Studienarbeit

Integration von dynamischen Inhalten in Suchmaschinen mit Hilfe von Metadaten

Götz Waschk

3. September 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

Die momentan verbreiteten Suchmaschinen im Internet sind in der Lage, die Seiten zu durchsuchen, die im öffentlich zugänglichen Teil des Web vorhanden sind. Dabei kommt ein Crawler[1] zum Einsatz. In [K+] wird ein Crawler wie folgt definiert:

> Ein Crawler ist ein Programm, dass automatisch die Hypertextstruktur des Webs durchläuft und rekursiv alle referenzierten Dokumente lädt.
>
> Die Bezeichnung „rekursiv“ ist dabei nicht als Einschränkung des Verfahrens zum Durchlaufen der Linkstrukturen auf einen bestimmten Algorithmus zu verstehen, auch ein Programm, das Heuristiken für die Auswahl der zu besuchenden Dokumente benutzt, entspricht dieser Definition.

Dem Crawler wird eine Liste mit Start-URLs übergeben, die Dokumente unter diesen Adressen werden heruntergeladen und es werden rekursiv aus den enthaltenen Hyperlinks weitere URLs ermittelt.

Die Dokumente werden dann bezüglich relevanter Informationen ausgewertet, und diese in einer Datenbank[2] gespeichert.Über eine Anfrageschnittstelle, z. B. ein Web-Formular, ist dann der Zugriff auf die Informationen möglich.

Crawler können also Daten von allen URLs einsammeln, die ihnen mitgeteilt wurden oder die sie selbständig ermitteln konnten. Komplett ignoriert werden Suchformulare und Seiten, die eine Anmeldung oder vorherige Registrierung erfordern. Wünschenswert wäre es, auch die Inhalte von über das Internet zugänglichen Datenbanken in eine Suchmaschine zu integrieren. Dazu muss der Crawler Anfragen an die Datenbank stellen. Diese Studienarbeit beschäftigt sich mit der Implementierung einer Software, die Anfragen an eine Datenbank über Web-Formulare stellen und die dynamisch erzeugten Webseiten einsammeln kann.

In den folgenden Abschnitten werden die unterschiedlichen Formen dynamischer Inhalte analysiert und die Möglichkeiten, diese Daten in eine Suchmaschine einzubinden, dargestellt.

1.1 Integrationsmöglichkeit von Datenquellen

Man kann die Integrationsmöglichkeit von Datenbankinhalten anhand der Kooperationsmöglichkeit der Anbieterseite in drei Klassen einteilen, die verschiedene Vorgehensweisen bei der Implementierung erforderlich machen. Diese Klassifizierung entspricht der Darstellung in [Web01] :

[1] andere Bezeichnungen sind Spider oder Robot

[2] dabei muss es sich nicht um ein vollständiges DBMS handeln, es sind oft einfachere Speicherungsmethoden üblich

1. *Kooperative Anbieter* ermöglichen direkten strukturierten Zugriff auf eine zu indizierende Sicht des Datenbankinhaltes über eine standardisierte Schnittstelle wie JDBC. Dabei ist es möglich, die volle Funktionalität einer Anfragesprache auszunutzen.

2. *Nicht-kooperative Anbieter* stellen ausser der für den interaktiven Zugriff konzipierten Formularschnittstelle keine Anfragemöglichkeit auf die Datenbankinhalte zur Verfügung. Das Schema der Datenbank und die Funktionalität der Anfrageschnittstelle sind nicht bekannt.

3. *Eingeschränkt kooperative Anbieter* stellen Informationen zur Verfügung, die das Datenbankschema beschreiben und die Funktionsweise des Anfrageprogrammes dokumentieren. Der Zugriff erfolgt aber wie bei 2. über WWW-Formulare.

1.1.1 Integration von Daten kooperativer Anbieter

In [Tit98] wurde ein Realisierungsvorschlag gemacht, um Datenbankinhalte in die Suchmaschine Harvest zu integrieren. Dazu wird ein sogenannter Summarizer benötigt, der auf dem Rechner des Datenbankanbieters installiert wird. Dieses Programm stellt Datenbankanfragen, ermittelt die interessanten Attribute, fasst diese zusammen und stellt sie über das Internet zur Einbindung in die Suchmaschine zur Verfügung. Dieser Ansatz wurde inzwischen erfolgreich implementiert, allerdings kann er nur im oben beschriebenen *kooperativen Fall* eingesetzt werden, d. h. der Anbieter der Datenbank muss bereit sein, ein fremdes Programm zu installieren, dass Informationen an externe Rechner übermittelt. Viele Administratoren lehnen dies aus Sicherheitsgründen ab, sie ermöglichen nur den Datenbankzugriff über die Formularschnittstelle.

1.1.2 Integration von Daten nicht kooperativer Anbieter

Im *nicht kooperativen* Fall stehen als einzige Informationen die Daten des Anfrageformulares zur Verfügung. Leider können gängige Suchmaschinen keine Informationen indizieren, die in durchsuchbaren Datenbanken hinter HTML-Formular-Suchmasken versteckt sind. Dieser Teil des Web wird in [RGM01] als *Hidden Web* bezeichnet. Der „Standard for Robots Exclusion“ dient, wie in [K+93] beschrieben, unter anderem auch dazu, um Crawler von Seiten fern zu halten, die von Anfragen erzeugt wurden. Der Grund liegt darin, dass Crawler nicht in der Lage sind, Formularfelder sinnvoll auszufüllen und auch nicht feststellen können, ob sie eine Anfrage-URL schon eingesammelt haben, was zu Schleifen führen kann, die Netz- und Rechenlast verschwenden.

In [RGM01] wird ein Ansatz namens *Hidden Web Exposer* vorgestellt, der im Fall *nicht kooperativer* Anbieter das *Hidden Web* durchsucht. Diese Methode beruht auf einer sehr komplexen Analyse der Formulare durch Heuristiken und ist nicht auf den allgemeinen Fall übertragbar.

1.1.3 Integration von Daten eingeschränkt kooperativer Anbieter

Diese Studienarbeit beschreibt ein Konzept und die dazugehörige Implementierung eines Ansatzes, der nur eine *eingeschränkte Kooperation* durch den Datenbankanbieter voraussetzt. Dabei wird das Anfrageformular um zusätzliche Metadaten erweitert, die von einem geeigneten Programm ausgelesen werden können. Diese Daten enthalten alle Informationen, die benötigt werden, um mit einer minimalen Anzahl

von Anfragen alle für die Speicherung in einem Suchindex geeigneten Informationen zu ermitteln.

1.2 Dynamischer Web-Inhalt

Dieser Abschnitt definiert dynamischen und statischen Inhalt im Web und klassifiziert die dynamischen Inhalte nach den Gründen der Verwendung und anschließend nach den Arten der Erzeugung.

Eine statische Seite ist in fertiger Form auf dem Server gespeichert, sie wird als Antwort auf einen Anfrage an den Client übermittelt.

Im Gegensatz dazu wird eine dynamische Webseite erst *nach* dem Request durch den Nutzer generiert, es handelt sich dabei um die Ausgabe eines Programms, dass entweder auf dem Server oder dem Client läuft.

1.2.1 Gründe für die Verwendung dynamischer Seiten

Es sind drei Gründe üblich, die den Anbieter veranlassen, Inhalt dynamisch zu generieren: zeitkritische Informationen, Personalisierung von Inhalten und die Möglichkeit der Reaktion auf Nutzereingaben.

Zeitliche Dynamik Viele Seiten müssen in kurzen Abständen aktualisiert werden, z. B. Nachrichten- oder Aktienkursticker. Es wäre möglich, dieses Ziel durch häufiges Ändern von statischen Seiten auf dem Web-Server zu erreichen, diese Kategorie wird hier aber nicht betrachtet, sondern nur nach o. g. Definition dynamisch generierte Inhalte.

Diese Inhalte können von üblichen Crawlern eingesammelt werden, da sie nach aussen hin wie statische Inhalte wirken. Es ist allerdings wichtig, dass die Abstände zwischen den Besuchen der Seite durch den Crawler klein genug sind, da sonst die Informationen in der Datenbank der Suchmaschine veraltet sind.

Client-basierte Dynamik Seiten können für einen bestimmten Nutzer erstellt oder für ihn angepasst werden. Dies dient hauptsächlich dem Zweck der Personalisierung, es können Inhalt, Aussehen oder Verhalten einer Seite individuell verändert werden.

Diese Seiten werden nach einer Identifizierung, z. B. durch ein Login, aus auf dem Server gespeicherten Informationen und eventuell auf dem Client in Form von Cookies abgelegten Daten, dynamisch erzeugt. Viele Portale bieten diese Art der Personalisierung an, ein Beispiel dafür ist Slashdot,[3] der Nutzer kann dort entscheiden, aus welchen Themenbereichen er Artikel aufgelistet haben will.

Es ist möglich, aber in vielen Fällen nicht sinnvoll, personalisierte dynamische Seiten in eine Suchmaschine zu integrieren. Wenn dies doch erforderlich sein sollte, dann muss man einen Crawler verwenden, der die benötigte Authentifizierungsfunktionalität (z. B. HTTP mit Nutzername und Passwort oder die Übertragung von Cookies) unterstützt. Das dazu verwendete Programm wird in [RGM01] als *eingeschränkter Crawler* bezeichnet, der nur eine genau begrenzte Menge von Adressen durchsucht.

Dynamik auf Basis von Nutzereingaben Ein großer Teil der Seiten im Internet wird als Reaktion auf eine Nutzereingabe erzeugt, zum Beispiel als Antwort auf die Übertragung eines Formulars. Das Ausfüllen der Suchmaske einer Online-Datenbank hat als Ergebnis eine oder mehrere dynamisch generierte Seiten. Diese

[3] `http://www.slashdot.org`

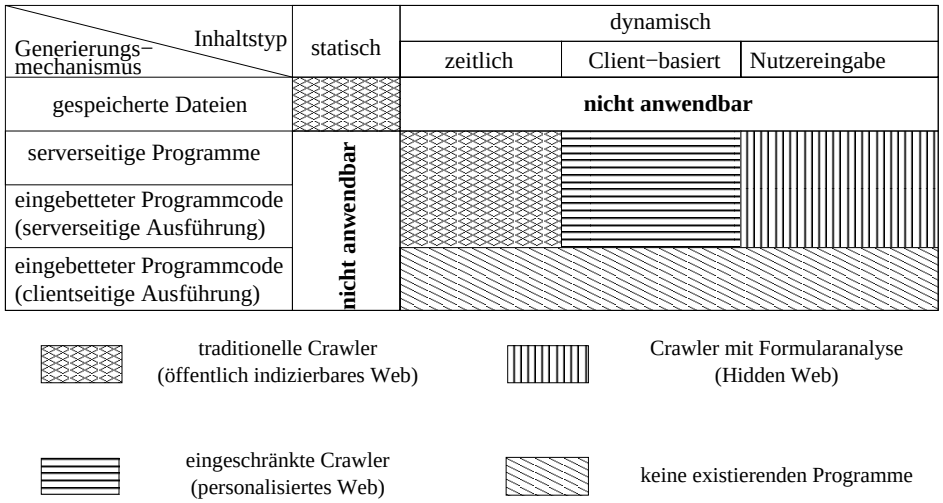

Abbildung 1.1: Klassifikation von Web-Inhalten in ihrer Wirkung auf die Verwendbarkeit von Crawlern

machen das sogenannte *Hidden Web* aus und sie sollen in die Suchmaschine integriert werden.

Diese Kategorien können auch kombiniert auftreten, vorstellbar wäre ein personalisiertes Portal mit Nachrichtenticker und mit Möglichkeit der Suche über ein Formular, dass alle drei o. g. Kriterien erfüllt.

1.2.2 Einteilung nach Art der Erzeugung

Statische Inhalte bestehen aus Dateistrukturen, die auf dem Webserver gespeichert sind. Dynamisch generierte Seiten können mit verschiedenen Mechanismen erzeugt werden.

Serverseitige Programme, z. B. CGI-Skripte, laufen auf einem Server, ihre Ausgabe ist im Web-Browser darstellbarer Inhalt, also HTML-Dokumente, GIF-Bilder o. ä. Oft muss die Formular-Anfrageschnittstelle benutzt werden, um auf diese Dokumente zuzugreifen.

Eingebetteter Programmcode mit serverseitiger Ausführung Unter diese Kategorie fallen alle Verfahren der Dokumenterstellung, bei denen in ein statisches HTML-Fragment Code-Schnipsel eingefügt werden, die vor der Übermittlung des Dokumentes durch ihre Ausgabe ersetzt werden. Beispiele für Sprachen, die für dieses Verfahren eingesetzt werden können, sind [PHP] und Server Side Includes (SSI) [Apa].

Eingebetteter Programmcode mit clientseitiger Ausführung In vielen Anwendungen wird Programmcode, der in HTML eingebettet ist, an den Client übertragen und vom Browser oder einer anderen Komponente ausgeführt. Dabei kommen am häufigsten die von Netscape entwickelte Sprache JavaScript bzw. ihre standardisierte Variante ECMAScript [ECM99] oder Java Applets zum Einsatz.

Ein Anwendungsbeispiel ist ein Warenkorb in einem Online-Shop, der auf Nutzerseite die ausgewählten Produkte darstellt und automatisch den Gesamtpreis berechnet.

