

Anreicherung von Webseiten um beschreibende Metadaten

Studienarbeit

Universität Rostock, Fachbereich Informatik, Lehrstuhl Datenbank- und
Informationssysteme



Andreas Schulz

Betreuer: Dr.-Ing. Meike Klettke

Abgabedatum: 01.09.2001

Inhaltsverzeichnis

1	Einleitung	4
2	Metadaten	5
2.1	Sinn von Metadaten	5
2.1.1	Information Retrieval, Suche in grossen Datenbeständen . .	5
2.1.2	“Originärer“ Sinn der Metadaten	5
2.2	Begriffsdefinition - Was sind Metadaten?	6
2.3	Klassifikation von Metadaten	6
2.3.1	Klassifikation nach der Inhaltsabhängigkeit	6
2.3.2	Klassifikation nach der Nutzungsart	7
3	Datengewinnung durch Abfragen des WWW	8
3.1	Design und Aufgabe von Wrappern	8
3.1.1	Aufgabe von Wrappern	8
3.1.2	Design von Wrappern	9
3.2	Das W4F-Toolkit	11
3.2.1	Prinzipielle Arbeitsweise eines W4F-Wrappers	13
3.3	Probleme bei der Arbeit mit WWW-Datenquellen	22
4	Wissensbasierte Verfahren aus dem Projekt GETESS	23
4.1	Einleitung	23
4.2	Die GETESS-Architektur	23
4.3	Ontologie	25
5	Kombination von Wrappermethoden und Ontologiewissen	28
5.1	Konzept zur Kombination	28
5.2	Prototypische Implementierung anhand der Beispieldomäne ”Geschichte Mecklenburg-Vorpommerns”	29
6	Anhänge	30
6.1	Anhang A — Wrapper-Spezifikation.	30
6.2	Anhang B — Quelltext für die generierte Wrapper-Klasse.	31
6.3	Anhang C — Quelltext für die Swing-Oberfläche.	34
6.4	Anhang D — Quelltext für die Klasse, die gewrappte Daten steuert.	41
6.5	Anhang E — Quelltext für die Klasse, die den Datenbankzugriff steuert.	44
6.6	Anhang F — Ontologietabelle.	47
6.7	Anhang G — Tabelle mit den Ausprägungen zu den Konzepten.	49
	Tabellenverzeichnis	51

<i>Inhaltsverzeichnis</i>	3
---------------------------	---

Abbildungsverzeichnis	52
------------------------------	----

1 Einleitung

Das explosionsartige Anwachsen des World Wide Web (WWW) in den letzten Jahren und die damit verbundene Flut von bereitgestellten Daten machen es unumgänglich, relevante von unwichtigen Informationen zu trennen. Weltweit stellen Firmen, Behörden, Universitäten, andere Forschungseinrichtungen und Privatpersonen aus verschieden motivierten Gründen Informations- und Serviceangebote bereit. Dabei handelt es sich sowohl um einfache Textdokumente als auch um Schnittstellen zu Datenbanken. Auch als Erstveröffentlichungsquelle ist das WWW nicht zu unterschätzen. Viele Arbeiten werden zu erst oder in einigen Fällen nur im WWW veröffentlicht. Der Drang nach Steigerung der Aktualität lässt das WWW aufgrund der Schnelligkeit der elektronischen Informationsvermittlung zu der Alternative schlechthin im Bezug auf die klassischen Medien wie Zeitung, Fernsehen, Rundfunk oder auch Buch werden.

Sucht der Nutzer in der Masse von Dokumenten nach zu einem Thema relevanten Informationen steht er vor mehreren Problemen. Zunächst muss er WWW-Datenquellen finden, die relevante Daten zur Verfügung stellen. Dies geschieht im allgemeinen unter Zuhilfenahme von diversen Suchmaschinen, welche aufgrund von Stichworten die Metadaten einer gewissen Anzahl von registrierten Dokumenten durchsuchen und eine Ergebnismenge zurückliefern. Aufgrund der Möglichkeit von Doppeldeutigkeiten von Daten liegt es am Nutzer, diese Dokumente selbst zu durchforsten, um eine mögliche Relevanz zum benötigten Thema zu erkennen.

Ziel dieser Studienarbeit ist es, kontextbezogene Webseiten um beschreibende Metadaten anzureichern. Dabei wird eine Kombination von Wrappermethoden und wissensbasierten Verfahren eingesetzt. Mit Hilfe dieser generierten Metadaten soll nun ein weiteres Schrumpfen der Datenmenge ermöglicht werden, um nur noch kontextrelevante Dokumente zu behalten. Kapitel 2 beschäftigt sich mit Metadaten und versucht sich mit einer möglichen Klassifizierung dieser. Anschliessend wird das Wrapper-Toolkit W4F zur Erzeugung von HTML-Wrappern untersucht. Wissensbasierte Verfahren aus dem Projekt GETESS werden dann in Kapitel 4 vorgestellt. Abschliessend wird eine Kombination der beiden Verfahren anhand der Beispieldomäne "Geschichte Mecklenburg-Vorpommerns" vorgestellt.

2 Metadaten

Ziel dieses Kapitels ist es die Notwendigkeit von Metadaten zu verdeutlichen. Im Verlauf des Kapitels wird versucht, den Begriff Metadaten zu definieren um anschliessend mögliche Klassifikationen¹ von Metadaten zu erläutern.

2.1 Sinn von Metadaten

Anwendungsgebiet und Nutzen von Metadaten sind sehr gross. Anhand von zwei Beispielen sollen Verwendungszweck und Notwendigkeit von Metadaten verdeutlicht werden.

2.1.1 Information Retrieval, Suche in grossen Datenbeständen

Anstatt grosse Datenbestände mit aufwendiger Volltextsuche zu durchforsten, besteht die Möglichkeit Metadaten anzulegen, da diese um ein Vielfaches kleiner sind und damit schneller durchsucht werden können. Die Effizienz der Suche wird so unter Umständen enorm verbessert. Zwar gibt es sicherlich auch im Information Retrieval Methoden und Techniken die Effizienz in der Suche und Qualität im Suchresultat garantieren, was aber nicht dagegen spricht, Metadaten gezielt einzusetzen. Die Tatsache, dass die durchschnittlich benötigte Datenmittlungszeit sinkt, ist hierbei ein nicht zu vernachlässigender Aspekt. Bedeutung gewinnt diese Technik gerade im Bezug auf die ständig zunehmende Vernetzung von Rechnern. Die Suche nach Informationen von auf entfernt liegenden Rechnern wird verbessert, wenn man vor dem Herunterladen der Daten weiss, ob diese sich als nützlich erweisen. Metadaten erlauben in diesem Kontext die Brauchbarkeit der Daten gezielt zu ermitteln.

2.1.2 “Originärer“ Sinn der Metadaten

Neben dem bisher genannten Bereich, der für die Nutzung von Metadaten spricht, gibt es ein klassisches Beispiel aus dem alltäglichen Leben, das auf intuitive und verständliche Art den Zweck von Metadaten illustriert. Dieser Ansatz wurde mit dem Ausdruck originärer Sinn von Metadaten überschrieben, da es sich wahrscheinlich um den ursprünglichsten Sinn überhaupt handelt: Bei einer Bibliothek handelt es sich um eine Menge von Büchern (Dateneinheiten). Bücher lassen sich mit Hilfe von Attributen verschlagworten, so dass ein Katalog (Metadatenbank, Metadata Dictionary) entsteht, der die wichtigsten Suchkriterien (Metadaten) umfasst. Dazu gehören etwa der Titel, der Autor, Erscheinungsdatum, assoziative Schlagwörter aus dem Schlagwortverzeichnis oder andere Klassifizierungsattribute. Daneben

¹nach [TM01]

werden noch weitere Informationen zu finden sein, etwa der Verwahrungsort, Verweise auf andere Bücher oder weitere externe Verweise wie beispielsweise auf Publikationen in anderen Fachbibliotheken. Mit Hilfe des Kataloges lässt es sich auf einfache und effiziente Weise unter Angabe von wenigen Informationen suchen. Es kann also über die Relevanz eines Buches entschieden werden, ohne dass es vorher gelesen wurde.

Es gibt natürlich weitere Aspekte und Szenarien, die den Gebrauch von Metadaten befürworten, aber weniger mit dem Thema dieser Studienarbeit zu tun haben. Hierfür sei auf entsprechende Fachliteratur verwiesen.

2.2 Begriffsdefinition - Was sind Metadaten?

Wie bereits festgestellt, handelt es sich bei Metadaten um "Daten über Daten", also Daten, die verschiedene Eigenschaften von Datensätzen beschreiben und den inhaltlichen Kontext herstellen. Es bleibt festzustellen, dass es keine einheitliche Definition gibt und so jeder Autor eines Artikels eine eigene Definition proklamiert. Eine kurze, treffende Definition lässt sich im Entwurf der ISO Spezifikation 11179 finden. Metadaten werden dort beschrieben als

"The information and documentation which makes data sets understandable and shareable for users"

Die genannte ISO Definition 11179 ist die wichtigste Definition, die auf Metadaten Anwendung finden kann. Dort werden in sechs Abschnitten die Aufgaben, Prinzipien und Richtlinien für die Klassifizierung, Attributierung, Namenskonvention (Identifikation) und Strukturierung von Daten diskutiert.

2.3 Klassifikation von Metadaten

In diesem Abschnitt werden einige Möglichkeiten der Klassifikation von Metadaten diskutiert. Hierbei handelt es sich nur um eine Auswahl von Klassifikationsmöglichkeiten und erhebt keinen Anspruch auf Vollständigkeit. So ist es möglich Metadaten nach ihrer Inhaltsabhängigkeit oder Nutzungsart zu unterteilen.

2.3.1 Klassifikation nach der Inhaltsabhängigkeit

- inhaltsabhängige Metadaten
Inhaltsabhängige Metadaten beziehen sich auf den Inhalt des Dokuments bzw. der Information. Beispiele für solche Metadaten sind Schlüsselwörter oder Daten über den syntaktischen Aufbau der Informationen.

- **inhaltsunabhängige Metadaten**
Inhaltsunabhängige Metadaten beziehen sich auf Informationen, die nichts mit dem Inhalt des Dokuments, welches sie beschreiben, zu tun haben. Hierbei handelt es sich sowohl um identifizierende Metadaten (ID-Nummer, Versionsnummer,...) als auch um administrative Metadaten (Aufbewahrungsort, Status,...).

2.3.2 Klassifikation nach der Nutzungsart

- **aktive Nutzung der Metadaten**
Ein System (Metadata Repository, System Catalog), das aktive Nutzung der Metadaten betreibt, ist immer konsistent mit der zugrundeliegenden Datenstruktur. Alle Änderungen in Struktur und Daten werden automatisch im System geändert.
- **passive Nutzung der Metadaten**
Dem gegenüber können bei passiver Nutzung von Metadaten Inkonsistenzen zwischen Metainformationen und Datenstruktur auftreten. Akteure der Datenbank (Administratoren, Endanwender) sind selbst verantwortlich für die Aktualität und Konsistenz ihrer Dokumentation von Struktur und Prozessen der Datenbank.

