Diplomarbeit

Dynamische Validierung von Zugriffen auf externe Informationsquellen in einer Mediatorumgebung

Zoltán Ádám Mann

Prof. Dr. J. Calmet
Institut für Algorithmen und Kognitive Systeme (IAKS)
Fakultät für Informatik
Universität Karlsruhe

22. Mai 2001

Danksagung

Zu danken habe ich: Peter Kullmann und Prof. Dr. Calmet für die Betreuung bzw. für die hervorragende Arbeitsbedingungen; Prof. Dr. Arató, Prof. Dr. Goos und dem Ministerium für Wissenschaft, Forschung und Kunst des Bundeslandes Baden-Württemberg dafür, dass sie den Studentenaustausch zwischen Budapest und Karlsruhe ermöglichen und organisieren; Dr. Kushmerick von Department of Computer Science, University College Dublin für die riesige Menge von Daten, die er mir zukommen ließ; Anna Flórencz für ihre Hilfe in medizinischen Fragen und für die sonstige Unterstützung; András Orbán für viele Diskussionen; und meinen Eltern für viele Sachen, die die Diplomarbeit nicht direkt betreffen.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit ohne unzulässige Hilfe erstellt und alle verwendeten Literaturquellen angegeben habe.

Karlsruhe, den 22. Mai 2001

Kurzfassung

Mediatorsysteme dienen zur Integration heterogener Informationsquellen. Die autonome, externe Informationsquellen werden durch sogenannte Wrapper eingebunden. Die Aufgabe der Wrapper ist, die Anfragen aus der Sprache des Mediators in die jeweilige Anfragesprache der Informationsquelle, und die Antworten der Informationsquellen zurück in die Sprache des Mediators zu übersetzen.

Bei dem Zugriff auf die externe Informationsquellen können verschiedene Fehler auftreten, insbesondere bei Veränderungen der Informationsquelle, was bei autonomen Systemen immer wieder passieren kann. In der Diplomarbeit werden diese Fehler untersucht und in mehrerer Hinsicht – nach Fehlerquelle, Symptom und Dauer – klassifiziert. Dabei reicht die Spanne von Fehlermöglichkeiten von relativ unkomplizierten technischen Problemen beim Verbindungsaufbau über syntaktische Fehler bis zu semantischen Fehler, die nur mit zusätzlichem Wissen zu erkennen sind.

Dementsprechend werden in der Diplomarbeit auch viele verschiedene Lösungsansätze präsentiert: Testanfragen, Validierung der Arbeitsweise der Wrapper, syntaktische und semantische Validierung der Ausgabe der Wrapper, Validierung der ganzen Mediatoranwendung, Einbeziehen anderer Informationsquellen. Es wird auch untersucht, wie die einzelnen Validierungsverfahren in die Mediatorarchitektur eingebunden werden können. Um die Methoden bewerten und vergleichen zu können, werden auch die wichtigsten funktionale und nicht-funktionale Qualitätsmerkmale der Validierung diskutiert.

Die Lösungen wurden teilweise auch in die Praxis umgesetzt, und im Kontext von Metasearch, einer Metasuchmaschine, die mit dem Mediatorsystem KOMET erstellt wurde, getestet. Die Arbeit beschreibt auch diese Versuche, aus denen ersichtlich ist, dass die benutzte Methoden eine genauere Validierung ermöglichen, als die in der Fachliteratur bisher beschriebenen Ansätze.

Összefoglalás

A mediator rendszerek lehetővé teszik heterogén információforrások integrálását. Az autonóm információforrások és a mediator komponens közötti konverziót úgynevezett wrapperek végzik. Feladatuk, hogy a mediatortól érkező lekérdezéseket a mediator nyelvéből az információforrás nyelvére, a válaszokat pedig az információforrás nyelvéből a mediator nyelvére fordítják.