Abbildung 1.1 fasst die beiden Einteilungen der Inhaltstypen und die Auswirkungen auf die Suchmöglichkeiten über Inhalten dieser Kategorien zusammen. Daraus lässt sich ablesen„ dass der Crawler, dessen Konzeption in dieser Studienarbeit dargestellt wird, das Suchen in Seiten ermöglichen soll, die *serverseitig*, entweder durch serverseitige Programme oder eingebetteten Programmcode mit serverseitiger Ausführung, als *Antwort auf Nutzereingaben* in ein HTML-Formular erzeugt werden.

1.3 Gliederung der Arbeit

Diese Studienarbeit beschäftigt sich mit der Konzeption und der Implementierung einer Software, die mit Hilfe der durch *eingeschränkt kooperative* Anbieter bereitgestellten Informationen Anfragen über ein Web-Formular stellt und die ermittelten Daten zur Integration in eine Suchmaschine aufbereitet.

In Kapitel 2 sind die Grundlagen zusammengefasst, die für die Verwendung der HTML-Formularschnittstelle benötigt werden. Kapitel 3 dokumentiert die zur Implementierung verwendete Software, Kapitel 4 beschreibt die Implementierung und die Konzepte, auf denen diese aufbaut. Insbesondere werden spezielle Metadaten definiert, die zur Einbindung von Web-Formularen verwendet werden. In Kapitel 5 wird schließlich der Ansatz ausgewertet.

Kapitel 2

Grundlagen

Dieses Kapitel definiert die wichtigsten theoretischen Grundlagen, die für die Verwendung von Formularen in automatischen Anfragen notwendig sind. Es werden Formulare formell modelliert und anschließend die konkreten Elemente, aus denen ein Formular bestehen kann, analysiert.

2.1 Modellierung von Formularen

Ein Formular F kann man als Menge von Tripeln der Form (*Element, Domäne, interne Bezeichner*) darstellen. In der Formel

$$F = \{(E_1, D_1, V_1), (E_2, D_2, V_2), \ldots (E_n, D_n, V_n)\}$$

ist E_i der interne Name eines Elementes, D_i die zugehörige Domäne und V_i die Menge der internen Bezeichner. Ein Formularelement kann neben einigen anderen einen dieser Typen haben: Texteingabefeld, Checkbox, Radio-Knopf, Auswahlliste, verstecktes Formularelement oder Submit-Knopf. Diese Typen werden in Abschnitt 2.2.2 genauer beschrieben.

Die Domäne ist die Menge der nach aussen sichtbaren Werte, die ein bestimmtes Element annehmen kann. Ein Element hat entweder eine bestimmte Domäne, d.h. im Formular werden bereits alle gültigen Werte der Domäne aufgelistet, oder eine unbestimmte Domäne, siehe dazu 2.2.2. In vielen Fällen ist die Zuordnung der Domäne zu einem Element nur schwer zu bestimmen, so bei Checkboxen oder Radio-Knöpfen. Siehe dazu den Abschnitt über die Checkbox.

Den meisten Formularelementen ist ausserdem noch ein beschreibender Text zugeordnet, den man als Label bezeichnet. Formal ist $L(E_i)$ das Label des Elementes E_i.

Diese an [RGM01] angelehnte formale Darstellung der Formularschnittstelle muss für die Modellierung der Übertragung der Formulardaten erweitert werden um die Menge der möglichen Rückgabewerte eines Formularelements, $R(E)$. Die Menge $R(E_i)$ lässt sich bei Elementen mit bestimmter Domäne aus den internen Bezeichnern $v_{ij} \in V_i$ des Formularelementes E_i ermitteln, $R(E_i)$ ist dann eine Menge von Tupeln (E_i, v_{ij}).

Die internen Bezeichner v_{ij} lassen sich immer aus dem HTML-Code von E_i ermitteln, die Werte haben aber bei den meisten Element-Typen nichts mit den nach aussen sichtbaren Werten der Domäne zu tun, die Ausnahmen werden in 2.2.2 erläutert.

2.2 Die HTML-Formularschnittstelle

HTML unterstützt zur Übertragung von Daten vom Nutzer zum Webserver Formulare in Webseiten, mit denen sich komplexe Interaktionen realisieren lassen. Dadurch ist es möglich, Suchanfragen an eine Datenbank mit Weboberfläche zu stellen.

2.2.1 Übertragung des Formularinhaltes

Die vom Nutzer in ein Formular eingetragenen Daten können über verschiedene Techniken ausgewertet werden. Üblicherweise werden die Daten an ein CGI-Programm auf einem Webserver mit einer der Methoden **GET** oder **POST**[1] übertragen. Andere Methoden, wie die Übertragung als Email werden an dieser Stelle nicht behandelt, da sie für diesen Anwendungsbereich keine Rolle spielen.

GET

Bei der GET-Methode wird der Rückgabewert des Formulars komplett in der URL der Anfrage übertragen. Die Formulardaten werden üblicherweise nach dem Typ `application/x-www-form-urlencoded` kodiert und nach einem Fragezeichen an die URL des Anfrageskriptes gehängt. Dadurch ist es für den Web Browser möglich, ein Suchergebnis mit einem Lesezeichen abzuspeichern.

Der Nachteil ist die Beschränkung durch die maximal mögliche Länge einer URL und durch die verwendete Kodierung eingeschränkte Auswahl übertragbarer Zeichen.

POST

Die Eingabewerte des Formulars werden im Körper einer HTTP-POST-Anfrage übertragen. Dadurch können größere Datenmengen übertragen werden und es sind andere Eingabekodierungen als bei einer GET-Anfrage möglich.

Die Ergebnisse einer POST-Anfrage lassen sich in den gebräuchlichen Browsern nicht mit einem Lesezeichen versehen.

2.2.2 Charakterisierung der Formularelemente

Alle hier aufgeführten Formularelemente verfügen über einen eindeutigen internen Bezeichner, mit dem der Rückgabewert gekennzeichnet wird. Dieser ist obligatorisch im Attribut `name` des HTML-Tags enthalten und entspricht E_i in der formalen Darstellung. Die Elemente sind meistens mit einem Text $L(E_i)$ versehen, der die Funktion des Elements für den Benutzer beschreibt. Dieses Label ist aber bei einigen Typen von Eingabeelementen nicht eindeutig zu einem Formularelement zugeordnet, d.h. der Nutzer kann diesen Zusammenhang über die visuelle Gruppierung von Formularelement und Labeltext herstellen. Der HTML-Standard Version 4.01 [WB+99] sieht zwar das Tag <label> für diesen Zweck vor, dessen Verwendung hat sich aber bislang nicht durchgesetzt. Die Formularelemente können nach mehreren Kriterien eingeteilt werden, wichtig ist die Unterteilung in Elemente mit bestimmter oder unbestimmter Domäne, da diese für die Auswertung des Formulars relevant ist.

Texteingabefeld

Das Texteingabefeld wird mit dem HTML-Tag `<input>` und optionalem Attribut `type="text"` erzeugt. Dieses Element besitzt eine unbestimmte Domäne, der

[1] Es handelt sich um die HTTP-Befehle, siehe [RI+97]

Nutzer kann beliebigen Text, nur eingeschränkt von der maximalen Länge des Eingabefeldes, eintragen. Die Menge der internen Bezeichner V_i ist mit der Domäne D_i identisch. Da die Domäne nicht automatisch bestimmt werden kann, müssen die möglichen Werte entweder manuell aufgelistet oder auf andere Weise generiert werden.

Das Tag <textarea> entspricht in seiner Funktion dem einfachen Texteingabefeld, es ermöglicht nur zusätzlich die Eingabe mehrzeiligen Textes.

Checkbox

Das HTML-Tag <input type="checkbox"> erzeugt eine Checkbox, die eine bestimmte Domäne besitzt. Das Attribut value enthält den internen Wert, ist es nicht vorhanden, dann wird value="on" angenommen. Rückgabewert ist entweder ein Tupel (E_i, v_{ij}), wobei $v_{ij} \in V_i$ der Wert des value-Attributs ist, oder die leere Menge, abhängig davon, ob die Checkbox aktiviert ist oder nicht, also $R(E_i) = \{\emptyset, \{(name, value)\}\}$.

Wenn mehrere Checkboxen mit dem gleichen Bezeichner existieren, bilden diese eine Gruppe, in der eine beliebige Kombination der einzelnen Checkboxen aktiviert werden können. Für jedes aktivierte Element wird dann ein Tupel (E_i, v_{ij}) übermittelt.

Formal ergibt sich dann $R(E_i)$ bei n Elementen als eine Menge von Teilmengen der Vereinigung aller $v_{ij} \in V_i$.

$$\begin{aligned} R(E_i) \quad = \quad & \{\emptyset, \{(E_i, v_{i0})\}, \{(E_i, v_{i1})\}, \ldots \\ & \{(E_i, v_{i0}), (E_i, v_{i1})\}, \ldots, \{(E_i, v_{i0}), \ldots, (E_i, v_{in})\}\} \end{aligned} \tag{2.1}$$

Die Domäne einer Checkbox lässt sich schwer automatisch zuordnen, da kein eindeutiger Zusammenhang zwischen dem HTML-Element, dass die Checkbox erzeugt und dem HTML-Fragment, das die Domäne dieser Checkbox beschreibt, besteht. Dieser Zusammenhang wird durch den Benutzer hergestellt, der den dicht neben der Box stehenden Text als dessen Domäne erkennt.

Radio-Knopf

Ein Radio-Knopf wird durch das Tag <input type="radio"> erzeugt, normalerweise sollten mehrere Knöpfe mit dem gleichen Namensattribut existieren, die dann eine Gruppe bilden, in der immer genau ein Knopf aktiviert ist. Die Domäne ergibt sich aus der Vereinigung der Werte aller Knöpfe einer Gruppe. Die Menge der Rückgabewerte ist dann die Vereinigung aller value-Attribute.

Das gleiche Problem wie bei der Domäne einer Checkbox liegt auch beim Ermitteln der Domäne eines Radioknopfes vor.

Auswahlliste

Eine Auswahlliste besteht aus dem Tag <select> mit mehreren durch dieses Tag eingeklammerten Tags vom Typ <option>. Als Besonderheit dieses Formularelementes lässt sich die Domäne leicht ermitteln, es handelt sich dabei um die Vereinigung der Textfragmente unterhalb der <option>-Tags.

Die Menge der Rückgabewerte ist aus den value-Attributen der <option>-Tags ermittelbar, wenn diese vorhanden sind. Andernfalls ist sie mit der Domäne identisch. Ist das Attribut multiple vorhanden, können mehrere Optionen aktiv sein. Ist dies bei der Übermittlung des Formulars der Fall, dann liefert das Formularelement mehrere Tupel (E_i, v_{ij}) zurück, analog der Menge der Rückgabewerte bei Checkboxen.

Versteckte Formularelemente

Oft tauchen in Formularen Eingabefelder der Form <input type="hidden"> auf. Diese haben eine genau bestimmte Domäne der Mächtigkeit 1, die explizit über das Attribut value angegeben wird und mit der Menge der Rückgabewerte identisch ist..

Der Zweck liegt in der Übertragung interner Parameter[2] und hat dann einen Sinn, wenn das Dokument, das das Formular enthält, dynamisch generiert wird. Die automatische Abfrage eines Formulars sollte es ebenfalls beinhalten, da sonst eventuell eine ungültige Anfrage erzeugt wird.

Submit-Knöpfe

Submit-Knöpfe haben eine Sonderrolle. Damit ein Formular funktioniert, muss mindestens ein Submit-Knopf vorhanden sein, der die Übermittlung der Daten auslöst. Ein Formular kann mehrere Knöpfe enthalten, von denen aber pro Anfrage nur einer aktiviert werden kann.

Zur Erzeugung eines Submit-Knopfes stehen zwei Möglichkeiten zur Verfügung: ein Eingabefeld <input type="submit"> oder das in HTML 4 [WB+99] definierte, aber nicht sehr verbreitete Tag <button type="submit">.

Ob ein Submit-Knopf einen Wert zurückliefert, hängt davon ab, ob das erzeugende HTML-Tag das optionale Attribut name besitzt. Dann liefert der Knopf ein Tupel (E_i, v_i) zurück mit $V_i = \{v_i\}$. Dabei ist v_i der Inhalt des Attributs value. Die Domäne ist dann mit dem internen Wert des Elements identisch, da des Inhalt von value auch als Beschriftung des Knopfes verwendet wird.

Für die Anfragegenerierung ist die Menge der möglichen Rückgabewerte[3] wichtig, sie lässt sich als Vereinigung von Tupeln (E_i, v_i) berechnen, mit $\emptyset$ als Wert für alle Submit-Knöpfe ohne internen Bezeichner E_i.

Sonstige Formularelemente

Neben den bereits genannten existieren noch andere Typen von Formularelementen, die aber an dieser Stelle nicht berücksichtigt werden, da sie für die Übermittlung von Formularinhalten keine Relevanz besitzen. Ein Reset-Knopf hat keine Bedeutung für die Formularübertragung, er hat die clientseitige Funktion des Zurücksetzens der Formularelement auf die Vorgabewerte. Passwort-Eingabefelder unterscheiden sich von den Texteingabefeldern nur darin, dass der eingegebene Text nicht auf dem Bildschirm ausgegeben wird, sie tauchen aber in öffentlichen Suchmasken normalerweise nicht auf.