3 Datengewinnung durch Abfragen des WWW

Inhalt dieses Kapitels ist das automatische Gewinnen von Daten durch Abfragen von WWW-Datenquellen. Der Zugriff wird durch eine Software, dem sogenannten Wrapper, realisiert. Da das WWW als Datenquelle für den menschlichen Nutzer konzipiert wurde, sind Probleme bei der Datenextraktion mittels einer Mensch-Maschinen-Schnittstelle nicht vermeidbar. Derzeit sind Wrapper aber die einzige Möglichkeit, um Daten aus WWW-Datenquellen abzufragen.

Im Abschnitt 3.1 wird der Begriff Wrapper geklärt und auf Konzepte für das Wrapper-Design eingegangen. Der Hauptabschnitt 3.2 ist dann dem W4F-Toolkit als Vertreter von Toolkits zur Wrapperimplementierung gewidmet. Abschliessend werden Probleme, die beim Einsatz von Wrappern bei der Abfrage von WWW-Datenquellen auftreten und somit eine Einschränkung des Einsatzbereiches darstellen, aufgeführt.

3.1 Design und Aufgabe von Wrappern

3.1.1 Aufgabe von Wrappern

Was versteht man unter einem Wrapper?

Als Wrapper wird eine Software bezeichnet, die Daten und Anfragen zwischen zwei Datenmodellen konvertiert. Im Zusammenhang mit dem WWW bedeutet es folgendes:

Ein Wrapper für eine WWW-Datenquelle extrahiert die im HTML-Dokument implizit gespeicherten Daten und konvertiert diese in explizit gespeicherte Daten eines Datenmodells. Das HTML-Dokument ist das Ergebnis einer Anfrage an die WWW-Datenquelle mittels des HTTP-Protokolls².

Den Ablauf kann man sich folgendermassen vorstellen:

Ein Client verbindet sich über eine URL (Unified Request Locator), unter Angabe der Übertragungsmethode (GET oder POST) für die zu übermittelnden Parameter, mit einem Server. Der HTTP-Server liefert bei einer erfolgreichen Anfrage ein HTML-Dokument zurück. Desweiteren wird immer eine Menge von HTTP-Parametern zurückgeliefert, die dem Client zu behandelnde Fehlerzustände, Authorisierungszustände, Umleitungen, ... mitteilen.

Für die implizit im Dokument gespeicherten Daten müssen nun eine Zielstruktur und der Datentyp ermittelt werden. Bei der Bestimmung des Datenmodells sind HTML-Struktur und Text-Struktur hilfreich. Desweiteren werden Verfahren

²Eine genaue Beschreibung von HTTP und HTML ist in [MK97] zu finden

zur Extraktion der Daten aus dem HTML-Dokument benötigt. Dieses Reverse-Engeneering ist Aufgabe des Wrapper-Entwicklers.

3.1.2 Design von Wrappern

Inhalt dieses Abschnitts sind Entwurfskriterien und zwei Design-Konzepte für Wrapper.

Entwurfskriterien

- **Änderungsfreundlichkeit**
Da sich das Layout von Dokumenten im WWW recht häufig ändert, ist eine leichte Anpassbarkeit an diese Änderungen erforderlich.
- **Robustheit**
Wrapper sollen nicht bei jeder kleineren Layout-Änderung scheitern, sondern trotzdem den Grossteil der Daten extrahieren.
- **Wiederverwendbarkeit**
Eine fertige Extraktionslösung soll auch bei anderen Wrappern einsetzbar sein.
- **Portierbarkeit**
Plattformunabhängigkeit ist eine wünschenswerte Eigenschaft von Wrappern. Der Einsatz eines unter Windows geschriebenen Wrappers soll beispielsweise auch in einer Unix-Umgebung möglich sein.
- **Implementierungsunabhängigkeit**
Ein nachträglicher Wechsel der Programmiersprache ist ebenfalls ein Kriterium für das Wrapper-Design.

Klassisches Design³

Sind Extraktionslösung und Zieldatenstruktur Bestandteil des Programmcodes in einer Programmiersprache, so wird der Wrapper als klassisch bezeichnet. Es gibt keine Trennung zwischen Programmcode für Programmablauf, Datenextraktion und Abbildung auf die Zieldatenstruktur. Eine negative Folge dieser Eigenschaft zeigt sich bei einer eventuell nötig werdenden Anpassung des Wrappers, da der gesamte Programmcode analysiert werden muss. Ausserdem sind Probleme bei der Wiederverwendbarkeit der Extraktionslösung nicht ausgeschlossen, da der Programmcode für die Extraktion nur schwer vom Programm getrent werden kann. Ein

³siehe [RG00]

nachträglicher Wechsel der Programmiersprache ist nicht möglich, der Wrapper müsste neu geschrieben werden. Geringe Probleme im Bezug auf die Portierbarkeit könnten nur auftreten, wenn die Verfügbarkeit der gewählten Programmiersprachen auf anderen Plattformen nicht gewährleistet werden kann.

Design-Richtlinien des W4F-Toolkits⁴

Um den oben genannten Entwurfskriterien für einen Wrapper zu entsprechen, wurden von den W4F-Entwicklern Design-Richtlinien aufgestellt und im W4F-Tolkit umgesetzt. Diese werden im folgenden beschrieben.

- Die Verwendung einer modularen dreistufigen Schichtenarchitektur gewährleistet Unabhängigkeit und Wiederverwendbarkeit. Die Schichten stellen die Teilaufgaben des Wrappers dar und werden mit Retrieval-Schicht, Extraktionsschicht und Abbildungsschicht bezeichnet.
 - Das Abfragen der WWW-Datenquelle mit dem HTML-Dokument als Ergebnis ist die Aufgabe der Retrieval-Schicht.
 - In der Extraktionsschicht werden die Daten von Interesse aus dem HTML-Dokument extrahiert. Die benutzte Sprache soll die HTML-Struktur und die Struktur der Text-Elemente ausnutzen können.
 - Die Abbildung der extrahierten Daten auf die weiterverwendbare Ziel-datenstruktur ist Aufgabe der Abbildungsschicht.
- Die einzelnen Schichten werden deklarativ spezifiziert. Da jede durch eine Menge von Spezifikationsregeln beschrieben wird, gibt es eine eindeutige Semantik, die wiederum Verständlichkeit, leichte Wartbarkeit und Wiederverwendbarkeit sichert. Die zur Implementation benutzte Programmiersprache kann leicht ausgetauscht werden, ohne die Spezifikation ändern zu müssen. Ein weiterer Vorteil ist die leichte Erlernbarkeit der Spezifikationssprache, da sie weniger mächtig ist, als es Programmiersprachen sind.
- Die Extraktion der Daten muss an deren Granularität anpassbar sein. Das heisst, dass die Datenstruktur sowohl anhand der HTML-Struktur ermittelbar, als auch die Textstruktur innerhalb der HTML-Tags auswertbar sein muss. Durch Kommas getrennte Aufzählungen können beispielsweise durch reguläre

⁴Entnommen aus [SA00] und [RG00]

Ausdrücke ausgewertet werden.

- Um eine eindeutige Identifikation der Elemente und eine einfache Navigation im HTML-Dokument zu ermöglichen, wird das HTML-Dokument abstrakt repräsentiert. Dies wird durch eine deklarative Spezifikation erreicht.
- Werkzeuge zur Unterstützung des Users bei der Erstellung und Wartung von Wrappern helfen unnötigen Aufwand zu vermeiden und die Einarbeitungszeit zu verkürzen.

Im nächsten Abschnitt wird die Wrapper-Implementierung unter Berücksichtigung der Design-Richtlinien mittels des W4F-Toolkits beschrieben.

3.2 Das W4F-Toolkit

W4F bedeutet World Wide Web Wrapper Factory und wurde von der Universität Pennsylvania (USA) und der Telecom (E.N.S.T.) Paris (Frankreich) entwickelt. Das Toolkit ist bis Version 1.21 frei verfügbar.

W4F basiert auf Java und besteht aus mehreren Komponenten. Dabei handelt es sich um:

- **W4F-Parser**
Er dient zur Analyse von HTML-Dokumenten und zeichnet sich durch eine hohe Fehlertoleranz aus. Dies bedeutet, dass auch fehlerhafte HTML-Seiten analysiert werden können. Der Parser unterstützt HTML 3.2.
- **W4F-Compiler**
Zur Generierung von Java-Klassen zur Wrapperspezifikation wird der W4F-Compiler verwendet. Seine generierten Klassen sind in jedem Java-Programm anwendbar.
- **W4F-Laufzeitmodul**
Das mitgelieferte Laufzeitmodul erlaubt die Ausführung der generierten Java-Wrapper als selbständige Programme. Es ist beispielsweise beim Testen von geschriebenen Wrappern eine nicht zu unterschätzende Hilfe.
- **Werkzeuge zur Unterstützung des Nutzers**
Die mitgelieferten Werkzeuge funktionieren allerdings nur ab dem Internet Explorer 4 und sind besonders für Einsteiger interessant.

- **Formular-Wizard**

Dient zur Hilfe bei der Erstellung von Retrieval-Regeln für WWW-Seiten, die Formulare enthalten. Er gibt die wesentlichen Elemente eines HTML-Formulares aus.

- **Extraktions-Wizard**

Dieser Wizard hilft bei der Erstellung von Extraktionsregeln. Er gibt den identifizierenden Pfadausdruck eines vom Nutzer in einer HTML-Seite selektierten Textelementes an.

- **Abbildungs-Wizard**

Dieses Hilfetool ist zur Unterstützung des Nutzers bei der Abbildung vom intern verwendeten Datenformat NSL auf nutzerdefinierte Datenstrukturen geplant.

Nach diesem kurzen Überblick erfolgt eine Beschreibung der Wrapper-Erstellung mit W4F.

3.2.1 Prinzipielle Arbeitsweise eines W4F-Wrappers

Der vorgestellten Schichtenarchitektur entsprechend, besteht ein Wrapper aus drei Teilen. In der Retrieval-Schicht erfolgt das Laden des durch die Retrieval-Regeln spezifizierten HTML-Dokumentes. Es dient als Eingabe für den HTML-Parser, der daraus eine abstrakte Darstellung, den HTML-Baum oder Analysebaum, erstellt. Jedes Element des HTML-Dokumentes kann anhand des Analysebaumes über einen Pfadausdruck eindeutig identifiziert werden. In der Extraktionsschicht werden die durch Extraktionsregeln spezifizierten Elemente extrahiert und in der NSL-Datenstruktur (Nested String List) gespeichert. In der darauffolgenden Abbildungsschicht erfolgt das Mapping dieser Daten auf die gewünschte Zieldatenstruktur. Das Ergebnis ist dann ein Java-Objekt bzw. ein Java-Standarddatentyp. In Abbildung 1 ist die prinzipielle Arbeitsweise eines Wrappers dargestellt.