Az információforrásokhoz történő hozzáférés során számos hiba léphet fel, különösen az információforrás változásai esetén, amire az információforrások autonomitása miatt gyakran sor is kerülhet. A diplomamunka összegyűjti és több szempontból – a hiba forrása, megjelenési formája és időtartama szerint – csoportosítja a lehetséges hibákat. A hibalehetőségek skálája az egyszerű, technikai jellegű kommunikácós hibáktól a szintaktikai hibákon keresztül a komplex szemantikai hibákig terjed, amelyek detektálásához komoly többlettudás szükséges.

Ennek megfelelően a dologozat számos különböző megoldási lehetőséget mutat be: tesztlekérdezések, a wrapper működésének validálása, a wrapper kimenetének validálása, a teljes

mediator-alkalmazás validálása, további információforrások bevonása. A dolgozat bemutatja, hogy az egyes validációs eljárások hogyan építhetők be a mediator architektúrába, valamint a validáló eljárások értékelhetősége és összevethetősége érdekében a validálás főbb funkcionális és nem-funkcionális minőségi jellemzőit is ismerteti.

A megoldások közül néhányat a gyakorlatban is implementáltam, és a KOMET mediator rendszerben megvalósított meta kereső alkalmazáson teszteltem. A diplomamunka tartalmazza az implementáció és a tesztelés részleteit is. A teszteredmények alapján megállapítható, hogy az alkalmazott módszerek segítségével lényegesen pontosabb validáció valósítható meg, mint amit a szakirodalomban eddig leírtak.

Abstract

A mediator system integrates distributed, heterogeneous information sources. These autonomous sources are accessed through so-called wrappers: components that transalate the queries from the language of the mediator into the query languages of the information sources and the answer from the language of the information source back into the language of the mediator.

During the access to the external information sources, several errors may occur, especially if the information source changes, which can occur frequently because of the autonomity of the sources. This work gives a detailed description and classification – with respect to the source, the symptom and the duration – of the possible errors. Errors can range from relatively simple technical problems, such as connection failures to more complex syntactic and semantic eroors that can be only be detected with the help of much extra knowledge.

Accordingly, several potentials solutions are presented: test queries, validating the work of the wrapper, validating the output of the wrapper, validating the whole mediator system, comprising other information sources. The way these validation methods can be incorporated into the mediator architecture are also discussed. In order to valuate and compare the different validation schemes, the most important functional and non-functional quality criteria of the validation are also presented.

Some of these mechanisms have been implemented and tested on Metasearch, a meta Web search application based on the KOMET Mediator system. Details of the implementation and testing process are presented, as well as the results of the tests, which show that the applied methods yield a better validation scheme than the best ones previously described in the literature.

Inhaltsverzeichnis

Kapitel 1

Einleitung und Problemstellung

1.1 Motivation

Im 20. Jahrhundert wurde eine immense Menge von Daten in elektronischer Form gespeichert. Im Jahre 1995 hat diese Menge die in Büchern geschriebene Datenmenge überholt; es sind bis dann etwa 1 Milliarde Bücher erschienen mit etwa 10^{15} Byte Informationsgehalt [13]. Die Menge der in elektronischer Form verfügbaren Daten verdoppelt sich jährlich [29]. Die meisten Informationen, die man braucht, sind schon irgendwo auf dem Web, unabhängig davon, ob man eine Literaturrecherche für seine Diplomarbeit durchführt, Reiseverbindungen nach Oslo sucht, oder Tipps für Weihnachtsgeschenke braucht.

Für die zukünftige Informationsgesellschaft reicht das aber nicht aus. Die Menge der verfügbaren Informationen ist zwar schon mehr als ausreichend, aber man möchte diese Informationen für immer neue Zwecke, in immer neuen Applikationen, in Verbindung mit anderen Informationen verwenden. Das Problem ist, dass die Informationsquellen für ihre eigene spezifische Zwecke verwirklicht worden sind, dementsprechend auch verschiedene Schnittstellen haben, und in verschiedenen Händen sind. Solange sie nur für ihren ursprünglichen Zweck benutzt werden, gibt es keine Probleme, wenn man aber die schon existierenden Informationsquellen für neue Aufgaben wiederverwenden möchte, dann treten Schwierigkeiten auf.