[2] z. B. einer Nutzerkennzeichnung oder einer Session-ID

[3] Es existiert nur eine Menge $R(E)$ für die Zusammenfassung aller Submit-Knöpfe

Kapitel 3

Beschreibung verwendeter Werkzeuge

Dieses Kapitel beschreibt die zur Implementierung verwendeten Softwaresysteme in Aufbau und Funktionsweise. Diese Studienarbeit baut auf dem SWING-Projekt[1] auf, das eine verteilte Suchmaschine auf Basis von Harvest entwickelt. Dieses System wurde daher auch als Basis für die Implementierung gewählt, dadurch wird es möglich, die Suchergebnisse in SWING zu integrieren. Der folgende Abschnitt befasst sich mit der Architektur von Harvest und speziell mit den Erweiterungsmöglichkeiten, die für die Integration einer neuen Datenquelle in Form automatischer Web-Anfragen genutzt werden können.

Die deklarativen Extraktionsregeln der HEL[2] machen es dem Nutzer einfach, strukturierte Informationen aus Webseiten zu ermitteln, darum war es naheliegend, von W4F generierte Wrapper in das Konzept zu integrieren. Abschnitt 3.2 beschreibt die Funktionsweise von W4F.

3.1 Harvest

Bei Harvest handelt es sich um ein frei verfügbares Softwaresystem zum Einsammeln, Extrahieren, Organisieren und Suchen von Informationen im Internet. Es besitzt eine Komponentenstruktur, die einerseits vielfältige Anpassungsmöglichkeiten eröffnet, andererseits aber auch das Verteilen verschiedener Einzelaufgaben auf mehrere Rechner ermöglicht.

Das SWING-Projekt[3] nutzt diese Eigenschaften, um eine verteilte Suchmaschine zu entwickeln. Im folgenden Abschnitt wird die in [HSWL01] dokumentierte Funktionsweise von Harvest erläutert.

3.1.1 Komponenten

Gatherer Der Gatherer sammelt Dokumente von verschiedenen Datenquellen, extrahiert Metainformationen, wie z. B. Schlüsselworte, Autorennamen und Titel und speichert diese im SOIF (Summary Object Interchange Format) ab. Dabei handelt es sich um einen Menge aus Attribut-Wert-Paaren, das Format wird in Anhang A formal beschrieben. Die SOIF-Dokumente können dann in Form eines Datenstromes von Brokern abgerufen werden.

[1] Suchdienst für WWW-basierte Informationssysteme der nächsten Generation
[2] HTML Extraction Language, ein Teil von W4F
[3] http://swing.informatik.uni-rostock.de

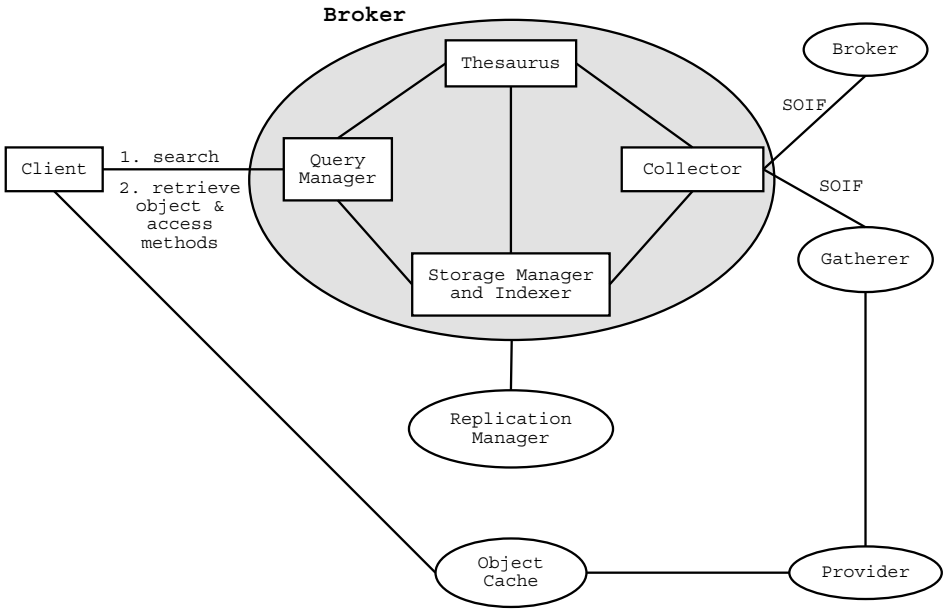

Abbildung 3.1: Architektur von Harvest

Broker Das Broker-Subsystem liest zu indizierende Informationen von einem oder mehreren Gatherern und entfernt mehrfach vorhandene Daten. Mittels *Index/Search Subsystem* indiziert er die gesammelten Informationen inkrementell[4] und stellt sie über eine WWW-Anfrageschnittstelle zur Verfügung.

Index/Search Subsystem Über die allgemeine Broker-Indexer-Schnittstelle ermöglicht Harvest die Verwendung verschiedener Retrievalsysteme. Die Standardeinstellung benutzt Glimpse, es werden aber auch Swish, FreeWAIS und ZQuery unterstützt.

Das Retrievalsystem erzeugt einen Index über die Daten, die der Broker überträgt und führt die Suchanfragen mit diesem Index durch. Von den Fähigkeiten des verwendeten Systems hängt der Funktionsumfang der Anfrageschnittstelle des Brokers ab. So sind bei Verwendung von Glimpse sogar strukturierte Anfragen möglich, die innerhalb eines bestimmten SOIF-Attributs suchen. Ein Beispiel dafür wäre

```
Title : Anfrage
```

zum Durchsuchen aller SOIF-Attribute *Title* nach dem String `Anfrage`.

Object Cache Der in Abbildung 3.1 dargestellte Object Cache ist nicht mehr Teil von Harvest, sondern wurde als eigenständiges Programm unter dem Namen Squid[5] ausgegliedert. Er ist für die Funktion von Harvest nicht zwingend erforderlich, sondern dient der Beschleunigung der Operationen durch Zwischenspeicherung von HTTP-, FTP- und DNS-Anfrageergebnissen.

Replikation Die Datenbanken der Broker können repliziert werden, um eine Lastverteilung zu ermöglichen. Ein Broker kann seine Daten nicht nur von einem oder mehreren Gatherern beziehen, sondern auch einen oder mehrere Broker abfragen. Diese Operation kann auch durch eine Anfrage gefiltert werden.

[4] das bedeutet, nur die neuen oder geänderten Daten werden indiziert
[5] http://squid.nlanr.net/Squid/

3.1.2 Funktionsweise des Gatherers

Der Gatherer von Harvest ist modular aufgebaut. Für das Hinzufügen eines neuen Dateityps, dessen Daten zusammengefasst werden können, steht eine Erweiterungsschnittstelle zur Verfügung.

Für den neuen Typ muss ein Summarizer bereitgestellt werden, ein Programm, das die Datei einliest und die extrahierten Daten als SOIF-Fragment, bestehend aus einer Attribut-Wert-Liste[6], ausgibt. Dem Summarizer wird dabei der Name der zusammenzufassenden Datei als Kommandozeilenparameter übergeben, das SOIF-Fragment muss über die Standardausgabe zurückgegeben werden.

Die Komponente *Essence* ist für die Erkennung des Dateityps und den Aufruf des passenden Summarizers zuständig. Um den Typ einer Datei festzustellen, hat Essence mehrere Möglichkeiten. Drei Methoden werden für HTTP-URLs nacheinander durchprobiert, bis der Typ ermittelt werden konnte. Anschließend wird entschieden, ob das Objekt eingesammelt werden soll und wenn ja, welcher Summarizer aufgerufen werden muss.

1. **Typerkennung durch URL** Die URL wird mittels regulärer Ausdrücke geprüft. In der Standardkonfiguration wird diese Methode hauptsächlich benutzt, um eine Reihe von Objekten frühzeitig auszuschließen, beispielsweise von Anfrageskripten generierte Seiten.

2. **Typerkennung durch Namen** Der Dateiname wird ebenfalls mit Hilfe regulärer Ausdrücke ausgewertet. Der Typ kann dann aus einem bekannten Namenspräfix oder -suffix ermittelt werden.

3. **Typerkennung durch Inhalt** Bei der letzten und aufwendigsten Methode[7] zur Feststellung des Objekttyps werden Daten aus der Datei gelesen und mit vorher definierten Merkmalen verglichen. Die dabei zur Anwendung kommende Methode ist die des UNIX-Programms file.[8] In einer Datei magic[9] werden die zur Erkennung vorgesehenen charakteristischen Bytefolgen und der Offset, an dem sie in der Datei stehen, zugeordnet.

Jede dieser Methoden wird über ihre eigene Konfigurationsdatei gesteuert, in der einem charakteristischen Merkmal ein Inhaltstyp zugeordnet wird. Wenn der Typ festgestellt werden konnte, wird in der Datei `stoplist.cf` nachgesehen, ob dieser dort vorhanden ist, wenn nicht, wird entweder ein Summarizer aufgerufen, dessen Dateiname aus dem Inhaltstyp und der Endung `".sum"` besteht, oder die in `quick-sum.cf` angegebenen regulären Ausdrücke zur Erzeugung von Attribut-Werte-Paaren werden angewendet.

Nachbearbeitung der zusammengefassten Daten

Der Gatherer ermöglicht es, zur „Feinabstimmung“ die Ausgabe der Summarizer zu verändern. Zu diesem Zweck existiert ein einfaches Regelsystem, das Operationen auf SOIF-Attributen erlaubt. Die Regeldatei muss in der Konfigurationsdatei des Wrappers unter `Post-Summarizing:` eingetragen werden.

Dadurch können die Eigenschaften spezieller Objekte ausgenutzt werden, ohne dass der Summarizer angepasst werden muss. Ein Beispiel ist die Anpassung des TTL-Wertes[10] einer URL, wenn die Lebensdauer von dem Standardwert abweicht. Eine Beispielregel ist in Abbildung 3.2 dargestellt. Diese setzt die Lebensdauer von Seiten, deren Titel *News* enthält, auf drei Tage herab.

[6] siehe Tabelle A.1 für die formale Beschreibung ATTRIBUTE-LIST
[7] es muss evtl. ein Netzzugriff erfolgen
[8] siehe Manpage file(1)
[9] das Dateiformat wird in der Manpage magic(5) beschrieben
[10] Time-to-Live – Zeitdauer zwischen Einsammeln eines Objekts und dessen Löschung

```
title ~ /News/
        time-to-live="259200"
```

Abbildung 3.2: Änderung der Lebensdauer von Objekten über eine Post-Summarizing-Regel

Regelmäßige Aktualisierung der Daten

Damit die Suchanfragen an Harvest zu den gewünschten Ergebnissen führen, muss das Einsammeln der Daten durch den Gatherer regelmäßig wiederholt werden. Dabei müssen nur die Objekte neu zusammengefasst werden, die sich geändert haben. Um dies zu erkennen, speichert *Essence* mehrere Attribute in jedes SOIF-Dokument:

- `Update-Time` ist der Zeitpunkt der Einspeicherung des Objektes bzw. der letzten Aktualisierung.
- `Refresh-Rate` ist die Wiederauffrischungsrate, nach diesem Zeitintervall wird geprüft, ob sich das Objekt geändert hat.
- `Time-to-Live` ist die Lebensdauer eines Objektes in der Datenbank. Wenn sie abgelaufen ist, wird es sofort gelöscht und muss neu eingesammelt werden.
- `Last-Modification-Time` dient zum Erkennen von modifizierten Dateien, es handelt sich um einen Zeitstempel der letzten Änderung. Der Gatherer kann das Kommando `If-Modified-Since` [RI+97] im HTTP-Header verwenden, um nur veränderte Daten neu zu laden.
- `MD5` speichert eine Prüfsumme des Objekts nach dem *Message Digest 5* [Riv92] und erlaubt die Erkennung geänderter Objekte, wenn die Datei schon lokal vorliegt.

3.2 W4F

W4F[11] (World Wide Web Wrapper Factory) ist ein Werkzeug zum schnellen Generieren von Wrappern für Web-Datenquellen. Es wurde an der Universität Pennsylvania von der Telecom Paris (E.N.S.T.) entwickelt.

Das System besteht aus einem Übersetzer für Wrapper-Definitionsdateien und einem Laufzeitsystem, jeweils implementiert in Java. Es besitzt die im folgenden Abschnitten dargestellte modulare Schichten-Architektur.

In [Goh00] werden mehrere Systeme zum Generieren von Wrappern verglichen, dort besticht W4F durch sein Modularisierungskonzept und durch die Leistungsfähigkeit der Extraktionsschicht. Die regelbasierte Extraktionssprache ist vom Umfang kompakt und dadurch leicht zu lernen, gleichzeitig aber sehr leistungsfähig und im Gegensatz zu den in anderen Systemen verwendeten prozeduralen Skriptsprachen eher für wiederverwendbare Extraktionslösungen geeignet.

Auch die direkte Integration der ermittelten Daten in Java-Programme ist ein entscheidender Vorteil.