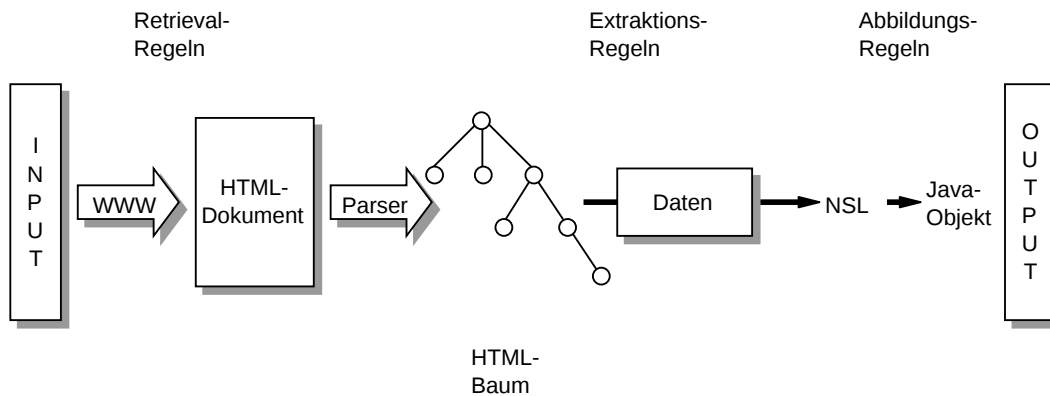


Abbildung 1: Prinzipielle Arbeitsweise eines Wrappers

Im folgenden wird auf die einzelnen Schichten detaillierter eingegangen.

- **Die Retrieval-Schicht**

In dieser Schicht wird durch Angabe von Retrieval-Regeln⁵ festgelegt, welches HTML-Dokument mit welchen Parametern von welcher Web-Adresse geladen wird. W4F unterstützt vier Retrieval-Methoden:

- File GET
- HTTP HEAD
- HTTP GET
- HTTP POST

Der Retrieval-Block im Wrapper-Quelltext beginnt mit dem Schlüsselwort `RETRIEVAL_RULES`. Es ist zwar möglich mehrere Retrieval-Rules zu definieren, aber zu jedem Zeitpunkt ist nur eine Regel aktiv, wodurch gewährleistet wird, dass immer nur ein HTML-Dokument an die Extraktionsschicht übergeben wird. Abschliessend folgen je ein Beispiel für die Übertragungsmethoden GET und POST.

Laden eines HTML-Dokumentes mit Hilfe der GET-Methode

Die URL ist in dieser Regel variabel und eventuelle Parameter werden in ihr übergeben.

```
// GET-Methode
get_document(String url)
{
METHOD: GET;
URL: $url$;
}
```

Dem gegenüber sieht ein Beispiel für die Verwendung der POST-Methode folgendermassen aus.

⁵Die ausführliche Grammatik ist in [W4F] zu finden.

Laden eines HTML-Dokumentes mit Hilfe der POST-Methode

Parameter werden in diesem Beispiel in Form von Name-Wert-Paaren hinter dem Token PARAM: angegeben. Die Werte können dabei variabel gehalten werden.

```
//POST-Methode
get_document(String keyword)
{
METHOD: POST;
URL: "http://www.amazon.com/exec/obidos/xxxx";
PARAM: "keyword-query- $keyword$, mode- "books";
}
```

Nach Abarbeitung des Retrieval-Blocks wird das geladene HTML-Dokument an die Extraktionsschicht übergeben.

- **Die Extraktionsschicht**

Sobald ein HTML-Dokument geladen wurde, beginnt der HTML-Parser mit der Erstellung des zugehörigen Analysebaums. Dabei handelt es sich um eine eindeutige Zuordnung von HTML-Dokument und Analysebaum, d.h. dasselbe HTML-Dokument ist wieder aus dem Analysebaum ermittelbar und umgekehrt.

Er besteht aus drei Bausteinen. Dabei handelt es sich um:

- **Wurzel**

Die Wurzel repräsentiert den Startknoten im Analysebaum und wird mit HTML bezeichnet.

- **Interne Knoten**

Geschlossene HTML-Tags werden als internen Knoten abgebildet und mit deren Namen bezeichnet. Sie kapseln Unterbäume, haben also Kinder und können Attribute besitzen. Bei dieser Art von Tags handelt es sich um die HTML-Sprachelemente, die durch Start- und Endmarke gebildet werden, wie z.B. `< table > ... < \table >`

- **Blätter**

Blätter des Analysebaums werden aus den offenen HTML-Tags gebildet und erhalten deren Namen als Bezeichnung. Ausserdem werden Textstücke auf die Blätter im Analysebaum abgebildet und mit PCDATA bezeichnet. Sie haben keine Kinder, können aber Attribute besitzen.

Zur Veranschaulichung wird der zum folgenden HTML-Dokument gehörende Analysebaum in Abbildung 2 dargestellt.

```

< HTML >
  < HEAD > < TITLE > Text < \TITLE >
  < BODY >
    < H1 > Text < \H1 >
    < IMG >
    < TABLE >
      < TR >
        < TH > text < BR > text < \TH >
        < TH > ... < \TH >
        < TH > ... < \TH >
      < \TR >
      < TR >
        < TD > < IMG > < A > < B > text < \B > < \A >
        < BR > < IMG > < \TD >
        < TD > ... < \TD >
        < TD > ... < \TD >
      < \TR >
      < TR >
        ...
      < \TR >
    < \TABLE >
  < \BODY >
< \HTML >

```

Die Identifikation der Elemente des Baumes sowie der Zugriff auf diese erfolgt über Pfadausdrücke, die aus Knotennamen, Operatoren, Indizes bzgl. der Knotennamen und Attributnamen gebildet werden. Die Indizes werden links absteigend vergeben, d.h. es wird zuerst links absteigend im Baum nach einem vorgegebenen Knotennamen gesucht und das Auftreten gezählt. Stimmen Zähler und Index überein, wurde der gesuchte Knoten gefunden. Indexwert Null adressiert das erste Auftreten.

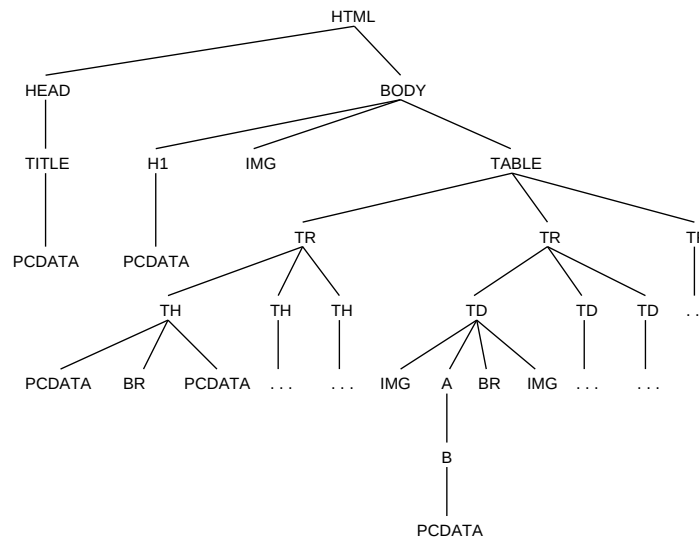


Abbildung 2: Beispiel eines HTML-Baums

Zur Bildung der Pfadausdrücke stehen zwei - entstanden aus verschiedenen Navigationsansätzen - Operatoren zur Verfügung.

- Punkt-Operator . :
Dieser Operator zeichnet sich durch die Ausnutzung der Baumstruktur aus. Hinter jedem Element im Pfadausdruck wird durch den Punkt-Operator getrennt der unmittelbare Kindknoten im Analysebaum angegeben. Damit ist es möglich zu jedem Knoten genau einen Pfadausdruck anzugeben.

Beispiele:

Pfadausdruck für den TITLE-Knoten:

HTML.HEAD.TITLE

Pfadausdruck zum ersten Bild-Knoten in der Tabelle:

HTML.BODY.TABLE[0].TR[1].TD[0].IMG

- Pfeil-Operator - > :
Charakteristisch für diesen Operator ist seine Orientierung am Fluss im HTML-Dokument. Ausgehend von einer Position im Analysebaum wird das nächste Auftreten des angegebenen Knotens unter Berücksichtigung des Indexes im Dokument gesucht. Dabei arbeitet er über die Hierarchie des Baumes hinweg. Für einen Knoten lassen sich mehrere Pfadaus-

drücke angeben.

Beispiel:

Mögliche Pfadausdrücke für das zweite Bild im BODY sind:

HTML.BODY – > *IMG*[1]

HTML.BODY – > *IMG*[0] – > *IMG*[0]

– Pfeil-Operator -/ > :

Auch dieser Operator orientiert sich am Fluss im HTML-Dokument. Im Gegensatz zum vorherigen Pfeil-Operator wird aber nur der im Pfadausdruck angegebene Teilbaum durchsucht.

Beispiel:

Pfadausdruck für das erste Bild in der zweiten Zeile der Tabelle ist:

HTML.BODY.TABLE.TR[1] – / > *IMG*[0]

Es wird nur die erste Zeile der Tabelle nach Bildern durchsucht. Ist hier kein Bild enthalten, so wird abgebrochen und im Gegensatz zum ersten Pfeiloperator, das HTML-Dokument nicht weiter durchsucht.

Eine Kombination der genannten Operatoren ist jederzeit möglich. Als Index für ein Pfadelement können auch Wildcards oder Intervalle verwendet werden, so dass mehrere Knoten das Ergebnis bilden.

Beispiele:

Alle Anker innerhalb eines HTML-Dokuments:

HTML.BODY – > *a*[*]

Alle Listenelemente der 3. bis 5. ungeordneten Liste des HTML-Dokumentes:

HTML.BODY – > *ul*[2 – 4] – > *li*[*]

Interne Knoten und Blätter können HTML-Attribute besitzen, während PCDATA-Blättern Textwerte als Attribute zugeordnet werden.

	Wurzel	Interne Knoten	Blätter	PCDATA
Kinder	x	x	-	-
HTML-Attribute	-	x	x	-
Textwert	-	-	-	x

Tabelle 1: Eigenschaften von Knoten eines HTML-Baums

Ihre Eigenschaften können mit Hilfe spezieller Zugriffsfunktionen abgefragt werden. Dies geschieht durch das Anhängen des Punktoperators und der gewünschten Funktion an das letzte Element des Pfadausdrucks. Dabei stehen folgende Zugriffsfunktionen zur Verfügung:

– Funktion **txt** :

Diese Funktion liefert den Textwert eines PCDATA-Blattes oder eine rekursive Verknüpfung der Textwerte im gesamten Unterbaum eines Knotens mit Kindern. Die Reihenfolge der Verknüpfung ist linksabsteigend zuerst.

