt werden konnten, reichte es aus, alle Seiten im untersuchten Webshop zu klassifizieren, die Produktdaten erfolgreich zu extrahieren und sie in eine relationale Datenbank zu speichern.

²RapidMiner - <http://rapid-i.com/>

³Lucene - Open-Source Information Retrieval System

Die Produkttabelle in der relationalen Datenbank kann als Ausgangsbasis für die Erstellung eines Preisvergleichssystems dienen. Um Produkte aus verschiedenen Internetshops zu vergleichen, muss ein Clustering-Verfahren entwickelt werden, das die unter Umständen unterschiedlich benannten, aber gleichen Produkte matchen, und zu einem Meta-Produkt zusammenzufassen kann. Zusätzlich kann eine Prozedur entwickelt werden, die mit Hilfe der Methoden des maschinellen Lernens, neu extrahierte Produkte in eine vorgegebene Produktkategorie ordnet.

Literaturverzeichnis

- [Abi97] Serge Abiteboul. Querying semi-structured data. In *ICDT*, pages 1–18, 1997.
- [AFG03] Mohammad Abolhassani, Norbert Fuhr, and Norbert Gövert. Information extraction and automatic markup for xml documents. In *Intelligent Search on XML Data*, pages 159–178, 2003.
- [BMSW98] Daniel M. Bikel, Scott Miller, Richard M. Schwartz, and Ralph M. Weischedel. Nymble: a high-performance learning name-finder. *CoRR*, cmp-lg/9803003, 1998.
- [BPSM97] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language (xml). *World Wide Web Journal*, 2(4):27–66, 1997.
- [Bra02] Tim Bray. Internet media type registration, consistency of use, 2002. <http://www.w3.org/2001/tag/2002/0129-mime>.
- [ByRN99] Ricardo Baeza-yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. 1999.
- [Cla97] James Clark. Comparison of sgml and xml, 1997. <http://www.w3.org/TR/NOTE-sgml-xml-971215.html>.
- [Cla99] James Clark. Xml path language (xpath) version 1.0, 1999. <http://www.w3.org/TR/xpath>.
- [Col04] Gord Collins. Optimizing for froogle search engine - froogle optimization, 2004. <http://www.seoconsultants.com/articles/1383/froogle-optimization.asp>.
- [Con04] Dan Connolly. <http://www.w3.org/protocols/rfc2616/rfc2616.html>, 2004. <http://www.w3.org/2001/tag/2002/0129-mime>.
- [DEW97] Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld. A scalable comparison-shopping agent for the world-wide web. In *Agents*, pages 39–48, 1997.
- [DRJ99] Arnaud Le Hors Dave Raggett and Ian Jacobs. Html 4.01 specification, 1999. <http://www.w3.org/TR/html401/>.
- [Eck99] Robert Eckstein. *XML pocket reference*. O'Reilly & Associates, Inc., pub-ORA:adr, 1999.
- [Eer06] Matthew Eernisse. Build your own ajax web applications, 2006. <http://www.sitepoint.com/article/build-your-own-ajax-web-apps/>.
- [FG99] Walter Fierz and Rolf Grütter. The sgml standardization framework and the introduction of xml, 1999. <https://tspace.library.utoronto.ca/html/1807/4573/jmir.html>.

- [Fos] Charles Foster. Xml databases - the business case. <http://www.cfooster.net/articles/xmldb-business-case>.
- [Gri97] Ralph Grishman. Information extraction: techniques and challenges. In *Information Extraction (International Summer School SCIE-97)*, pages 10–27. Springer-Verlag, 1997.
- [GW00] Paul Grosso and Norman Walsh. Xsl concepts and practical use, 2000. <http://nwalsh.com/docs/tutorials/xsl/xsl/slides.html>.
- [Hal05] David Hall. An xml-based database of molecular pathways. Master’s thesis, Linköpings universitet, 2005.
- [Hea78] HS Heaps. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, 1978.
- [Ima] Lucid Imagination. Getting started with lucene. <http://www.lucidimagination.com/Community/Hear-from-the-Experts/Articles/Getting-Started-Lucene>.
- [Kri01] David M. Kristol. Http cookies: Standards, privacy, and politics. *ACM Trans. Internet Techn.*, 1(2):151–198, 2001.
- [MRS09] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, 2009. <http://www.informationretrieval.org/>.
- [MWK⁺06] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. Yale: Rapid prototyping for complex data mining tasks. In Lyle Ungar, Mark Craven, Dimitrios Gunopulos, and Tina Eliassi-Rad, editors, *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940, New York, NY, USA, August 2006. ACM.
- [Rob98] Jonathan Robie. What is the document object model?, 1998. <http://www.w3.org/TR/REC-DOM-Level-1/introduction.html>.
- [Rät98] Manfred Rätzmann. Three-tier-development, 1998. http://www.dfpug.de/konf%5Ckonf_1998%5C09_tier%5Cd_tier/d_tier.htm.
- [SBF00] Don Chamberlin Scott Boag and Mary F. Fernández. Xquery 1.0: An xml query language, 2000. <http://nwalsh.com/docs/tutorials/xsl/xsl/slides.html>.
- [SMB94] C.M. Sperberg-McQueen and Lou Burnard. Guidelines for electronic text encoding and interchange (TEI P3). Technical report, Text Encoding Initiative, Chicago, Illinois, April 1994.
- [SMYM⁺08] C. M. Sperberg-McQueen, François Yergeau, Eve Maler, Jean Paoli, and Tim Bray. Extensible markup language (XML) 1.0 (fifth edition). W3C proposed edited recommendation, W3C, February 2008. <http://www.w3.org/TR/2008/PER-xml-20080205>.

- [Stu03] Tony Stubblebine. *Regular Expression Pocket Reference*. O'Reilly, 2003.
- [Ver06] Versandhandel.org. Entwicklung des e-commerce in deutschland, 2006.
http://www.versandhandel.org/uploads/media/2008-11-05__Charts_bvh_E-Commerce_PK_01.pdf.