3.2.1 Retrieval-Schicht

Diese Komponente ist für das Laden einer HTML-Seite zuständig. Regeln der Retrieval-Sprache[12] geben an, von welcher Adresse (aus dem WWW über HTTP oder

[11] ursprünglich die Abkürzung für **W**ysi**W**yg **W**eb **W**rapper **F**actory
[12] diese ist in [Sah99] dokumentiert

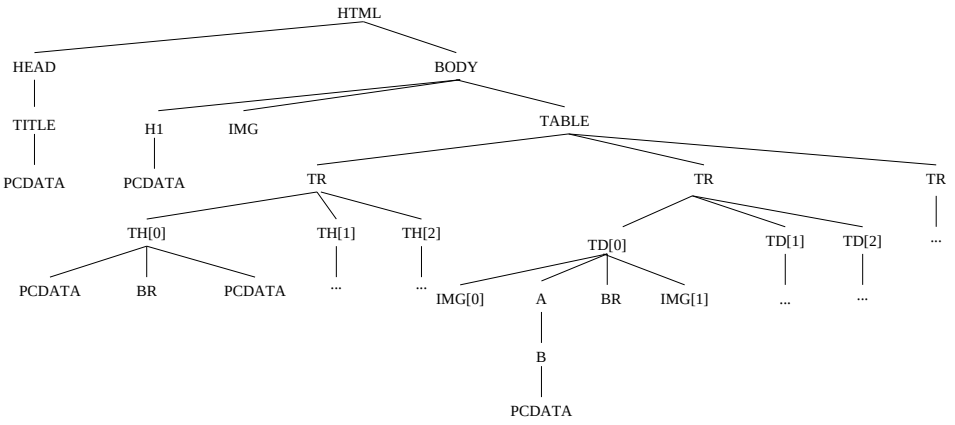

Abbildung 3.3: Beispiel für Analysebaum

aus einer lokalen Datei) ein Dokument bezogen wird und welche Parameter für das Anfrageprogramm verwendet werden sollen. Dies geschieht vollkommen transparent für die anderen Schichten, die Ausgabe ist letztendlich ein HTML-Dokument.

Ein Beispiel für eine Retrieval-Regel ist in Abbildung 4.2 auf Seite 28.

3.2.2 Extraktionsschicht

Die Extraktionsschicht verwendet Regeln der HEL (HTML Extraction Language), um die gewünschten Textelemente aus der Struktur einer HTML-Seite zu ermitteln. Zu diesem Zweck wird ein Parser verwendet, der das HTML-Dokument eineindeutig auf einen Analysebaum abbildet. Dabei wird die hierarchische Struktur des Dokumentes gemäß *Document Object Model* [WB+98] ausgenutzt.

Der Baum besitzt eine Wurzel mit dem Bezeichner `html` und Knoten, die mit den HTML-Tags und Textstücken (die mit dem Bezeichner `PCDATA` versehen werden) des Dokumentes korrespondieren. Geschlossene HTML-Tags[13] werden auf interne Knoten abgebildet, deren Kind-Knoten aus dem Textabschnitt innerhalb des Elementes abgeleitet werden. Blattknoten sind entweder offene HTML-Tags (z. B. `img` oder `br`) oder `PCDATA`-Knoten. Kindknoten besitzen einen Index, der sie in der Reihenfolge ihres Auftretens, beginnend bei 0, durchnummeriert. Die Vergabe der Indizes erfolgt nach dem *left-depth-first-Prizip*, die Nummerierung gleichnamiger Knoten erfolgt zuerst links absteigend.

In Abbildung 3.3 ist der HTML-Baum für das in Abbildung 3.4 dargestellte Dokument angegeben.

3.2.3 HTML Extraction Language

Die Extraktionsschicht verwendet die HTML-Extraktionssprache, deren Grammatik in Anhang B formal beschrieben wird. In den Regeln dieser Sprache werden Pfadausdrücke zur Navigation auf dem Analysebaum genutzt. Dabei kommen zwei verschiedene Operatoren zur Anwendung.

`"."` Der Punkt-Operator dient zur Navigation entlang der Dokumenthierarchie. Der Pfadausdruck `html.head[0].title[0]` führt zum ersten <title>-Tag unterhalb des ersten <head>-Tags. Die Indizes können in diesem Fall auch weggelassen werden, bei einem Bezeichner ohne Index wird `0` angenommen.

[13] bestehen aus einem öffnenden und einem schließenden Tag, z. B. `<a>...</a>`

```
<html>
  <head><title>Beispiel</title></head>
  <body>
    <h1>Überschrift</h1>
    <img>
    <table>
      <tr>
        <th>Tabellen-<br>überschrift</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>
```

Abbildung 3.4: HTML-Code für den in Abbildung 3.3 dargestellten Analysebaum

Auf diese Weise kann man jedem Element der Baumstruktur einen eindeutigen Namen zuordnen. In der Praxis hat diese Form des Zugriffs auf die Struktur den Nachteil, dass sie nicht robust gegenüber kleineren Änderungen, z. B. dem Einfügen von Tags zur Anpassung des Layouts, sind.

`"->"` Der Pfeil-Operator ermöglicht es, dem Dokumentfluss zu folgen, die Navigation folgt der Anzeigereihenfolge der Tags. Ein Pfadausdruck `html->b[0]` führt von der Wurzel `html` zum ersten <b>-Tag, das beim Durchlaufen des Analysebaumes nach dem Prinzip der Tiefensuche gefunden wird.

Der Vorteil bei der Verwendung des Pfeil-Operators liegt in der Möglichkeit, Tags zu überspringen und von einem Teilbaum zu einem anderen zu springen. Dadurch können Pfadausdrücke gebildet werden, die robuster auf kleinere Änderungen der Dokumentstruktur reagieren.

Extraktion strukturierter Daten

Die HEL bietet zwei Sprachelemente zum Datenermittlung aus strukturierten, auch irregulär verschachtelten HTML-Tags. Dabei handelt es sich um Indexbereiche und den `"#"`-Operator.

Indexbereiche erlauben die Erfassung von mehreren Tags des gleichen Typs über ihre am Anfang dieses Abschnittes beschriebene Nummerierung. In Indexbereichen ist die Aufzählung der gewünschten Werte, die Angabe von Intervallen oder die Verwendung von ***** als Platzhalter möglich. Hier sind ein paar gültige Indexbereiche:

- `[0]` – nur ein Element, das mit der Nummer 0
- `[1,2,4]` – eine Aufzählung, die das zweite, dritte und fünfte Element erfasst.

```
<select multiple name="tt-bereich">
<option value="Alle">Alle Bereiche
<option value="1155">Automatisierungs-/ Meß-/Sensortechnik
<option value="1156">Bauwesen / Bautechnologie
<option value="1157">Bildung
...
</select>
```

Abbildung 3.5: HTML-Code einer Auswahlliste

```
select=html->form->select[*]
        (
        .getAttr(name) #
        .option[*].getAttr(value) #
        .option[*].txt #
        .getAttr(multiple)
        );
```

Abbildung 3.6: Extraktionsregel, die alle Auswahllisten ermittelt

- `[3-6]` – ein Intervall, das vom vierten bis einschließlich dem siebten Element reicht
- `[-4]` – Kurzform für `[0-4]` – ein Intervall vom ersten bis zum fünften Element
- `[0,3-]` – eine Kombination aus dem ersten Element und dem Intervall vom vierten bis einschließlich dem letzten Element
- `[*]` – alle Elemente

Der "#"-Operator wird auch als *Fork*-Operator bezeichnet, weil er eine Aufspaltung der Suche im Analysebaum und damit die gleichzeitige Verfolgung mehrerer Pfade, die von einem Knoten ausgehen, ermöglicht. Abbildung 3.5 zeigt den HTML-Code einer Auswahlliste und Abbildung 3.6 die dazugehörige Extraktionsregel. Diese Extraktionsregel ermittelt die in Abbildung 3.7 dargestellte Struktur der Auswahlliste als *NestedStringList*, siehe 3.2.4.

Bedingungen

Eine Extraktionsregel kann mit einer oder mehreren Bedingungen[14] versehen sein, die Pfadausdrücke mit Strings vergleichen. Diese Operatoren stehen dabei zur Verfügung:

- `"="` Test auf Gleichheit
- `"!="` Test auf Ungleichheit
- `"=~"` Test auf Passen eines regulären Ausdrucks mit Perl-Syntax[15]
- `"!=~"` Test auf Nicht-Passen eines regulären Ausdrucks

Der Formale Aufbau der Bedingungen steht in Anhang B, Abbildung 3.8 zeigt eine Extraktionsregel mit zwei Bedingungen, die genutzt wird, um ein Meta-Tag zu

[14] wenn mehr als eine Bedingung vorhanden ist, müssen diese mit AND verknüpft werden
[15] siehe Unix-Manpage perlre(1)

```
select : [
   [
      tt-bereich
      [
         Alle
         1155
         1156
         1157
         ...
      ]
      [
         Alle Bereiche
         Automatisierungs-/ Meß-/Sensortechnik
         Bauwesen / Bautechnologie
         Bildung
         ...
      ]
      #DEFAULT
   ]
]
```

Abbildung 3.7: NestedStringList zum HTML-Code in Abbildung 3.5

```
navigation = html->meta[i].getAttr(content)
        where html->meta[i].getAttr(name) = "DB.Navigation"
        and html->meta[i].getAttr(content) =~ "YES|NO";
```

Abbildung 3.8: Extraktionsregel mit Bedingungen

ermitteln, dessen Namensattribut den Wert `DB.Navigation` hat und dessen Attribut `content` entweder `YES` oder `NO` ist.

3.2.4 Abbildungsschicht

Die in der bei der Extraktion gewonnenen Daten werden zur Weiterverarbeitung auf Datentypen abgebildet. Die von W4F verwendete Repräsentation ist die *NestedStringList* (NSL), die in Abbildung 3.9 definiert ist. Dieser Datentyp ermöglicht

```
NestedStringList ::= String
                   | listOf (NestedStringList)
                   | null
```

Abbildung 3.9: BNF der NestedStringList

es, jede beliebig tief verschachtelte Struktur aufzunehmen, die von einer Extraktionsregel vorgegeben wird. Durch die Angabe von Schema-Regeln kann man diese interne Darstellung auf andere Formen abbilden. Eine Regel hat die Form einer Variablendeklaration, es steht eine Abbildung auf die primitiven Java-Datentypen `int` und `float`, auf die `String`-Klasse, auf selbst definierte Klassen sowie auf Arrays dieser Typen zur Verfügung.

Über die in [Sah99] dokumentierte API können Daten aus der *NestedStringList*

ermittelt werden. Die selbst definierte Klasse muss dann einen Konstruktor mit dieser Signatur besitzen:

```
public KlassenName(NestedStringList nsl)
```

Ebenfalls möglich ist eine Abbildung von NSL auf XML durch deklarative Regeln in Form von Templates, auf die hier aber nicht weiter eingegangen werden soll.

Kapitel 4

Konzeption und Implementierung

Dieses Kapitel beschreibt die Implementierung eines Crawlers, der die Anfragen über die HTML-Formularschnittstelle stellen kann. Zunächst werden die Metadaten erläutert, mit denen *eingeschränkt kooperativen Anbieter*[1] ihre Formulardateien anreichern müssen.

Anschließend wird die Implementierung in Form eines aus den Metadaten eines Formulars dynamisch generierten W4F-Wrappers dokumentiert.

4.1 Metadaten

Im diesem Abschnitt werden die Metadaten beschrieben, die bei der Einbindung von Angeboten eingeschränkt kooperativer Datenbankanbieter Verwendung finden. Das Konzept wurde aus [Web01] übernommen. Die Elemente werden in das HTML-Tag <meta> eingebettet, alle HTML-spezifischen Sonderzeichen und Umlaute, vor allem aber < und > müssen durch ihre HTML-Entitites ersetzt werden.

Ausserdem müssen alle Elemente mit dem Präfix **DB.** anfangen, um sie von Metadaten anderer Programme besser zu unterscheiden.

In der Tabelle 4.1 werden Nichtterminalsymbole in spitzen Klammern dargestellt, Terminalsymbole erscheinen im Fettdruck. Alternativen werden durch "|" getrennt, null- oder mehrmalig auftretende Symbole stehen zwischen geschweiften Klammern.

ConfigFile Verweist auf eine optionale Konfigurationsdatei, in der die datenbankspezifischen Metadaten abgelegt werden. Diese Datei kann mit eingeschränkten Zugriffsrechten ausgestattet werden, um nur bestimmten Suchmaschinen die Abfrage des Datenbankinhaltes zu gestatten. Das Format der Konfigurationsdatei ist ebenfalls HTML mit den hier beschriebenen speziellen Metadaten. Dieses Element ist optional.

QueryScript Enthält die URL des Anfrageskriptes, sowie nach einem Komma den Übertragungstyp der Formulardaten, es werden POST und GET unterstützt.