Beispiel:

Titel des HTML-Dokumentes
HTML.TITLE.PCDATA.txt

– Funktion **src** :

Ergebnis dieser Funktion ist der HTML-Code zu einem Knoten. Die Bildung erfolgt analog zur txt-Funktion.

Beispiel:

Quellcode des HTML-Dokumentes
HTML.src

– Funktion **getAttr**(Attributname) :

Der Attributwert des angegebenen HTML-Attributes eines Knotens wird mit dieser Funktion zurückgegeben. Der Wurzel wird als Attribut der empfangene HTTP-Kopf zum HTML-Dokument zugeordnet.

Beispiel:

URL des HTML-Dokumentes
HTML.getAttr(url)

- Funktion **numberOf**(Knotenname) :
Alle Vorkommen eines Knotens im Unterbaum werden gezählt. Als Sonderfall zählt numberOf(all) alle Knoten eines Unterbaums.

Beispiel:

Anzahl aller HTML-Tags im HTML-Dokument

HTML.numberOf(all)

Die Anwendbarkeit der Funktionen ist in der folgenden Tabelle zusammengefasst.

	Wurzel	Interne Knoten	Blätter	PCDATA
txt	x	x	-	x
src	x	x	x	x
getAttr()	x	x	x	-
numberOf	x	x	-	-

Tabelle 2: Zugriffsfunktionen der Knotentypen

Die vorgestellten Pfadausdrücke benutzt man bei der Bildung von Extraktionsregeln, die in der Sprache HEL⁶ (HTML Extraction Language) spezifiziert werden. Als Ergebnis werden die Attributwerte bestimmter Knoten im intern verwendeten NSL-Format geliefert. Eine NSL (Nested String List) ist eine geschachtelte Liste von String-Listen beliebiger Tiefe. Der Datentyp NSL ist folgendermassen definiert:

NestedStringList ::= String | listOf(Nested String List) | null .

Die Struktur einer NSL besteht aus Dimension, Tiefe und Zahl der Nestungen und wird durch die angegebenen Pfadausdrücke und deren Verknüpfungen mit HEL-Operatoren bestimmt.

Beispiel:

NSL der Form *listOf(listOf(string))*

array = HTML.BODY -> ul[2 - 4] -> li[].txt*

Weitere HEL-Sprachelemente sind die Operatoren *match*, *split* und *#*. Für weitere Informationen sei dem interessierten Leser auf [W4F] verwiesen.

Der Extraktions-Block im Wrapper-Quelltext beginnt mit dem Token *EXTRACTION_RULES*. Es folgen die Extraktionsregeln, wobei die Variable

⁶die genaue Grammatik ist in [W4F] zu finden

auf der linken Seite der Extraktionsregel im Abbildungsblock deklariert sein muss, falls sie auf einen anderen Datentyp als das intern verwendete NSL-Format abgebildet werden soll.

- **Die Abbildungsschicht**

Aufgabe der Abbildungsschicht ist die Definition des Zieldatentyps, auf den die durch Extraktionsregeln gewonnenen NSL-Konstrukte abgebildet werden sollen. Für jede Extraktionsregel wird genau eine Abbildung festgelegt. Dies geschieht durch eine Typdeklaration der Extraktionsvariable im Abbildungsblock. In [W4F] ist die genaue Grammatik zu finden.

W4F bietet die Möglichkeit, die NSL auf Java-Standarddatentypen (String, int, float) und deren Array-Erweiterungen abzubilden.

Beispiel:

Abbildung der Extraktionsvariable pointers auf ein zweidimensionales String-Array

```
String[][] pointers;
```

Der deklarierten Variable pointers werden Verweisnamen und die URL aller Anker im Dokument zugewiesen.

```
pointers = HTML.BODY - > a[*](.txt # .getAttr(href));
```

Sollen komplexe NSL-Strukturen abgebildet werden, so muss der Nutzer eigene Java-Objekte (auch die Array-Erweiterungen) angeben, deren Konstruktor NSL verarbeiten kann. Die Konvertierung von NSL zu Java ist dann in der Konstruktorroutine zu implementieren. Weil keine Abbildungsregeln mehr spezifiziert werden, ist die Art der Abbildung nicht mehr deklarativ, sondern nur noch ein Interface.

Beispiel:

Die Extraktionsvariable pointers wird auf Liste von Pointer-Objekten abgebildet.

```
Pointer[] pointers;
```

```
pointers = HTML.BODY - > a[*](.txt # .getAttr(href));
```

```
//Java - Klasse Pointer
```

```
public class Pointer
```

```
{
```

```
    String name, url;
```

```
    public Pointer(NestedStringList in)
```

```
    {
```

```
        nsl = (NSL_List) in;
```

```
        name = ((NSL_String) nsl.elementAt(0)).toString;
```

```
        url = ((NSL_String) nsl.elementAt(1)).toString;  
    }  
}
```

Die Klassen `NSL.List` und `NSL.String` werden von W4F zu Verfügung gestellt, über die der Zugriff auf die `NSL`-Elemente abläuft.

Mit dem Token `SCHEMA` beginnt im Wrapper-Quelltext der Abbildungsblock. Es folgen die Typdeklaration der Extraktionsvariablen und die Extraktionsregeln.

3.3 Probleme bei der Arbeit mit WWW-Datenquellen

Beim automatischen Abfragen von Daten aus dem WWW stösst man schnell auf Grenzen. So sind beispielsweise mit dem Einsatz von Javascript, ActiveX, Java-Applets und anderen Plugins zur Darstellung von Informationen auf WWW-Seiten diese Daten für Wrapper-Systeme verloren, da keine klare Seitenbeschreibung mehr existiert. Einzig bei Javascript, dass zur Dynamisierung des sonst statischen HTML-Codes benutzt wird, ist eine Auswertung des mit übermittelten Quellcodes denkbar und somit die Ermittlung der dadurch dargestellten Informationen. Aufgrund der Komplexität dieser Scriptsprache ist aber eine Auswahl bestimmter Konstrukte erforderlich.

Selbst bei reinem HTML tauchen Probleme bei der automatisierten Datengewinnung auf. Dazu folgen einige Beispiele:

- Häufiger Layoutwechsel und teilweise fehlerhafter HTML-Code erfordern robuste und leicht anpassbare Wrapper
- Bei der Ersetzung von Text durch nicht mehr auswertbare Bilder stossen Wrapper ebenfalls auf ihre Grenzen.
- Der Einsatz von Frames erhöht den Aufwand bei der Datengewinnung durch Wrapper, da das nach aussen als Einheit wirkende HTML-Dokument aus mehreren Teilseiten besteht und diese vom Wrapper abgefragt werden müssen.
- Dynamisch durch ein Programm erzeugte HTML-Dokumente können Wrapper ebenfalls vor Probleme stellen. Beim automatischen Ausfüllen eines HTML-Formulares kann beispielsweise eine Ergebnisseite erzeugt werden, auf die der Wrapper nicht vorbereitet ist, da dieses Ergebnis noch nie auftrat.
- Erzeugt ein Programm ein Formular mit dynamisch generierten Variablennamen, so stellt das einen Programmierer, der einen Wrapper für ein solches Formular schreiben soll, vor erhebliche Probleme.

4 Wissensbasierte Verfahren aus dem Projekt GETESS

Dieses Kapitel soll einen kurzen Überblick über das Projekt GETESS und die darin angewandten Verfahren zur Erzeugung von Informationskondensaten auf Basis von Ontologiewissen geben.

4.1 Einleitung

Ziel des Projektes GETESS (G^Erman T^Ex^T E^Xploitation and S^Earch S^Ystem) ist die Realisierung eines Suchdienstes, der sich von existierenden Suchdiensten in der Hinsicht unterscheidet, dass die zu untersuchenden Texte detailliert inhaltlich analysiert werden. Dabei werden Informationskondensate (Abstracts) auf Basis von linguistischem und Ontologien-Wissen unter Verwendung einer natürlichsprachlichen Dialogschnittstelle, die es dem Nutzer erlaubt, seine Suchanfrage in natürlichsprachlichen Phrasen zu formulieren und im Dialog zu konkretisieren, gebildet. Als Ergebnis wird eine Menge von HTML-Dokumenten geliefert, deren Inhalt dem Benutzer natürlichsprachlich auf Basis der Abstracts als Zusammenfassung präsentiert wird. Die Informationen werden in relationalen Datenbanken gespeichert. Suchaufträge werden als Kombination von Information-Retrieval-Suchkommandos und Datenbankabfragen (IRQL) formuliert.

Es handelt sich um ein vom BMBF gefördertes 3-Jahres-Projekt, dass von den Partnern DFKI Saarbrücken (automatische Sprachverarbeitung), Universität Karlsruhe (Konstruktion von Ontologien), GECKO mbH Rostock (Dialoggestaltung) und der Universität Rostock (Suchmaschinen mit Datenbanktechnik) getragen wird.

4.2 Die GETESS-Architektur

Die GETESS-Architektur gliedert sich in vier Hauptkomponenten: das Dialogsystem, die Datengenerierung, das Knowledge Aquisition Tool und die Suchmaschine.

- **Datengenerierung**

Die Erstellung von Stichwort-Index und Index-Einträgen für Informationen aus Datenbanken, den sogenannten Abstracts, ist die Aufgabe der Datengenerierung. Als Eingabe dienen sowohl Textdokumente (HTML, PS, RTF, ...) als auch verschiedene Arten von Datenbanken.

- **Abstracts**

Das grundlegende Element des GETESS-Systems bilden die sogenannten

Abstracts. Sie stellen neben dem herkömmlichen Stichwort-Index die Suchbasis des Systems dar. Ein Analysesystem (Abstract-Generator) zwischen Suchmaschine und Informationen sorgt für die Kondensation der Informationen in sprachunabhängige, gewichtete Zusammenfassungen, den Abstracts. Die Ontologie nimmt als Basis der Abstract-Generierung eine wichtige Stellung ein. Die so generierten Abstracts dienen als Basis und Ergebnis der Suchanfragen. Wenn man eine Suchmaschine fürs Internet erstellt, geht man davon aus, dass sehr viele Informationen vorhanden sind und somit eine grosse Menge von Abstracts generiert werden müssen. Das bedeutet, dass Datenbanken für eine effiziente Speicherung und einen schnellen Zugriff verantwortlich sind.