Diese Information könnte auch durch Analyse des Formulars ermittelt werden, da der Anbieter aber ein anderes Anfrageprogramm angeben kann, dessen Ausgabe leichter auswertbar ist, wurde festgelegt, dass dieses Element obligatorisch ist. Dadurch ist es z. B. auch möglich, dass bei Layoutänderungen der Ergebnisseiten

[1] siehe Definition in Abschnitt 3 auf Seite 6

Bezeichner	**Wert**
ConfigFile	<URL_der_Konfigurationsdatei>
QueryScript	<URL_Script>, **GET** | **POST**
QueryElement Queryelements	<Element> <Element>, <Element> {, <Element> }
QueryElement.<Element>	/<Einfacher_regulärer_Ausdruck>/| <Element> {, <Element>}
Result	<gemeinsamer_Teil_einer W4F_Extraktionsregel>
Result.<Attributname>	<W4F_Extraktionsregel> | **{**<Attributwert> {, <Attributwert> }**}**
Navigation	**Yes** | **No**
NavigationURL	<URL_mit_Platzhalter_für_Parameter>
MaxPages	<Anzahl>
MaxQueries	<Anzahl>

Tabelle 4.1: verwendete Metadaten

einfach auf die alte Version verwiesen wird und die Extraktionsregeln somit nicht geändert werden müssen.

QueryElement Enthält das für die Anfrage wesentliche Formularelement. Sind mehrere Elemente wesentlich, so werden diese durch Kommata getrennt unter **QueryElements** eingetragen. Es muss immer eine der beiden Möglichkeiten angegeben werden. Sind beide vorhanden, dann hat *QueryElements* Vorrang.

QueryElement.<Element> Hier wird für das Anfrageelement <Element>, das eine unbestimmte Domäne besitzt, eine Regel zur Erzeugung in das Formular einzusetzender Parameter angegeben. Die Regel hat die Form eines regulären Ausdrucks, der allerdings eingeschränkt ist und nur eine endliche Ergebnismenge besitzt. Stattdessen kann aber auch eine durch Kommata getrennte Liste der möglichen Werte angegeben werden.

Wenn die minimale Anfrageoperation zur Extraktion des Datenbankinhaltes eine Einschränkung der zu verwendenden Wertemenge eines Formularelements erfordert, dann können die Werte ebenfalls in diesem Meta-Tag angegeben werden. Dabei kann es sich z. B. um einen Radio-Knopf handeln, der für die gewählte Anfrage nur in einer Einstellung Sinn macht, wie das Beispiel in 5.1.1 auf Seite 32 demonstriert.

Result Der gemeinsame Teil der W4F-Extraktionsregeln kann in diesem Element angegeben werden. Dies ist dann sinnvoll, wenn alle Ergebniselemente in einer Tabelle ausgegeben werden. Ansonsten ist diese Element optional.

Wenn *Result* angegeben wurde, dann gilt dieser Wert für alle nachfolgenden *Result.<Attributname>*-Tags, deren Extraktionsregeln nicht mit der Wurzel `html` anfangen. Dieses Meta-Tag kann mehrmals verwendet werden, der Wert wird dann für alle nachfolgenden *Result.<Attributname>*-Tags überschrieben.

Result.<Attributname> Für jedes Attribut, dessen Werte aus den Ergebnisseiten bestimmt werden sollen, wird entweder eine HEL-Regel angegeben, oder es werden alle möglichen Werte durch Kommata getrennt aufgelistet.

Die HEL-Regel kann jede Form annehmen, die von der in Anhang B angegebenen Grammatik akzeptiert wird.

Eine Auflistung aller Attributwerte ist dann notwendig, wenn das Attribut durch ein Formularelement mit schwer bestimmbaren, nach aussen sichtbaren Werten dargestellt wird (Checkboxen, Radiobuttons).

NavigationURL Wenn bei einer einzelnen Anfrage mehrere Ergebnisseiten ausgegeben werden, deren Inhalte jeweils ermittelt werden sollen, dann muss dieses Element die URL enthalten, über die zwischen den einzelnen Seiten navigiert werden kann. Dabei wird die URL in Form eines vereinfachten Glob-Musters[2] angegeben, um die Anfrageparameter, die die veränderlichen Bestandteile der Navigations-URL darstellen, mit zu erfassen. Würden die möglichen Links auf der Ergebnisseite nicht über diese Methode eingeschränkt werden, dann müssten die für die Navigation relevanten Links über Heuristiken[3] ermittelt werden.

Navigation Wenn dieses Element auf den Wert `YES` gesetzt wird, dann dürfen die Links auf den Ergebnisseiten, auf die der Ausdruck, der unter *NavigationURL* angegeben ist passt, weiterverfolgt werden.

MaxPages Wenn über mehrere Ergebnisseiten navigiert werden muss, dann wird die maximale Anzahl der einzusammelnden Teilseiten pro Anfrage hier festgelegt.

MaxQueries Der Anbieter kann mit diesem Eintrag die Anzahl der Anfragen über das Formular begrenzen. Ist dieser Wert kleiner als die eigentlich benötigte Anzahl, dann kann nicht die vollständige Datenmenge ausgelesen werden.

4.2 Generieren des Wrappers

Die Integration der Metadaten in die Formulardateien ermöglicht es, dynamisch W4F-Wrapper zu generieren, die Anfragen stellen können und die ermittelten Daten in den Harvest-Gatherer integrieren können. Die Funktionsweise dieses Systems ist in Abbildung 4.1 dargestellt. und wird in den folgenden Abschnitten erläutert.

Die Auswertung der Metadaten und der Formularelemente erfolgt über ein Java-Programm, dem die HTML-Datei mit den Metadaten als Parameter übergeben wird. Dieses Programm verwendet einen W4F-Wrapper, um alle relevanten Informationen aus der Datei zu extrahieren. Dies sind neben den Metadaten alle Formularelemente mit bestimmter Domäne, also alle Radio-Knöpfe, Checkboxen, versteckte Elemente und Auswahllisten.

Ist das Element *ConfigFile* vorhanden, dann wird zusätzlich über einen HTTP-Request die Konfigurationsdatei geladen. Diese wird ebenfalls durch den Wrapper geparst.

Aus den Metadaten und den Daten des Formulars wird dynamisch ein Wrapper Description File[4] (die formale Beschreibung des Formates steht in [Sah99]) erzeugt. Dieses wird dann durch einen Aufruf des Übersetzerprogrammes *w4f* in eine Datei mit Java-Quellcode und anschließend durch den Java-Compiler in eine Bytecode-Datei übersetzt.

Der Name der `.w4f`-Datei wird dabei aus einer Prüfsumme erzeugt, die vorher mit Hilfe des in [Riv92] definierten MD5-Algorithmusses aus den Daten des Formulars berechnet wurde. Dadurch wird verhindert, dass der Wrapper bei jedem Gatherer-Lauf neu erzeugt werden muss, wenn sich das Formular und die darin enthaltenen Metadaten nicht geändert haben.

[2] wir verwenden nur * und ? als Platzhalter

[3] z. B. das Verfolgen von Links, die ein Schlüsselwort wie „weiter“ oder eine Zahl als Text enthalten

[4] mit der Dateiendung `.w4f`

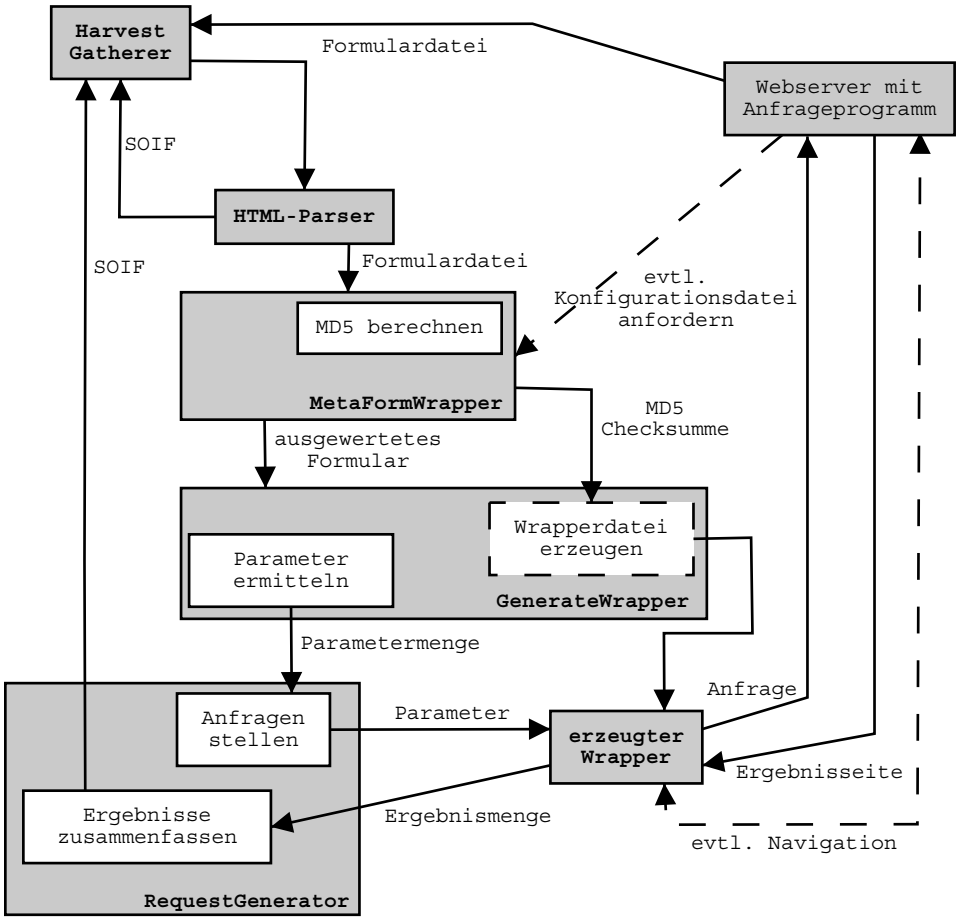

Abbildung 4.1: Ablaufplan des Metadaten-Crawlers

4.2.1 Erzeugung der Extraktionsregeln

Die Extraktionsregeln sind in den Metadaten in den `DB.Result*`-Tags vorgegeben, sie müssen evtl. nur noch vervollständigt werden, in dem der Inhalt der `Result`-Tags den HEL-Fragmenten der nachfolgenden `Result.<Attributname>`-Tags vorangestellt wird, um vollständige HEL-Regeln zu erhalten.

Zu beachten ist, dass die Attributnamen Zeichen beinhalten können, die in W4F bzw. in Java nicht als Variablenbezeichner erlaubt sind. Darum erhalten die Variablen, die in den Extraktionsregeln verwendet werden, generische Bezeichner.

4.2.2 Erzeugung der Retrieval-Regel

Die Retrieval-Regel ergibt sich aus den Angaben unter `QueryScript`, also der URL des Anfrageprogrammes und der Anfragemethode. Ausserdem relevant sind die Parameter des Anfrageprogrammes. Diese werden unterschieden in Parameter, die bei allen Anfragen konstant sind und solche, die verschiedene Werte annehmen können.

- Konstante Parameter haben für alle Anfragen den gleichen Wert, der sich aus der Analyse des Formulars ermitteln lässt. Entweder handelt es sich um den Rückgabewert eines Eingabefeldes vom Typ `hidden`, einen Submit-Knopf mit nur einem möglichen Rückgabewert oder um ein Formularelement E_i, dessen Rückgabewertemenge $R(E_i)$ durch einen Eintrag in den Metadaten in der Form `QueryElement.`$< E_i >$ mit nur einem angegebenen Wert eingeschränkt wurde.

  Diese Parameter und ihre Werte werden paarweise in die Retrieval-Regel als `"attribut" = "wert"` aufgenommen.

- Variable Parameter besitzen eine Menge $R(E_i)$ mit $|R(E_i)| > 1$. Sie werden in die Retrieval-Methode als Parameter aufgenommen.

Erwähnenswert ist ausserdem die Erweiterung des HTTP-Headers um ein Host-Feld, da dies von vielen Web-Servern vorausgesetzt wird. Näheres dazu findet man in [RI+97].

4.2.3 Erzeugung des nutzerdefinierten Java-Codes

Dieser Teil des generierten Wrappers enthält die Anweisungen zur Verarbeitung der Ergebnisse der Extraktionsregeln. Für jedes Attribut werden die ermittelten Werte, die zunächst vom Typ `NestedStringList` sind, von der eventuell verschachtelten Struktur in eine „flache“ Liste und anschließend in eine Menge von Strings umgewandelt, es findet also gleich eine Eliminierung doppelter Werte statt.

4.2.4 Wrapper mit Navigation über Ergebnisseiten

Ein Wrapper mit Unterstützung für die Navigation auf mehrteiligen Ergebnisseiten ist ein Sonderfall. Er erfordert Erweiterungen der zuvor beschriebenen Extraktions-, Retrieval-, und Java-Code-Regeln.

- Die Extraktionsregeln werden um einen Eintrag erweitert, der alle Links erfasst, die dem Muster im Metatag `NavigationURL` entsprechen.

- Die folgende Retrieval-Regel wird zum Laden der Navigationslinks verwendet. Dafür wird immer die `GET`-Methode genutzt.

```
navigate(String url) {
GET "$url$";
}
```

```
/* variable Parameter bilden die Parameterliste der Methode */
get(String p0,String p1) {
    METHOD: POST;
    URL: "http://www.nord-info.de/tt-meta/tt-meta.cgi";
    /* Der Host-Header ergibt sich aus Rechnername und Portnummer
    der URL des Anfrageprogrammes */
    HEADER: "Host: www.nord-info.de:80\r\n";
    PARAM:
    "host" = "",
        /* konstanter Parameter, der direkt in die Retrieval-Regel
        geschrieben wurde */
        "id" = "958728726",
        "choice" = $p0$,
        /* variabler Parameter mit Platzhalter für den beim Aufruf
        der Methode einzusetzenden Wert */
        "mv" = $p1$;
}
```

Abbildung 4.2: Beispiel für eine Retrieval-Regel

- Der nutzergenerierte Java-Code wird um einen „Mini-Crawler“ erweitert, der die Links in einer Mengenstruktur zusammenfasst und diese so lange verfolgt, bis die maximal erlaubte Anzahl an Requests[5] erreicht wurde oder keine noch nicht besuchten Links mehr vorhanden sind. Die extrahierten Daten werden ebenfalls pro Attribut zusammengefasst und letztendlich an den aufrufenden Programmteil, den Requestgenerator zurückgegeben.

In Abbildung 4.3 wird der Zusammenhang zwischen den aus dem Formular ermittelten Daten und den Elementen des generierten W4F-Wrappers dargestellt.

4.3 Erzeugung der Anfragen

Für alle Elemente E_i der Anfrage, die unter *QueryElement* bzw. *QueryElements* angegeben wurden, muss die Menge der Rückgabewerte $R(E_i)$ ermittelt werden. Dabei wird zunächst $QueryElement.E_i$ ausgewertet, wenn dies vorhanden ist. Wenn nicht, muss $R(E_i)$ aus dem Formular ermittelt werden. Das ist im Fall von Auswahllisten, Radio-Knöpfen und Checkboxen problemlos möglich.

Wenn für alle Anfrageelemente die möglichen Parameter ermittelt werden konnten, dann kann der zuvor generierte und in eine Java-Klasse übersetzte W4F-Wrapper mit den Parametern aufgerufen werden. Dabei werden alle möglichen Kombinationen der Rückgabewerte $r_{ij} \in R(E_i)$ aufgerufen. Für n Formularelemente E_i ergibt sich dann

$$\prod_{i=0}^{n} |R(E_i)|$$

als Anzahl der durchzuführenden Anfragen.

Da dieser Wert recht groß werden kann, wird die Angabe *MaxQueries* aus den Metadaten berücksichtigt und wenn diese nicht vorhanden war, dann wird der voreingestellte Wert von maximal 100 Anfragen verwendet.

[5] steht im Metatag `DB.MaxPages`, oder es wird als Standardwert 5 verwendet

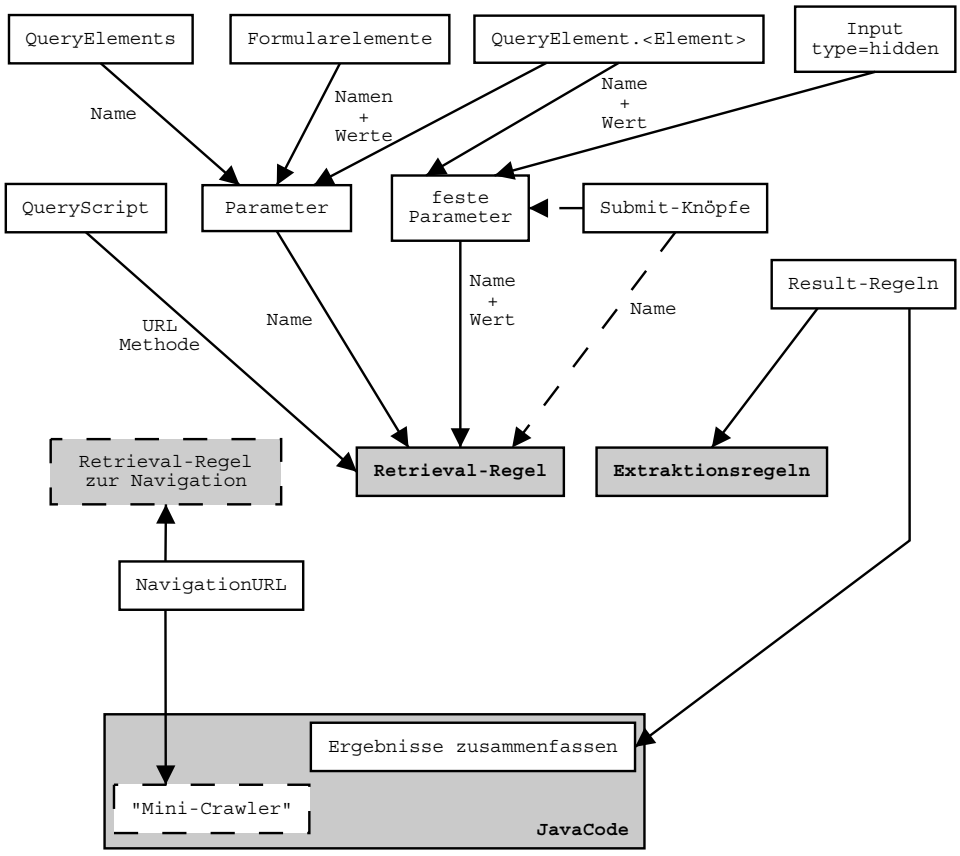

Abbildung 4.3: Generierung des Wrappers aus den Formulardaten

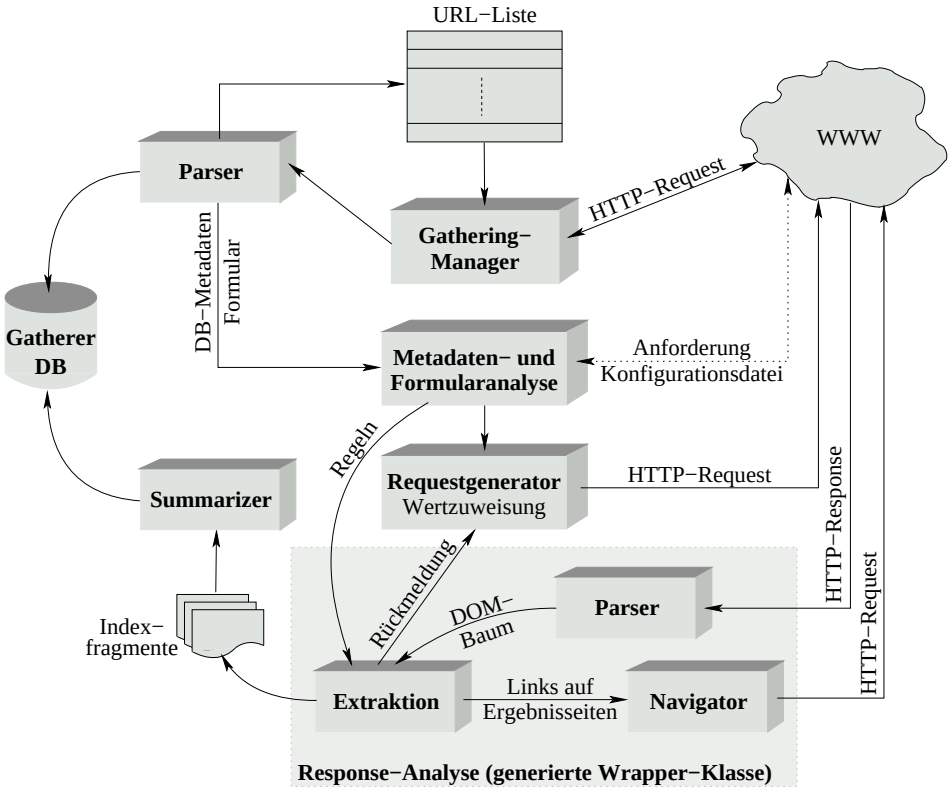

Abbildung 4.4: Aufbau des Gatherers mit Integration von Formularanfragen

Die Anzahl der benötigten Anfragen wird weiter reduziert, in dem die Menge $R(E_i)$ für Checkboxen vereinfacht wird. Statt wie in der Formel 2.1 auf Seite 12 eine Menge von Teilmengen von V_i zu verwenden, definieren wir $R(E_i)$ einfach als Menge von Tupeln (E_i, v_{ij}) mit $v_{ij} \in V_i \cup \emptyset$. Dies ist sinnvoll, da zur Ermittlung der kompletten Ergebnismengen der zu ermittelnden Attribute nicht alle möglichen Kombinationen von Rückgabewerten einer Checkbox-Gruppe nötig sind. Letztendlich werden die Ergebnismengen der Einzelabfragen schließlich zu einer Gesamtmenge vereinigt.

4.4 Integration in Harvest

Die Integration von Datenbanken hinter Suchmasken geschieht über das Hinzufügen der URL der mit DB.*-Metadaten angereicherten Formulardatei zur Liste der vom Crawler zu ladenden Adressen.

Dann muss dem Crawler, in diesem Fall also dem Gatherer von Harvest, noch mitgeteilt werden, wie er das Programm zum Extrahieren der gewünschten Daten über Suchanfragen aufrufen soll. Der Gatherer bietet mehrere Möglichkeiten, um zu einem Dateityp einen Summarizer zuzuordnen, siehe 3.1.2. Da die Formulardatei aber ein normales HTML-Dokument ist, und sich nur durch die Meta-Tags unterscheidet, wird sie von *Essence* als solches erkannt. Mit den in Harvest enthaltenen Typerkennungsmethoden wäre auch nichts anderes möglich, wenn man keine Flexibilität einbüßen will, in dem man beispielsweise ein bestimmtes Namensformat für die Formulardateien vorschreibt.

Damit die Formulardatei erkannt wird, muss diese geparst werden. Das übernimmt der zum Lieferumfang von Harvest gehörende Summarizer `HTML-sum.pl`.

Wenn in der Ausgabe dieses Programms ein Meta-Tag `DB.QueryScript` oder `DB.ConfigFile` gefunden wird, von denen eines unbedingt vorhanden sein muss, dann wird der Request-Generator mit dem Pfad zur Formulardatei als Parameter aufgerufen.

Die Ausgabe des Programms (eine Folge von Attribut-Werte-Paaren im SOIF-Format), wird mit der Ausgabe des HTML-Parsers kombiniert und anschließend in den Index des Gatherers als ein SOIF-Dokument aufgenommen.

Wenn der Requestgenerator für ein Attribut keine Werte ermitteln konnte, dann liefert er ein Attribut-Werte-Paar (siehe Anhang A) zurück, bei dem VALUE die Länge 0 hat. Es wird beim Parsen durch Essence vor dem Eintragen in den Index des Gatherers herausgefiltert.

4.4.1 Auffrischen der Daten

Die Auffrischungsmechanismen von *Essence* sind für statische Objekte konzipiert und können für dynamische Seiten nicht angewendet werden. Ein mit Metadaten angereichertes Formular ist normalerweise ein statisches Dokument, das sich nicht oft ändert. Veränderungen des Datenbankinhaltes oder der optionalen Konfigurationsdatei kann *Essence* also nicht erkennen.

Um die Aktualität auch der SOIF-Dokumente, die über automatische Anfragen erzeugt wurden zu gewährleisten, muss eine geeignete Einstellung des `Time-to-Live`-Wertes erfolgen. Dies ist über eine Post-Summarizing-Regel möglich, siehe Abschnitt 3.2. auf Seite 17.

Kapitel 5

Evaluierung des Ansatzes

Dieses Kapitel wertet die Implementierung des im vorherigen Kapitel konzipierten Ansatzes aus, es werden die Leistungsfähigkeit bewertet und die erkannten Probleme aufgezeigt.

5.1 Implementierung am Beispiel der Technologiedatenbank

Die Technologiedatenbank Nord-Info[1] bietet sich für Tests an, da sie schon über einen *kooperativen Ansatz* in SWING eingebunden wurde. Es ist dadurch möglich, die durch beide Methoden gewonnenen Daten miteinander zu vergleichen.

Abbildung 5.1 zeigt die Suchmaske und den Teil einer Ergebnisseite der Technologiedatenbank. Die Suchmaske muss analysiert werden, um die wesentlichen Anfrageelemente zu ermitteln, aus der Ergebnisseite müssen die HEL-Regeln abgeleitet werden, die gewünschte Attribute ermitteln können.

5.1.1 Ermitteln der Anfrageelemente

Die Suchmaske enthält eine Gruppe von Radio-Knöpfen mit dem internen Bezeichner `radio`, die Eingabefelder `firma` und `produkt`, die Auswahlliste `tt-bereich` und die Checkbox `mv`.

Die Radio-Knöpfe haben als Semantik die Auswahl der für die Anfrage verwendeten Eingabekomponente, entweder eines der Eingabefelder oder die Auswahlliste. Da die Eingabefelder eine unbestimmte Domäne besitzen, über die in diesem Fall nichts bekannt ist, kommen sie für die Anfrage nicht in Frage. Die bestimmte Domäne der Auswahlliste kann hingegen leicht ausgewertet werden.

Die Checkbox wählt die zu durchsuchende Datenbank aus, da nur eine vorhanden ist, sollte sie immer aktiviert sein.

Die minimale Anfrage erhält man also, in dem der Radio-Knopf für die Auswahlliste und die Checkbox aktiviert wird und alle internen Rückgabewerte der Auswahlliste als veränderliche Parameter verwendet werden.

Tabelle 5.1 zeigt die Auswirkungen, die eine nicht minimale Menge von Anfrageelementen auf die Gesamtzahl der Anfragen und die Zeit, die der Wrapper braucht, haben kann. Die extrem erhöhte Zeitdauer beim dritten Test mit der Technologiedatenbank wird durch unsinnige Anfragen ausgelöst, also die Aktivierung des Firmen- oder Produkt-Radio-Knopfes ohne Ausfüllen das dazugehörigen Eingabefeldes. Diese sind sehr allgemein und erzeugen den kompletten Datenbankinhalt als Ergebnis.