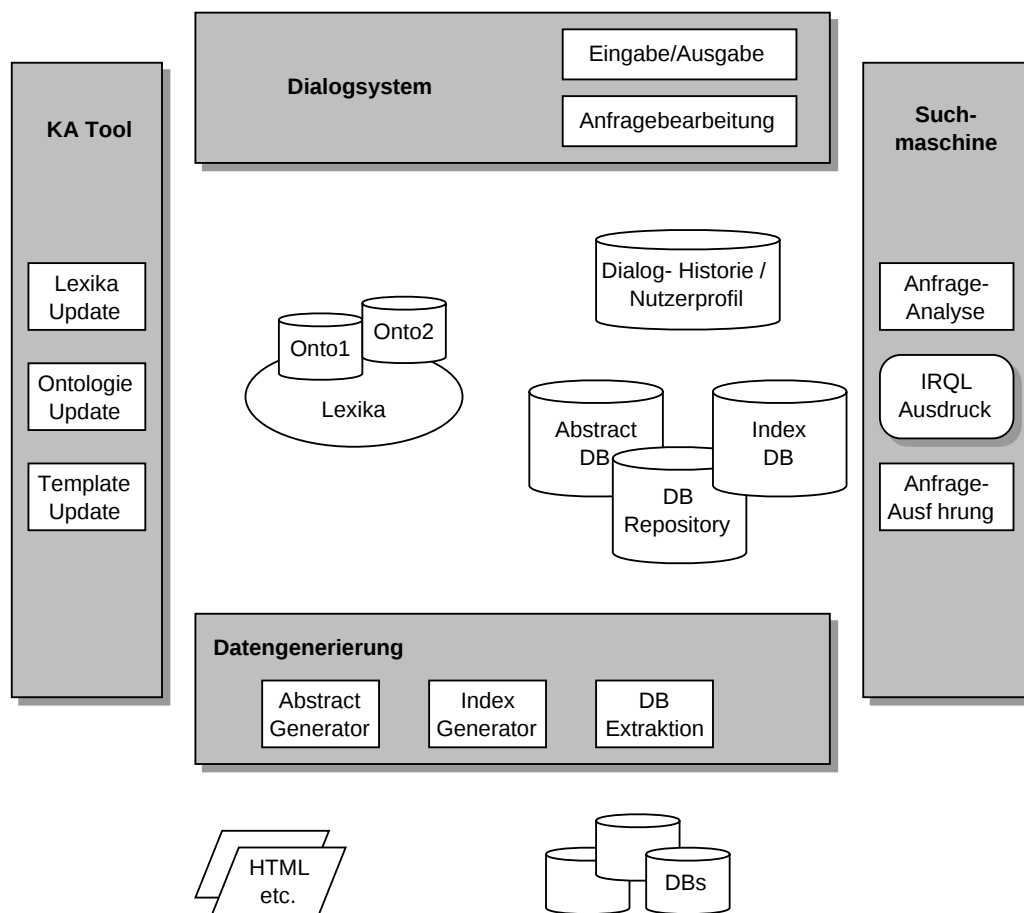


Abbildung 3: GETESS-Architektur

- **Dialogsystem**

Sprachspezifische und domainspezifische Informationen aus einer Ontologie

werden verwendet, um eine Suchanfrage zu erstellen und beantworten. Es wird angestrebt, den Benutzer in seiner Suchbeschreibung zu führen, d.h. eine Suchanforderung zu erstellen, die seine Wünsche hinsichtlich Umfang und Qualität der Information spezifiziert. Durch das Anbieten von Ober- und Unterbegriffen wird beispielsweise eine Eingrenzung des Suchraums erreicht. Eine bessere Spezifizierung des Suchvorgangs wird durch eine Aushandlung treffender Ober- und Unterbegriffe im Dialog erreicht. Dies ist hilfreich, wenn dem Benutzer keine treffenden Suchbegriffe einfallen. Der Einsatz von Dialoghistorie und Benutzer-Profile ermöglicht einen zielgerichteten Dialog zur Klärung bzw. zur Konkretisierung der Suchanfragen.

- **Knowledge Aquisition Tool**

Zur Aktualisierung des GETESS-Wissens wurde das Knowledge Aquisition Tool entwickelt. Diese kann durch häufige Nutzer-Anfragen zu einem spezifischen Thema oder neue Dokumente initiiert werden und zu einer Anpassung der Ontologie, der verwendeten Lexika für die Analyse der natürlichen Sprache führen.

- **Ontologie**

Die zentrale Stellung im GETESS-System nimmt die Ontologie ein. Sie beschreibt die Domäne Tourismus. Die Vorgabe des Kontext-Wissens zur Abstract-Bildung und die Auflösung von linguistischen Mehrdeutigkeiten sind Aufgaben der Ontologie. Sie wird auf Basis einer definierten Menge von Internet-Informationen entwickelt und ständig weiterentwickelt. Die Beschreibung, wie die Abstracts in die Datenbank gespeichert werden, ist ebenfalls Aufgabe der Ontologie.

4.3 Ontologie

Wie schon mehrfach erwähnt, nimmt die Ontologie innerhalb des Projektes GETESS eine herausragende Stellung ein. In diesem Kapitel soll ihre Aufgabe verdeutlicht werden.

Als Ontologie wird in GETESS eine Menge von Begriffen verstanden, die über inhaltliche Beziehungen zueinander geordnet sind. So ist beispielsweise Unterkunft ein Oberbegriff für Hotel, Jugendherberge oder auch Pension. Bei der Ontologieentwicklung wurden zwei wesentliche Ziele verfolgt. Zum einen stellt das Ontologiesystem Dienste zur Verfügung, die von anderen Modulen des GETESS-Systems genutzt werden. Zum anderen wirkt die Modellierung aber auch strukturgebend auf die Datenbankschemata der anderen Module und dient damit als semantisches Bindeglied zwischen Front-End und Back-End. Konkret lassen sich diese beiden Verwendungszwecke anhand der Interaktionen mit den anderen Modulen des GETESS-

Systems beschreiben:

- Interaktion mit dem Modul für die Analyse natürlichsprachlicher Eingaben:

Natürlichsprachliche Anfragen und das Parsen von Texten liefern syntaktische Beziehungen zwischen Worten und Phrasen. Diese liefern Hinweise darüber, welche Objekte inhaltlich relationiert werden sollen. Zur Klärung, ob und wenn ja, wie aufgrund der inhaltlichen Zusammenhänge relationiert werden kann, ist Aufgabe der Ontologie.

Beispiel:

Die Phrase **Rostocker Rathaus** wurde in einem Text erkannt und eine syntaktische Beziehung **Rostocker** und **Rathaus** festgestellt. Durch Abfrage der Ontologie wird jetzt festgestellt, ob aus Sicht der Ontologie eine semantische Relationierung zwischen **Rostock** und **Rathaus** möglich ist.

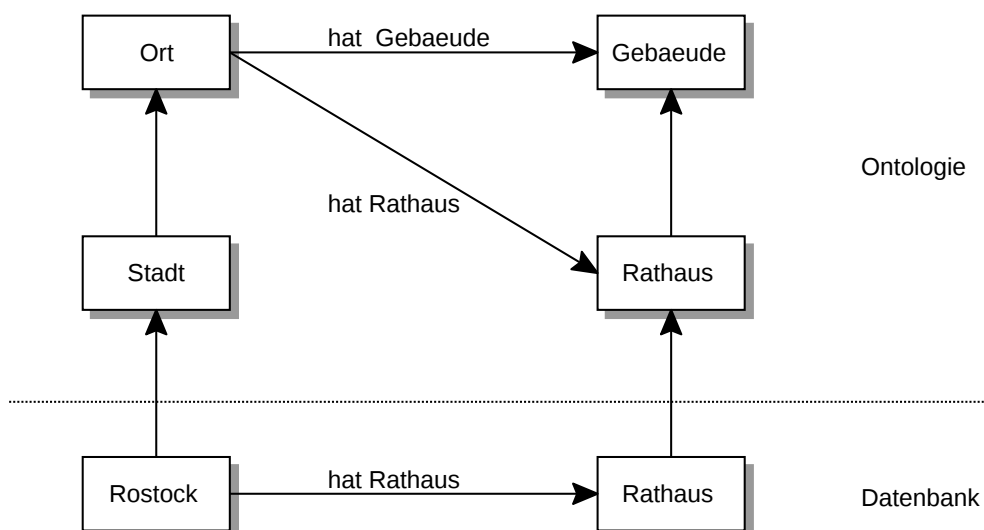


Abbildung 4: Semantische Relationierung textueller Informationen

Es wird ein Eintrag für den Begriff **Rostock** im Lexikon gefunden, der einen Verweis auf einen Eintrag in der Datenbank enthält. Es ist bekannt, dass dieses Objekt eine Instanz der Klasse **Stadt** und diese wiederum in der Ontologie definiert ist. Für **Rathaus** ist wiederum ein ähnlicher, direkter Verweis auf die in der Ontologie beschriebene allgemeine Klasse von Rathäusern definiert. Über die Ontologie erfolgt nun die Suche nach sinnvollen semantischen Relationierungen zwischen **Rostock** und **Rathaus**. Für den in Abbildung 4

angegeben Ausschnitt aus der Ontologie werden zwei Relationierungen gefunden. Das ist zum einen die Relation **hat Gebaeude** und zum anderen die Relation **hat Rathaus**. Die Beschreibungen dieser Relationen werden von der Klasse Ort an die Klasse Stadt vererbt. Da es sich bei der Relation **hat Rathaus** um die speziellere handelt, wird sie als sinnvollere Alternative angesehen und als Ergebnis zurückgegeben. **Rostock** und **Rathaus** werden als Ergebnis der Ontologieabfrage über **hat Rathaus** relationiert und diese Information mit in die Abstract-Datenbank eingebracht.

- Interaktion mit dem Datenbanksystem:

Wie schon erwähnt, bestimmt die Ontologie, wie die aus den Texten generierten Abstracts und damit die im Datenbanksystem zu speichernden semantischen Relationierungen aussehen. Dies muss sich natürlich in den Klassenbeschreibungen des Ontologiemodells im Datenbankschema niederschlagen. Andererseits ist es so, dass das Datenbanksystem das Ontologiesystem zur Hilfe nehmen kann, um Inferenzen durchführen zu lassen.

- Interaktion mit dem Dialogsystem:

Neben natürlichsprachlichen Anfragen kann das Wissen der Ontologie natürlich auch direkt über grafische Schnittstellen dem Benutzer zur Verfügung gestellt werden, um interessante Informationen zu liefern. Die Schnittstelle kann benutzt werden, um über Mausklicks komplexe Anfragen zusammenzustellen. Je nach Versiertheit mit dem System kann der Benutzer die für ihn effizienteste Anfragemethode wählen. Dies wird zunächst eine einfache natürlichsprachliche Anfrage sein, allerdings bieten bei zunehmendem Wissen die in der Ontologie modellierten Konzepte einen möglicherweise einfacheren, da zielgerichteteren, Zugang zum System.

Weitere Informationen sind für den interessierten Leser unter anderem in den Publikationen [MS1], [MS2], [MS3] und [GE99] zum Projekt zu finden.