[1] `http://www.nord-info.de/tt-meta/form.html?id=958728726`

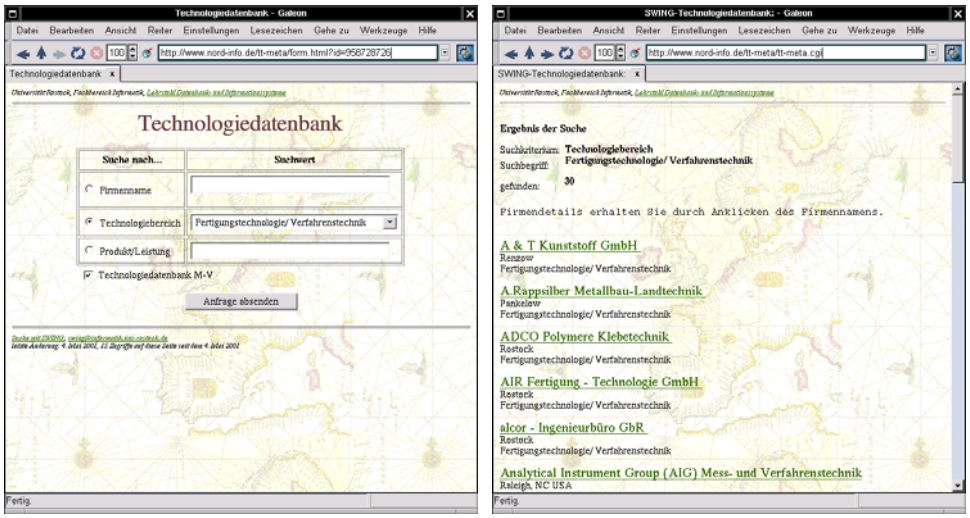

Abbildung 5.1: Suchmaske und Ergebnisseite der Technologiedatenbank

Wesentliche Anfrageelemente	**Anzahl der Anfragen**	**Zeitdauer** in Minuten
Technologiedatenbank		
Technologiebereich	37	2:01
Technologiebereich, Checkbox	74	2:33
Technologiebereich, Checkbox, Suche nach...	222	57:37
TMV		
Region	14	1:28
Kategorie	24	1:57
Region, Kategorie	336	12:24
Cityworld		
Region	12	1:11

Tabelle 5.1: Einfluss der Auswahl der wesentlichen Anfrageelemente

Name	Wert
DB.QueryScript	`http://www.nord-info.de/tt-meta/tt-meta.cgi ,POST`
DB.QueryElement	`tt-bereich`
DB.Result	`html->font[*].a`
DB.Result.firma	`.txt`
DB.Result.ort	`->pcdata[1].txt`
DB.Result.tbereich	`->pcdata[2].txt,match/([\w \/\-äöüß]*)/`
DB.QueryElement.choice	`tt-bereich`
DB.QueryElement.mv	`on`
DB.Navigation	`NO`

Tabelle 5.2: Metadaten für das Formular der Technologiedatenbank

5.1.2 Ermitteln der Ergebnisattribute

Aus der Ergebnisseite lassen sich leicht Firmenname, Ort und Technologiebereich ermitteln. Dies sind die verwendeten HEL-Regeln:

- **Firma:**
  `html->font[*].a.txt`
- **Ort:**
  `html->font[*].a->pcdata[1].txt`
- **Technologiebereich:**
  `html->font[*].a->pcdata[2].txt,match/([\w \/\-äöüß]*)/`

5.1.3 Verwendete Metadaten

Um die Technologiedatenbank für den Crawler durchsuchbar zu machen, werden die in Tabelle 5.2 dargestellten Metadaten verwendet. Im HTML-Code müssen die Sonderzeichen noch durch ihre Entities ersetzt werden.

5.2 Weitere Beispiele

Um die Fähigkeiten der Implementierung zu demonstrieren, wurden für weitere Beispiele die benötigten Metadaten erstellt und die Anfragen ausgewertet. Die anderen Beispiele sind der Hotelführer Cityworld[2] und der Veranstaltungskalender TMV.

5.2.1 TMV

Die Suchmaske von TMV[3] hat als Formularelemente zwei Auswahllisten für Region und Kategorie und zur Angabe des Datums eine Auswahlliste und als Alternative dazu sechs Eingabefelder.

Wesentliche Formularelemente

Der komplette Datenbestand lässt sich über die Abfrage aller Regionen oder aller Kategorien ermitteln. Da die Auswahlliste der Regionen weniger Werte enthält, sollte diese als wesentliches Anfrageelement verwendet werden, um eine minimale Anzahl von Formularanfragen zu erreichen.

[2] `http://www.cityworld.de/cgi-bin/HOTEL/auswahl.pl?dummy=Start+%21`
[3] `http://www.tmv.de/cgi-bin/suche/suche.pl`

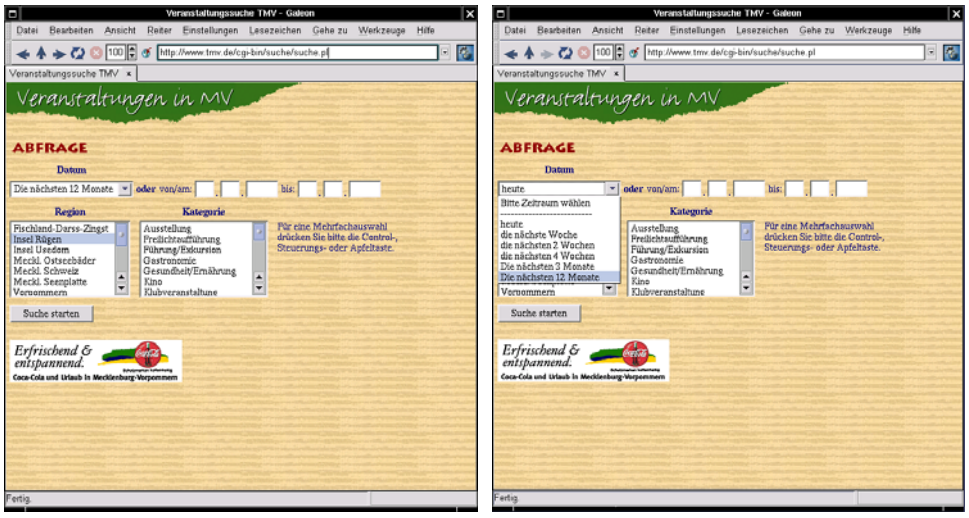

Abbildung 5.2: Suchmaske von TMV

Der naive Ansatz, beide Auswahllisten zu wesentlichen Anfrageelementen zu machen, führt zu einer hohen Zahl von Anfragen, die genauen Werte sind in Tabelle 5.1 aufgelistet. Ein Vergleich der Ergebnismengen ergibt, dass alle drei Anfragetypen zum exakt gleichen Ergebnis kommen, wenn die Anzahl der Anfragen nicht die maximal zulässige übersteigt.

Zeitraum

Das in Abbildung 5.2 gezeigte Formular von TMV bietet zur Suche nach Veranstaltungsterminen die Angabe eines Zeitraumes. Dies dient der Einschränkung der Suchergebnisse. Die Eingabefelder werden wieder ignoriert, da sie eine unbestimmte Domäne besitzen. Die Zeitspannen in der Auswahlliste bilden eine Hierarchie, ein höherer Wert ist immer ein Oberbegriff, der die kleineren mit einschließt. Bei einer Anfrage, die den kompletten Datenbankinhalt ermitteln soll, ist nur die Angabe des größten Wertes, in diesem Fall also *„Die nächsten 12 Monate“* erforderlich.

5.2.2 Verteilung der Werte

In Tabelle 5.3 wird die Verteilung der ermittelten Werte (nach der Entfernung von Duplikaten) bei den minimalen Anfragen der drei getesteten Anbietern dargestellt. Die Ergebnisse bedürfen noch einiger Anmerkungen:

- Eine Einzelanfrage an die Technologiedatenbank hat keine Werte zurückgeliefert, die Suche nach allen Technologiebereichen. Diese Anfrage ist zu allgemein gehalten und wird daher nicht bearbeitet.

- Aus der Anzahl der Werte lässt sich leicht erkennen, nach welchem Attribut gesucht wurde, etwa nach der Region bei Cityworld.

- Nur bei der Technologiedatenbank musste bei der Zusammenfassung der Ergebnismengen der Einzelabfragen eine größere Anzahl an Duplikaten entfernt werden. Das lässt sich aus der Relationen der Attribute untereinander ableiten, eine Firma kann in mehreren Technologiebereichen aktiv sein, während eine Veranstaltung bei TMV nur an einem Ort stattfindet.

Anbieter	Anfragen	Zeit (m:ss)	Attributnamen	Anzahl der Werte min.	max.	Ø	ges.
Technologiedatenbank	37	2:01	Firma	0	197	95.40	543
			Ort	0	62	15.19	129
			Tbereich	0	1	0.97	36
TMV	14	1:28	PLZ	2	28	10.0	107
			Ort	2	33	12.79	154
			Veranstaltung	10	117	59.29	753
Cityworld	12	1:11	Region	1	1	1.0	12
			Name	2	146	83.58	942
			Ort	3	79	29.83	343

Tabelle 5.3: Statistik der Attribute

5.3 Einschränkungen von W4F

Die hier verwendete Version von W4F, `Version 1.03, Build 19 Mar 1999`, wird von den Entwicklern nicht mehr unterstützt. Das bedeutet auch, dass die unter [SA98] auffindbare Dokumentation unvollständig ist. Die nachfolgenden Versionen unterscheiden sich in einigen Funktionen, die neue Dokumentation ist deshalb in vielen Fällen nicht anwendbar.

Hier ist eine Auflistung der in W4F 1.03 nicht vorhandenen Funktionen:

Keine Vererbung

Leider ist es nicht möglich, dass ein Wrapper die Eigenschaften einer Klasse erbt oder ein Interface implementiert. Das verkompliziert die Integration von W4F-generierten Wrappern in eigene Programme. Die vielen Vorteile von objektorientierter Programmierung können nicht ausgenutzt werden. Beispielsweise ist es nicht möglich, dass die Klassen, auf die die Daten abgebildet werden, vom Nutzer implementierte Exceptions auslösen können.

Nicht implementierte NSL-Operatoren

Diese Version von W4F unterstützt nur die Operatoren match() und split(), die anderen Operatoren wie default(), join() und flatten() können nicht verwendet werden. Darum muss ein größerer Teil des Wrappers als Java-Klassen implementiert werden und nicht in Form der regelbasierten HEL.

Eingeschränkte Attribut-Extraktion

Es ist nicht möglich, eine absolute URL aus dem Attribut eines HTML-Tags zu extrahieren, beispielsweise mit `html->a[0]@+href`. Darum muss diese Funktionalität ebenfalls nachgebildet werden.

Kein Vergleich mit `null` möglich

Die Vergleichsoperationen innerhalb von HEL-Regeln lassen keinen Vergleich mit Nullwerten zu, darum müssen bei der Auswertung der durch einen Wrapper ermittelten Daten immer mögliche Nullwerte berücksichtigt werden.

Navigation über Ergebnisseiten

Die aktuelle Version von W4F enthält eine direkte Unterstützung für die Navigation über mehrteilige Seiten, dazu ist nur die Angabe einer Retrieval-Regel, der optio-

nalen maximalen Rekursionstiefe und einer Extraktionsregel, die die Parameter für die Retrieval-Regel ermittelt, nötig.

Diese Funktionalität wurde wie in 4.2.4 beschrieben nachgebildet.

Kein Setzen des Host-Headers

Das Feld *Host* des HTTP-Headers ist im HTTP-Standard 1.1 [RI+97] vorgeschrieben und viele Anfrageprogramme funktionieren nicht, wenn es nicht gesetzt ist. Darum muss es, wie in Abbildung 4.2 auf Seite 28 dargestellt, explizit hinzugefügt werden.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Der hier gewählte Ansatz hat sich zur Lösung einer spezifischen Aufgabe als geeignet erwiesen: die Integration einer bestimmten Menge dynamisch generierter Webseiten in die SWING-Suchmaschine durch Generierung von Suchanfragen an Web-Server, Extraktion der relevanten Schlüsselworte und ihre Integration in den Suchindex von Harvest.

Nicht alle durchsuchbaren Datenbanken mit Formularschnittstelle können mit dieser Methode eingebunden werden, da verschiedene Einschränkungen existieren:

- Die **Datenmenge** kann bei Suchanfragen sehr groß werden, die Anzahl der Anfragen und die durch die Zugriffe entstehende Netzlast stellen eine praktische Einschränkung dar.

- **Komplexe Formulare** die über mehrere Seiten gehen, also Anfragen, deren Ergebnis wieder eine Formular erzeugen, können nicht abgefragt werden. Für diese Zwecke eignen sich spezielle Lösungen, wie die in [Goh00] beschriebene.

- **Eingabefelder** mit unbestimmter Domäne stellen das größte Problem dar, die einzusetzenden Werte können nicht automatisch bestimmt werden, sondern müssen von Hand in die Metadaten des Formulars eingetragen werden. Somit ergibt sich eine praktische Einschränkung.

Wenn jedoch ein Formular den Kriterien in dieser Arbeit entspricht, ist es recht einfach, die Metadaten zu formulieren und das Formular damit zu beschreiben. Das Stellen der Anfragen und das Zusammenfassen der Suchergebnisse geschieht effizient und die Integration in Harvest und damit SWING ist nahtlos.

6.2 Ausblick

Es existieren im derzeitigen Stand der Implementierung einige Einschränkungen, die durch geeignete Erweiterungen aufgehoben werden könnten. Durch die enge Einbindung in den Harvest-Gatherer kann das volle Potential von W4F und der HEL-Extraktionsregeln nicht ausgenutzt werden, da das System SOIF-Daten erzeugen muss, die der Gatherer verarbeiten kann. Dieses Format besteht aus Attribut-Wert-Listen, in dem nach einem Schlüsselwort gesucht werden kann.

Die Extraktionsschicht von W4F ermöglicht die Ermittlung strukturierter Daten aus den Anfrageergebnissen. Wenn die Daten auf geeignete Weise weiterverarbeitet und gespeichert werden, z. B. in einem Datenbanksystem, dann ergeben sich neue Anwendungsmöglichkeiten. Die Firmennamen der Technologiedatenbank könnten zusammen den zugehörigen URLs gespeichert werden, als Suchergebnis müsste nicht mehr auf das Formular verwiesen werden, in dessen Ergebnismenge der Firmenname gefunden wurde, sondern es könnte gleich ein Link auf die Seite der Firma zurückgeliefert werden.

Man sollte auch die Fähigkeit von W4F, die Ergebnisse als XML auszugeben, nicht vergessen. Dadurch könnten sich viele weitere Integrationsmöglichkeiten ergeben.

Das Formulieren der Extraktionsregeln erfordert eine Einarbeitung in W4F. Das Progamm enthält einen „Extraction Wizard[1]“, der beim Ermitteln der Pfade zu den Elementen einer Ergebnisseite vereinfacht. Das Problem ist allerdings, dass damit nur die eindeutigen, vollständig qualifizierten Pfadausdrücke, zusammengesetzt über Punkt-Operatoren, erzeugt werden. In der Praxis werden robuste Extraktionsregeln verwendet, die den Pfeil-Operator sowie evtl. Indexbereiche und *Where*-Bedingungen enthalten. Diese müssen manuell formuliert werden. Wünschenswert wäre eine Möglichkeit, dem Nutzer bei dieser Aufgabe zu unterstützen.

[1] dokumentiert in [Sah99]

Literaturverzeichnis

[Apa] Apache Today. *Apache Tutorial: Introduction to Server Side Includes.* http://www.apachetoday.com.

[ECM99] ECMA (European Computer Manufacturers Association). *ECMAScript Language Specification*, third edition, December 1999. http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM.

[Goh00] Rolf Gohla. Integrierte WWW-Anfragesichten. Diplomarbeit, Universität Rostock, Fachbereich Informatik, February 2000.

[HSWL01] Darren R. Hardy, Michael F. Schwartz, Duane Wessels, and Kang-Jin Lee. *Harvest User's Manual*, May 2001. http://www.arco.de/~kj/harvest/manual/.

[K+] Martijn Koster et al. *The Web Robots FAQ.* http://www.robotstxt.org/wc/faq.html.

[K+93] Martijn Koster et al. *Guidelines for Robot Writers*, 1993. http://www.robotstxt.org/wc/guidelines.html.

[PHP] The PHP Group. *PHP Hypertext Processor.* http://www.php.net.

[RGM01] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. Technical Report 2001-19, Computer Science Department, Stanford University, 2001.

[RI+97] R.Fielding, UC Irvine, et al. Hypertext Transfer Protocol -- HTTP/1.1. Request For Comments 2068, Network Working Group, January 1997.

[Riv92] R. Rivest. The MD5 Message-Digest Algorithm. Request For Comments 1321, Network Working Group, April 1992.

[SA98] Arnaud Sahuguet and Fabien Azavant. WysiWyg Web Wrapper Factory (W4F). Technical report, University of Pennsylvania and Telecom Paris (E.N.S.T.), 1998. http://db.cis.upenn.edu/DL/www8.pdf.

[Sah99] Arnaud Sahuguet. *W4F's documentation*, 1999. http://db.cis.upenn.edu/W4F/doc.html.

[Tit98] Patrick Titzler. Realisierungsvorschlag für die Implementierung einer verteilten Suchmaschine im WWW. Studienarbeit, Universität Rostock, Fachbereich Informatik, 1998.

[WB+98] Lauren Wood, Steve Byrne, et al. The Document Object Model (DOM) Level 1 Specification. W3C recommendation, W3C, 1998. http://www.w3c.org/TR/REC-DOM-Level-1.

[WB+99] Lauren Wood, Steve Byrne, et al. HTML 4.01 Specification. W3C recommendation, W3C, 1999. http://www.w3c.org/TR/html401.

[Web01] Gunnar Weber. Integration von Datenbanken in Suchmaschinen bei unterschiedlichen Kooperationsgraden. Technical report, Lehrstuhl Datenbank- und Informationssysteme Fachbereich Informatik, Universität Rostock, 2001.

Anhang A

Das SOIF

Harvest benutzt das *Summary Object Interchange Format* zum Datenaustausch zwischen Gatherer und Broker. Die Ausgabe der Summarizer wird in diesem Format im Gatherer zwischengespeichert.

In der Tabelle A.1 sind die Terminalsymbole fett gedruckt, <tab> steht für das Tabulatorzeichen im ASCII-Code.

SOIF	::=	OBJECT SOIF | OBJECT
OBJECT	::=	**@** TEMPLATE-TYPE
		{ URL ATTRIBUTE-LIST **}**
ATTRIBUTE-LIST	::=	ATTRIBUTE ATTRIBUTE-LIST
	|	ATTRIBUTE
ATTRIBUTE	::=	IDENTIFIER **{**VALUE-SIZE**}**
		DELIMITER VALUE
TEMPLATE-TYPE	::=	Alpha-Numeric-String
IDENTIFIER	::=	Alpha-Numeric-String
VALUE	::=	Arbitrary-Data
VALUE-SIZE	::=	Number
DELIMITER	::=	**:<tab>**

Tabelle A.1: Formale Beschreibung der Grammatik

In der Praxis, also der Implementierung in Harvest, werden die einzelnen Attribute noch durch ein Newline-Zeichen getrennt. Abbildung A.1 zeigt ein Praxisbeispiel für ein SOIF-Fragment, zu lange Zeilen wurden umgebrochen und mit einem + am Zeilenanfang fortgesetzt, Auslassungen durch ... ersetzt.

```
@FILE { http://www.nord-info.de/tt-meta/form.html
Update-Time{9}: 997969786
Description{20}: Technologiedatenbank
Last-Modification-Time{9}: 997969670
Time-to-Live{6}: 604800
Refresh-Rate{6}: 604800
Gatherer-Name{14}: Metadaten Test
Gatherer-Host{10}: amorgos.informatik.uni-rostock.de
Gatherer-Version{6}: 1.7.15
Type{6}: DBMeta
File-Size{4}: 4562
MD5{32}: 79bf0460e2d9358d828cf31bc0f231f7
firma{8033}: HANSEATISCHE WEITERBILDUNGSGEMEINSCHAFT ROSTOCK GmbH
+(HWR GmbH)
S & N Datentechnik GmbH Schulungszentrum und Systemhaus
...
ort{410}: Raleigh, NC USA
Hohen Luckow
...
tbereich{53}: Chemie/ Biochemie
Bildung
Biotechnologie/ Biomedizin
title{20}: Technologiedatenbank
keywords{107}: datenbank
fachbereich
...
full-text{1439}: Technologiedatenbank
Universität Rostock, Fachbereich Informatik,
...
url-references{122}: /
technologie-db.gif
...
url-texts{114}: Lehrstuhl Datenbank- und Informationssysteme
Technologie-Datenbank
...
db.queryscript{48}: http://www.nord-info.de/tt-meta/tt-meta.cgi,
+POST
db.queryelement{10}: tt-bereich
db.result{15}: html->font[*].a
db.result.firma{4}: .txt
db.result.ort{15}: ->pcdata[1].txt
db.result.tbereich{39}: ->pcdata[2].txt,match/([\w \/\-äöüß]*)/
db.queryelement.choice{10}: tt-bereich
db.queryelement.mv{2}: on
db.maxqueries{1}: 3
db.navigation{2}: NO
comments{440}:  X-URL: http://www.nord-info.de/tt-meta/form.html?
+id=958728726
...
}
```

Abbildung A.1: Beispiel für ein SOIF-Dokument (gekürzt)

Anhang B

HTML Extraction Language

extraction_rules	::=	**EXTRACTION_RULES** **{** (extraction_rule)* **}**
extraction_rule	::=	variable = path_expression (**WHERE** condition (**AND** condition)*)*
path_expression	::=	**html** (navigation_operator label (**[** index_range **]**)?)* attribute (**,** string_processing)*
navigation_operator	::=	**.** | –>
		| **(** path_expression (# path_expression)* **)**
index_range	::=	num_index_range
		| conditional_index_range
num_index_range	::=	integer (**,** num_index_range)?
		| integer – integer (**,** num_index_range)?
		| integer –
		| *****
conditional_index_range	::=	variable (**:** num_index_range)?
attribute	::=	**.txt**
		| **.src**
		| **.getAttr(** attribute_name **)**
		| **.getNumberOf(** tag **)**
string_processing	::=	**match /**regular_expression **/**
		| **split/**regular_expression **/**
condition	::=	path_expression comparison_operator string (**,** **!**)?
comparison_operator	::=	=
		| **!** =
		| = ~
		| **!** = ~

Tabelle B.1: BNF der Grammatik von W4F-Extraktionsregeln

Die Grammatik der Extraktionsregeln bedarf noch einiger Erläuterungen:

Die Regel zum Extrahieren von Attributen, `attribute`, liefert als Wert eine Eigenschaft des Teilbaumes, an dessen Wurzel sie angehängt ist.

- **.txt** ist die Aneinanderhängung aller PCDATA-Knoten des Teilbaumes
- **.src** ist der HTML-Quellcode des Teilbaumes

- **.getAttr(** *attribute_name* **)** ist der Wert des HTML-Attributs *attribute_name*, wenn der Knoten, an dem die Regel hängt, dieses Attribut besitzt. Sonst wird der Wert `null` zurückgegeben. Ein Sonderfall liegt bei der Anwendung von `.getAttr()` an der Wurzel `html` vor, damit lassen sich die Felder des HTTP-Headers ermitteln.
- **.getNumberOf(** *tag* **)** liefert die Anzahl der Kinder des Teilbaumes vom Typ *tag*. Als Spezialfall ist `.getNumberOf(all)` die Gesamtzahl der Kinder des Knotens.

Die vollständige Beschreibung der HTML Extraction Language und die Grammatik der anderen Schichten von W4F findet man in [Sah99].

Anhang C

Installation der Software

Dieser Anhang beschreibt das Einrichten der hier verwendeten Softwarekomponenten.

C.1 Voraussetzungen

Der Metadaten-Crawler wird als Erweiterung des HTML-Summarizers des Harvest-Gatherers installiert. Benötigt werden folgende Programme:

- **Harvest**, es wurde Version 1.7.15 verwendet.
- **Java 2 SDK**, ältere Versionen funktionieren wegen einiger verwendeter Klassen nicht, getestet wurden Sun JDK 1.3.1 und 1.2.
- **W4F** in der Version 1.03 wurde zur Implementierung verwendet.
- Das Paket **com.stevesoft.pat**[1] in der Version 1.32 wird von W4F benötigt.
- Der modifizierte HTML-Summarizer verwendet **mktemp** 1.5-9mdk zum Anlegen temporärer Dateien.
- Der verwendete Summarizer HTML-sum.pl setzt das Perl-Modul **HTML-Parser** voraus.

C.2 Installation

1. Die Variable `$CLASSPATH` des Harvest-Nutzers muss um die Jar-Archive von W4F und com.stevesoft.pat erweitert werden, also um `w4f.jar` und `patbin132.jar`.
2. Das Programm **mktemp** wird in ein Verzeichnis im Suchpfad des Harvest-Nutzers installiert.
3. Die Java-Bytecode-Dateien des Metadaten-Crawlers werden in ein Verzeichnis `lib/gatherer/meta-lib` unterhalb des Verzeichnisses `$HARVEST_HOME` kopiert.
4. Die Datei `HTML.sum` wird in das Verzeichnis `lib/gatherer` unterhalb von `$HARVEST_HOME` kopiert und überschreibt somit den Standard-Summarizer für HTML-Dateien. Eventuell muss der Pfad der Bash-Shell angepasst werden.

[1] als Shareware erhältlich von der Webseite `http://javaregex.com`

5. Optional wird der Gatherer um die Datei `myrules` als Post-Summarizing-Datei erweitert. Dazu wird sie in das Verzeichnis `lib` des Gatherers kopiert und diese Direktive in die Konfigurationsdatei des Gatherers eingefügt:

   ```
   Post-Summarizing: lib/myrules
   ```

   Der Wert für `Time-to-live` kann in der Datei `myrules` noch angepasst werden.