5 Kombination von Wrappermethoden und Ontologiewissen

Ziel dieser Studienarbeit ist es, Methoden zu entwickeln, die HTML-Dokumente um beschreibende Metadaten anreichern. Dabei soll eine Kombination von Wrappermethoden und wissensbasierten Verfahren aus dem Projekt GETESS eingesetzt werden.

5.1 Konzept zur Kombination

Das Anreichern von HTML-Dokumenten um beschreibende Metadaten erfolgt prinzipiell in drei Schritten: dem Laden lokaler oder im WWW liegender HTML-Dokumente, dem Wrappen von Daten und anschließender Datenextraktion auf Basis von Ontologiewissen. Die Ergebnisse werden im Dokument als Metadaten gespeichert.

Der Benutzer hat die Möglichkeit die um Metadaten anzureichernden HTML-Dokumente lokal von seinem Rechner oder aus dem WWW zu laden. Es dient dann als Eingabe für den Wrapper, welcher Metainformationen, die sich aus der Struktur des HTML-Dokumentes ableiten lassen, extrahiert. Ausserdem wird sämtlicher Textinhalt extrahiert, bearbeitet und mit dem Ontologiewissen verglichen. In Text und Ontologie vorkommende Daten werden zu Metadaten. Sowohl die gewrappten als auch die über das Ontologiewissen gewonnenen Metadaten werden ins HTML-Dokument eingefügt und lokal gespeichert. Die prinzipielle Vorgehensweise ist in der folgenden Abbildung dargestellt.

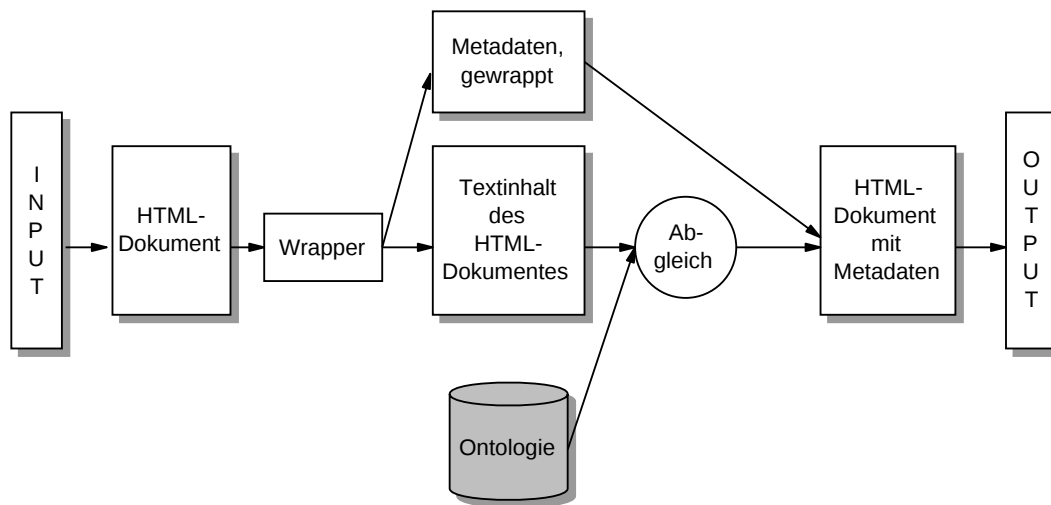


Abbildung 5: Prinzip der Kombination von Wrappermethoden und wissensbasierten Verfahren

Nach der oben beschriebenen Konzeption wird im folgenden Abschnitt eine prototypische Implementierung vorgestellt.

5.2 Prototypische Implementierung anhand der Beispieldomäne "Geschichte Mecklenburg-Vorpommerns"

6 Anhänge

6.1 Anhang A — Wrapper-Spezifikation.

```
SCHEMA
{
  String titel;
}

EXTRACTION_RULES
{
  titel = html.head.title.txt;
  inhalt = html->pcdata[*].txt;
}

RETRIEVAL_RULES{
  get(String url)
  {
    METHOD: GET ;
    URL: "$url$" ;
  }
}
```

6.2 Anhang B — Quelltext für die generierte Wrapper-Klasse.

```

/*****

class schu

This class was generated automatically by W4F

*****/

/*****/
/** user defined JAVAHEAD */
/*****/

import java.io.* ;
import java.net.* ;
import java.util.* ;

import edu.upenn.w4f.binop.* ;
import edu.upenn.w4f.conditions.* ;
import edu.upenn.w4f.nsl.* ;
import edu.upenn.w4f.util.* ;
import edu.upenn.w4f.htmlpath.* ;
import edu.upenn.w4f.parser.* ;
import edu.upenn.w4f.parser.html.* ;
import edu.upenn.w4f.mapping.xml.* ;

public class schu {

/*****/
/** type declarations */
/*****/

    public NestedStringList inhalt ;
    public String titel ;

    HtmlPath inhalt__extract_1 ;
    HtmlPath titel__extract_1 ;

//
// No mapping defined
//
/*****/
/** Retrieval method 0 */
/*****/
    static schu get(String url) throws IOException,
        UnknownHostException, edu.upenn.w4f.parser.ParseException {
        String __url ;
        RetrievalAgent __ra = new RetrievalAgent() ;

        __url = url ;

        return new schu(__ra, __url ) ;
    }
}

```

```

/*****/
/**      Constructor 1      */
/*****/
public schu(DataInputStream __in, Hashtable __header) throws IOException,
UnknownHostException, edu.upenn.w4f.parser.ParseException {
    HtmlParser __hp = new HtmlParser() ;
    HtmlNode __root = __hp.parse(__in) ;
    __root.addAttributes(__header) ;

    HtmlPathParser __hpp = new HtmlPathParser() ;
    int __i ;
    Condition __conditions[] ;

    // loading extraction rules
    __conditions = null ;
    titel__extract_1 = __hpp.parseCorrectPath(".head[0].title[0].text", __conditions) ;

    __conditions = null ;
    inhalt__extract_1 = __hpp.parseCorrectPath("->pcdata[0-].text", __conditions) ;

    // building object
    try {
        inhalt = (inhalt__extract_1.getNSLFrom(__root)) ;
    } catch (edu.upenn.w4f.htmlpath.NoSuchElementException e1) {
        inhalt = null ;
    }

    try {
        titel = NslToJava.to_String(titel__extract_1.getNSLFrom(__root)) ;
    } catch (edu.upenn.w4f.htmlpath.NoSuchElementException e1) {
        titel = NslToJava.default_String() ;
    }

}

/*****/
/**      Constructor 2      */
/*****/
public schu(RetrievalAgent ra, String url) throws IOException,
UnknownHostException, edu.upenn.w4f.parser.ParseException {
    this(ra.GET(url), ra.header) ;
    ra.close() ;
}

/*****/
/**      Constructor 3      */
/*****/
public schu(RetrievalAgent ra, String url, String postdata, String extraHeader)
throws IOException, UnknownHostException, edu.upenn.w4f.parser.ParseException {
    this(ra.POST(url, postdata, extraHeader), ra.header) ;
    ra.close() ;
}

/*****/
/**      toString      */
/*****/
public String toString() {
    String result = new String() ;

```

```
        result += "inhalt : " + inhalt + "\n" ;
        result += "titel : " + titel + "\n" ;
        return result ;
    }

    /*****
    /**      user defined JAVACODE below      */
    *****/

}
```

6.3 Anhang C — Quelltext für die Swing-Oberfläche.

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.text.*;
import java.util.*;
import java.lang.*;
import java.io.*;
import javax.swing.filechooser.*;
import javax.swing.text.html.*;
import java.net.*;

//interface des metadatengenerators
public class swing_main extends JFrame
{
    //variablendeklarationen
    static final String RED = "Red";
    static final String BLACK = "Black";
    static private final String newline = "\n";
    static private Writer out;
    private String indentString = "    ";
    private int indentLevel = 0;

    java.lang.String ext;
    JTextArea log;
    JPanel contentPane;
    JFileChooser fc;
    JMenuBar menu;
    JMenu datei, parse;
    JMenuItem punkt1, punkt2, punkt3;
    JScrollPane scrollPane, scrollPanel;
    JButton knopf, knopf2;
    JTextPane mypane, mypanel;
    StringBuffer Text;
    String endel, ausgabe, inputValue, rüber;
    File file;
    DefaultStyledDocument doc, doc1;
    String[] liste;

    //swing interface
    public swing_main()
    {
        super("Tool zur Anreicherung von Webseiten um beschreibende Metadaten");

        //Erstellen eines Frames
        setSize(800,600);

        // styles der schriftarten festlegen
        final Hashtable paraStyles;

        paraStyles = new Hashtable();
        SimpleAttributeSet attr = new SimpleAttributeSet();

        attr = new SimpleAttributeSet();
        StyleConstants.setForeground(attr, Color.red);
        paraStyles.put(RED, attr);

        final AttributeSet bigStyle = (AttributeSet) paraStyles.get(RED);
        final AttributeSet normalStyle = (AttributeSet) paraStyles.get(BLACK);
```

```
//der ganze Swing - Kram
log = new JTextArea(5,20);
    log.setMargin(new Insets(5,5,5,5));
    log.setEditable(false);
    JScrollPane logScrollPane = new JScrollPane(log);

contentPane = new JPanel();
    fc = new JFileChooser();

menu = new JMenuBar();
datei = new JMenu("Datei");
punkt1 = new JMenuItem("lokale Webseite laden");
punkt2 = new JMenuItem("Webseite laden");
punkt3 = new JMenuItem("Exit");
knopf = new JButton("File Parsen");
knopf2 = new JButton("save File");
knopf2.setEnabled(false);

//das linke JTextPane
doc = new DefaultStyledDocument();
mypane = new JTextPane (doc);
scrollPane = new JScrollPane(mypane);
JViewport vp = scrollPane.getViewPort();
vp.add(mypane);
scrollPane.setPreferredSize(new Dimension(380,500));

//das rechte JTextPane
doc1 = new DefaultStyledDocument();
mypanel = new JTextPane (doc1);
scrollPanel = new JScrollPane(mypanel);
JViewport vp1 = scrollPanel.getViewPort();
vp1.add(mypanel);
scrollPanel.setPreferredSize(new Dimension(380,500));

//wenn aktuelle File geparsed werden soll
knopf.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent c)
    {
        try
        {
            //instanz von metadaten_main und speicherung des ergebnisses in liste
            metadaten_main schu = new metadaten_main();

            //pfadnamen umformen, falls lokal
            //rüber --> dokumentadresse
            rüber = rüber.replace('\\','/');
            System.out.println("übergebene Adresse: "+rüber);
            liste = schu.toParseFile(rüber);

            //bei erfolg, save - button aktivieren
            knopf2.setEnabled(true);
        }
        catch (Exception a)
        {
            System.out.println("Fehler mit Adresse ist aufgetreten: " + a);
        }

        try
        {
            doc1.remove(0,doc1.getLength());
        }
        catch (BadLocationException v1)
```

```

{
System.out.println("Fehler beim Leeren des Textpanes ist aufgetreten: " + v1);
}

try
{
int leng1 = liste.length;
int c1 = 0;
//festlegen an welche stelle die metadaten eingetragen werden
int InsertAnStelle = ext.indexOf("<head>");
if (InsertAnStelle == -1)
{
InsertAnStelle = ext.indexOf("<HEAD>");
}
System.out.println(InsertAnStelle);
String begin = ext.substring(0,InsertAnStelle+6);
String restText = ext.substring(InsertAnStelle+7);
doc1.insertString(doc1.getLength(),begin+"\n", normalStyle);
doc1.insertString(doc1.getLength(),"<meta name =\"Metadaten_gewrappt\"
Titel=\""+liste[leng1-1]+ "\">\n", bigStyle);
for (c1=0; c1<(leng1-1); c1++)
{
doc1.insertString(doc1.getLength(),liste[c1]+\n", bigStyle);
}

doc1.insertString(doc1.getLength(),restText, normalStyle);
mypanel.setCaretPosition(0);
}
catch (BadLocationException s)
{
System.out.println("Fehler beim Füllen des Textpanes ist aufgetreten: " + s);
}
}

});

//File lokal speichern
knopf2.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent c)
{
int returnVal = fc.showSaveDialog(swing_main.this);

if (returnVal == JFileChooser.APPROVE_OPTION)
{ File file = fc.getSelectedFile();

//this is where a real application would save the file.
log.append("Saving: " + file.getAbsolutePath() + "." + newline);

try
{

System.out.println("Speichere in: " + file.getAbsolutePath());
FileWriter ausgabestrom = new FileWriter(file.getAbsolutePath());
ausgabestrom.write(mypanel.getText());
ausgabestrom.close();
}
catch (Exception l)
{
System.out.println("Fehler beim Speichern ist aufgetreten: " + l);
}

}

else
{

```

```
                log.append("Save command cancelled by user." + newline);
            }
        }

    });

    //WebFile wird geladen
    punkt2.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            knopf2.setEnabled(false);
            //Dialog zur Fileauswahl
            JOptionPane bla = new JOptionPane();
            inputValue = JOptionPane.showInputDialog("Bitte eine vollständige URL eingeben");

            try
            {
                doc.remove(0,doc.getLength());
                doc1.remove(0,doc1.getLength());
            }

            catch (BadLocationException v)
            {
                System.out.println("folgender Fehler ist aufgetreten URL: " + v);
            }

            System.out.println(inputValue);
            /*if (inputValue.equals("") == true)
            {
                //custom title, error icon
                JOptionPane.showMessageDialog(bla,
                    "Du musst schon was eingeben um weiter zu kommen!",
                    "Fehler",
                    JOptionPane.ERROR_MESSAGE);

            System.out.println(inputValue);

            }*/
            //zu überprüfende url wird gesetzt
            rüber = inputValue;
            try
            {
                //dateitextinhalt wird gelesen
                Text = new StringBuffer();
                /*URL verb = new URL(inputValue);
                BufferedReader eingabestrom = new BufferedReader(
                new InputStreamReader(verb.openStream()));

                String inputLine;

                while ((inputLine = eingabestrom.readLine()) != null)
                Text.append(inputLine);
                eingabestrom.close();

            */
            URL verb = new URL(inputValue);
            URLConnection conn = verb.openConnection();
            BufferedReader eingabestrom = new BufferedReader(
                new InputStreamReader(
                    conn.getInputStream()));

            String inputLine;
```

```
        while ((inputLine = eingabestrom.readLine()) != null)
        {System.out.println(inputLine);System.out.println(inputLine);
        Text.append(inputLine);
        }
        eingabestrom.close();

//bei erfolgreichem einlesen wird parseknopf aktiviert
knopf.setEnabled(true);

// Insert into pane
try
{
    ext= Text.toString();
    doc.insertString(doc.getLength(),ext, normalStyle);
    mypane.setCaretPosition(0);
}
catch (BadLocationException s)
{
    System.out.println("Fehler beim Datei einlesen ist aufgetreten: " + s);
}
catch (IOException n)
{
    knopf.setEnabled(false);
    knopf2.setEnabled(false);
    System.out.println("folgender Fehler ist aufgetreten:buffer " + n);
    JOptionPane.showMessageDialog(bla,
        "Fehlerhafte Eingabe!",
        "Fehler",
        JOptionPane.ERROR_MESSAGE);
    //knopf.setEnabled(false);
    //knopf2.setEnabled(false);
}

mypane.setEditable(false);
contentPane.add(scrollPane);
contentPane.add(scrollPanel);
contentPane.add(knopf);
contentPane.add(knopf2);
        setContentPane(contentPane);
}
});

//lokales File wird geladen
punkt1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        //Dialog zur Fileauswahl
        knopf2.setEnabled(false);
        knopf.setEnabled(true);
        int returnVal = fc.showOpenDialog(swing_main.this);

        if (returnVal == JFileChooser.APPROVE_OPTION)
        {
            file = fc.getSelectedFile();
            System.out.println(file.toString());
        }

        try
        {
            doc.remove(0,doc.getLength());
        }
```

```
doc1.remove(0,doc1.getLength());
}

catch (BadLocationException v)
{
System.out.println("folgender Fehler ist aufgetreten: " + v);
}
try
{
System.out.println("die ausgesuchte datei ist : " + file);

try
{
rüber = file.getPath().toString();
FileReader eingabestrom = new FileReader(file);
int gelesen;
Text = new StringBuffer(10);
boolean ende = false;

// lese Zeichen, bis Dateiende erreicht worden ist
while (!ende)
{
gelesen = eingabestrom.read();

if (gelesen == -1)
ende = true;
else
Text.append( (char) gelesen);
}

// Datei wieder schliessen
eingabestrom.close();
ext = Text.toString();

// Insert into pane
try
{
doc.insertString(doc.getLength(),ext, normalStyle);
mypane.setCaretPosition(0);
}
catch (BadLocationException s)
{
System.out.println("folgender Fehler ist aufgetreten: " + s);
}
}
catch (IOException n)
{
System.out.println("folgender Fehler ist aufgetreten: " + n);
}
}
catch(Exception f)
{
System.out.println("Da ist wohl was mit der Fileanzeige schief gelaufen!" + f);
}

mypane.setEditable(false);
contentPane.add(scrollPane);
contentPane.add(scrollPanel);
contentPane.add(knopf);
contentPane.add(knopf2);
        setContentPane(contentPane);

//this is where a real application would open the file.
```

```
        log.append("Opening: " + file.getName() + "." + newline);
    }
else
{
    log.append("Open command cancelled by user." + newline);
}
});

punkt3.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        System.exit(0);
    }
});

datei.add(punkt1);
datei.add(punkt2);
datei.addSeparator();
datei.add(punkt3);
menu.add(datei);
setJMenuBar(menu);
setContentPane(contentPane);

//Fenstersteuerung
addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});

}

//Hauptroutine
public static void main(String [] argv)
{
    swing_main frame = new swing_main();
    frame.setResizable(false);
    frame.setVisible(true);
}
}
```

6.4 Anhang D — Quelltext für die Klasse, die gewrappte Daten steuert.

```

import java.io.* ;
import java.net.* ;
import java.util.* ;
import edu.upenn.w4f.binop.* ;
import edu.upenn.w4f.conditions.* ;
import edu.upenn.w4f.nsl.* ;
import edu.upenn.w4f.util.* ;
import edu.upenn.w4f.htmlpath.* ;
import edu.upenn.w4f.parser.* ;
import edu.upenn.w4f.parser.html.* ;
import edu.upenn.w4f.mapping.xml.* ;
import java.sql.* ;

//klasse, die die ergebnisse der wrappergenerierten schu.class erhält
//und verarbeitet
public class metadaten_main
{

//methode um url der html-seite an die generierte klasse schu.class zu geben
//und ergebnis zu verarbeiten, um es als String[] zurückzugeben
//wird von swing_main aufgerufen
public String[] toParseFile(String url)throws Exception
{
//instanz von schu
schu f = schu.get(url) ;
//zu testzwecken, gewrappte überschrift der html-seite
System.out.println("Die Überschrift lautet: " + f.titel);
//gewrappter textinhalt der seite als NSL_List geholt
NSL_List nsl = (NSL_List) f.inhalt;

int size = nsl.size();
int i =0;
//Set, um darin Textinhalt, wort für wort ohne dopplungen zu speichern
Set testset = Collections.synchronizedSet(new HashSet());

/*jedes element der NSL_List wird in String verwandelt,
bearbeitet und im Set testset gespeichert
*/
for (i=0; i<size; i++)
{
String name = ((NSL_String) nsl.elementAt(i)).toString();
name = name.replace('(' , ' ');
name = name.replace(')' , ' ');
name = name.replace(',', ' ');
name = name.replace('|' , ' ');
name = name.replace('"' , ' ');
name = name.replace('? ' , ' ');
name = name.replace('!' , ' ');
name = name.replace(': ' , ' ');
name = name.replace('; ' , ' ');
name = name.trim();

if (name.indexOf("@") == -1)
{
name = name.replace('. ' , ' ');
}
}

//füllwörter entfernen

```

```
System.out.println(name + " FeldNr.:" + i );
Vector füllwörter=new Vector();
String a = "mit";
String b = "der";
String c = "die";
String d = "das";
String e = "wo";
String f1 = "ein";
String g = "über";
String h = "als";
String i1 = "an";
String j = "am";
String k = "eine";
String l = "zu";
String m = "so";
String n = "und";
String o = "kein";
String p = "jeden";
String q = "zur";

füllwörter.add(a);
füllwörter.add(b);
füllwörter.add(c);
füllwörter.add(d);
füllwörter.add(e);
füllwörter.add(f1);
füllwörter.add(g);
füllwörter.add(h);
füllwörter.add(i1);
füllwörter.add(j);
füllwörter.add(k);
füllwörter.add(l);
füllwörter.add(m);
füllwörter.add(n);
füllwörter.add(o);
füllwörter.add(p);
füllwörter.add(q);

//stringtokenizer, um jeden token der NSL_List zu extrahieren und
//ins testset zu speichern
StringTokenizer st = new StringTokenizer(name);
while (st.hasMoreTokens())
{
    String eintrag = st.nextToken();

    if (füllwörter.contains(eintrag)==false)
    {
        testset.add(eintrag);
    }
}

int d = testset.size();
System.out.println(d);
//umwandlung testset im array
String[] x = (String[]) testset.toArray(new String[0]);

//für testzwecke um arrayinhalt zu überprüfen
int leng = x.length;
System.out.println(leng);
int c = 0;

for (c=0; c<(leng); c++)
```

```
{
System.out.println("array-felder "+ x[c]+ " , " + c);
}

//instanz von DbKram, um ontologie abzufragen
final DbKram test = new DbKram();
//übergabe des arrys an DbKram und ergebnis in Set liste abspeichern
Set liste = test.eintraege(x);
System.out.println(liste.size());
int p =liste.size();
//liste in array umwandeln und an letzte stelle des feldes
//die gewrappte überschrift abspeichern
String[] listel = (String[]) liste.toArray(new String[p+1]);
listel[p] = f.titel.toString();
//für testzwecke
int leng1 = listel.length;
int c1 = 0;

for (c1=0; c1<(leng1); c1++)
{
System.out.println("array-felder "+ listel[c1]+ " , " + c1);
}
//rückgabe des feldes an swing_main
return listel;
}
}
```

6.5 Anhang E — Quelltext für die Klasse, die den Datenbankzugriff steuert.

```

/*****
 * Title:          DbKram.java
 *
 * Description:     Datenbankverbindungen und -abfragen und was sonst noch
 *                 so gebraucht wird
 *****/
import java.sql.*;
import java.util.*;

class DbKram
{
    static Connection conn = null;

    //klasse, die einen vorgänger zu einem konzept als string zurückliefert
    public String konzepte(String konzept)
    {
        String returne = "";
        String cmd1 = "SELECT vertreter, vorgaenger FROM geschichte where vertreter ='"+ konzept+"'";
        Statement query1 = null;
        ResultSet rs_query1 = null;

        try
        {
            query1 = conn.createStatement();
            rs_query1 = query1.executeQuery(cmd1);
            conn.commit();

            //ResultSet durchgehen
            while (rs_query1.next())
            {
                returne = rs_query1.getString(2);
            }
        }

        catch (SQLException e)
        {
            System.out.println("SQLException, " + e);
        }

        return returne;
    }

    public Set eintraege(String[] inhalt)
    {
        Set returner = Collections.synchronizedSet(new HashSet());

        if(connectDBServer()!=false)
        {
            int leng = inhalt.length;
            int c = 0;
            for (c=0; c<(leng); c++)
            {
                System.out.println("aktuelle Ausprägung: " + inhalt[c] + " , " + c);

                String cmd = "SELECT auspraegung, konzept FROM einstieg where auspraegung ='"+ inhalt[c]+'";
                System.out.println(cmd);
            }
        }
    }
}

```

```

Statement query = null;
ResultSet rs_query = null;

try
{
    query = conn.createStatement();
    rs_query = query.executeQuery(cmd);
    conn.commit();

    //ResultSet durchgehen
    while (rs_query.next())
    {
        String hilfstring = konzepte(rs_query.getString(2));String konzepthierarchie = hilfstring;
        String bla = "null";
        while (hilfstring.equals("ontologievater")==false)
        {
            hilfstring = konzepte(hilfstring);

            if (hilfstring.equals("ontologievater")==false)
            {
                konzepthierarchie = konzepthierarchie+"."+hilfstring;
            }
            System.out.println("konzepthierarchie danach  "+ konzepthierarchie);
        }

        returner.add("<meta name =\"Metadaten_aus_Ontologiewissen\"
        Originaltext =\"\" + rs_query.getString(1)+\"\"
        Ausprägung_von =\"\"+rs_query.getString(2)+\".\"+konzepthierarchie+\">\");
    }
}
catch (SQLException e)
{
    System.out.println("SQL - Fehler: "+ e);
}
return returner;
}
return null;
}

//*****
//schließt die connection zum Server
public static boolean closeDBServer()
{
    try
    {
        conn.close();
    }
}
catch (SQLException e)
{
    System.out.println("SchuERROR: Die Verbindung konnte
    nicht geschlossen werden: "+e.toString());
    return false;
}
return true;
}
//end of closeDBServer()

//*****
//öffnet eine Connection zum DB-Server
public static boolean connectDBServer()
{

```

```
        String DBUrl = "jdbc:interbase://localhost/C:/Studienarbeit/w4f/w4f/datenbank/gesch_m";
        try
        {
            Class.forName("interbase.interclient.Driver");
        }
        catch (Exception e)
        {
            System.out.println("SchuERROR: InterClient-Treiber konnten nicht geladen werden.");
            return false;
        }
        try
        {
            conn = DriverManager.getConnection(DBUrl, "sysdba", "masterkey");
        }
        catch (SQLException e)
        {
            System.out.println("SchuERROR: Der Server konnte nicht connectet werden: " + e.toString());
            return false;
        }
        return true;
    }
    //end of connectDBServer
}
```

6.6 Anhang F — Ontologietabelle.

VERTRETER	VORGAENGER
=====	=====
geschichtereignis	<null>
politik	geschichtereignis
wirtschaft	geschichtereignis
wissenschaft	geschichtereignis
gesellschaft	geschichtereignis
kunst	geschichtereignis
religion	geschichtereignis
philosophie	geschichtereignis
region	geschichtereignis
zeit	geschichtereignis
krieg	politik
aufstand	krieg
schlacht	aufstand
gegner	schlacht
sieger	gegner
verlierer	gegner
herrschaftsform	politik
herrscher	herrschaftsform
staat	herrscher
herrensitz	herrscher
partei	staat
verfassung	staat
gesetz	staat
mitglied	partei
burg	herrensitz
schloss	herrensitz
industrie	wirtschaft
landwirtschaft	wirtschaft
wirtschaftsordnung	wirtschaft
rohstoff	industrie
entdeckung	wissenschaft
erfindung	wissenschaft
technik	wissenschaft
entdecker	entdeckung
erfinder	erfindung
gesellschaftsordnung	gesellschaft
krankheit	gesellschaft
epedemie	krankheit
kultur	gesellschaft
architektur	kunst
skulptur	kunst
malerei	kunst
poesie	kunst
musik	kunst
kunstepoche	architektur
kunstepoche	skulptur
kunstepoche	malerei
kunstepoche	poesie
kunstepoche	musik
kuenstler	kunstepoche
werk	kuenstler
glaube	religion
einfluss_auf_staat_gesellschaft	religion
einrichtung	glaube
glaubensvertreter	glaube
philosoph	philosophie
werk	philosoph

zeitraum	zeit
zeitpunkt	zeit
anfang	zeitraum
ende	zeitraum
land	region
gebiet	land
stadt	land
stadtteil	stadt

6.7 Anhang G — Tabelle mit den Ausprägungen zu den Konzepten.

AUSPRAEGUNG =====	KONZEPT =====
Wiener	stadt
Schwerin	stadt
Rostock	stadt
Güstrow	stadt
Neustrelitz	stadt
mecklenburgischen	land
Mecklenburg-Strelitz	land
Mecklenburg-Schwerin	land
Preußen	land
Deutschen	land
Deutschland	land
Bundesstaaten	land
Österreichs	land
Dänemark	land
Herzöge	herrscher
Großherzöge	herrscher
Landesherren	herrscher
Großherzog	herrscher
Bismarck	herrscher
Bund	herrschaftsform
Großherzogtum	herrschaftsform
Ritterschaft	herrschaftsform
Reiches	herrschaftsform
Fürstenhaus	herrschaftsform
Königshaus	herrschaftsform
Bundesversammlung	verfassung
Verfassungssystem	verfassung
Verfassung	verfassung
Rechte	verfassung
Freiheiten	verfassung
Repräsentativvertretung	verfassung
Verfassungsentwicklung	verfassung
Verfassungen	verfassung
Landesverfassung	verfassung
Verfassungsreform	verfassung
Ständeverfassung	verfassung
Berfreiungskriege	aufstand
antinapoleonischen	aufstand
Revolution	aufstand
Niederlage	verlierer
Justizkanzleien	gesetz
Kriminalgesetzordnung	gesetz
Verordnung	gesetz
Regelung	gesetz
Gewerbegesetz	gesetz
Staatsgrundgesetzes	gesetz
Landesklöster	einrichtung
Gutsbetrieben	landwirtschaft
landwirtschaftlichen	landwirtschaft
agrarwissenschaftlichen	landwirtschaft
ertragssteigernden	landwirtschaft
Thünen	landwirtschaft
Landwirt	landwirtschaft
Agrarökonom	landwirtschaft
gutswirtschaftlich	landwirtschaft
Konjunktur	wirtschaft

wirtschaftlicher	wirtschaft
Verkehrsnetzes	wirtschaft
Fernverkehrsstrassen	wirtschaft
Kunststraße	wirtschaft
Eisenbahnlinie	wirtschaft
Eisenbahnverbindung	wirtschaft
Eisenbahnverbindungen	wirtschaft
Güterverkehr	wirtschaft
Aktiengesellschaften	wirtschaft
Großbetrieb	wirtschaft
Gewerbe	wirtschaft
Zunftzwang	wirtschaft
Stände	wirtschaft
industrielle	industrie
Alban	industrie
Hochdruckdampfmaschine	industrie
Landmaschinen	industrie
Demmler	architektur
Baumeister	architektur
Stadtarchitekt	architektur
Reuter	kuenstler
Dichter	kuenstler
Stromtid	werk
Vereine	partei
Opposition	partei
Arbeitervereine	partei
liberale	partei
Prügelstrafe	gesellschaft
Auswanderungswelle	gesellschaft
Konflikt	krieg
Österreich/Preußen	krieg
militärischen	krieg
Truppen	krieg
deutsch-französischen	krieg

Tabellenverzeichnis

1	Eigenschaften von Knoten eines HTML-Baums	19
2	Zugriffsfunktionen der Knotentypen	20

Abbildungsverzeichnis

1	Prinzipielle Arbeitsweise eines Wrappers	13
2	Beispiel eines HTML-Baums	17
3	GETESS-Architektur	24
4	Semantische Relationierung textueller Informationen	26
5	Prinzip der Kombination von Wrappermethoden und wissensbasier- ten Verfahren	29

Literatur

- [GE99] GETESS: Ein Analyse- und Suchdienst für Texte im Internet, 2.IUK-Tage MV, Rostock 1999
- [MK97] Musciano, C.; Kennedy, B.: HTML - The Definitive Guide. O'Reilly Associates, Inc., Köln, 1.Auflage, 1997.
- [MS1] BMBF-Projekt GETESS, Arbeitsbericht zum Meilenstein 1, 1998
- [MS2] BMBF-Projekt GETESS, Arbeitsbericht zum Meilenstein 2, 1999
- [MS3] BMBF-Projekt GETESS, Arbeitsbericht zum Meilenstein 3, 2000
- [RG00] Gohla, R.: Integrierte WWW-Anfragesichten, Diplomarbeit an der Universität Rostock, Februar 2000.
- [SA00] Sahuguet, A.; Azavant, F.: W4F, 2000, URL <http://db.cis.upenn.edu/W4F/>.
- [TM01] Marugg, Thomas: Metadaten für Content-Indizierung und Wissenssicherung, Januar 2001,
URL <http://www.internetmanagement.ch/index.cfm/fuseaction/shownews/newsid/351/>
- [W4F] W4F's BNF, Grammatik der Spezifikationssprachen eines W4F-Wrappers,
URL <http://db.cis.upee.edu/W4F/>.