Das typische Szenario ist das folgende. Man möchte ein neues System entwickeln, das komplexe Fragen beantworten soll. Dazu braucht man oft Informationen aus verschiedenen Domänen. Die dazugehörenden speziellen Informationsquellen neu zu entwickeln wäre ein sehr grosser und überflüssiger Aufwand, also möchte man schon existierende Datenbanken benutzen. Man möchte die existierenden Informationsquellen zu einer neuen, komplexeren Informationsquelle integrieren. Dabei soll, basierend auf den verschiedenen Schnittstellen der benutzten Informationsquellen, eine einheitliche Oberfläche entstehen.

Als Beispiel kann an dieser Stelle die Entwicklung eines Entscheidungsunterstützungssystems[1] betrachtet werden: eine hohe Führungskraft eines multinationalen Unternehmens muss Entscheidungen treffen, ob das Unternehmen gewisse neue Produkte auf den Markt bringen soll oder nicht. Um solche Entscheidungen unterstützen zu können, muss der Zugriff auf eine grosse Menge von Informationsquellen gewährleistet werden: Ergebnisse von

[1] engl. Decision Support System, DSS [30]

Marktuntersuchungen sind z. B. aus elektronischen Dokumenten zu entnehmen, Daten über verfügbare Arbeitskräfte können in der Datenbank der Personalabteilung gefunden werden, Produktionspotenzial kann aus SAP[2] ausgelesen werden, Daten über Transportmöglichkeiten sind im Web aufzutreiben, Informationen über die benötigten Rohstoffe liegen in einem EDI[3] Format vor und für die Berechnung von Steuer, Profit etc. ist eine mathematische Funktionsbibliothek vorhanden. Also braucht man viele autonome Informationsquellen, die alle ihre eingene Schnittstellen haben. Es kann natürlich nicht von der Führungskraft erwartet werden, dass er alle dieser Schnittstellen beherrschen soll. Zudem werden wahrscheinlich auch mehrere Informationsquellen gleichzeitig benötigt, um die Anfragen des Benutzers zu bearbeiten. Zum Beispiel würde man alle oben genannten Informationsquellen abfragen müssen, und ihre Antworten kombinieren, um die einfache – und wohl sehr wichtige – Frage "Und wieviel würde das uns kosten?" beantworten zu können.

Es gibt immer wieder Initiativen, eine gewisse Einheitlichkeit in diese Vielfalt zu bringen. Heutzutage verspricht XML[4], ein universeller Standard für Datenaustausch zu sein, aber es gibt auch Einiges, was dagegenspricht:

- Es liegt bereits eine immense Datenmenge in verschiedenen Formaten vor, die man auch noch benutzen möchte. Es ist unwahrscheinlich, dass diese Daten auch in XML umgewandelt werden. In ähnlicher Weise gibt es Programme, die Informationen in einem gewissen Format liefern. Es ist wiederum unwahrscheinlich, dass diese Programme verändert werden, um die Informationen nun in XML zu liefern.
- Auch bei XML müssen sich der Sender und der Empfänger einer Ontologie und XML-Schema einig sein, was ähnliche Probleme hervorruft. XML ist eben nur ein syntaktischer Ansatz.
- Die Stärke des World Wide Web scheint darin zu liegen, dass es dezentralisiert und unzensiert ist. Ein einheitliches Datenformat würde diesen Prinzipien widersprechen.
- Es gibt kein perfektes Datenformat. (Z. B. wird bei XML oft kritisiert, dass es nicht kompakt genug ist.) Wahrscheinlich wird es immer Menschen und Firmen geben, die gerade ein besseres Format ausgefunden haben, und nur dieses benutzen wollen.
- Es gibt Anwendungen, für die XML gerade nicht so günstig ist.

Diese Argumente sprechen natürlich nicht gegen XML, denn XML ist für viele Zwecke sehr gut geeignet, und wird auch in der Zukunft wohl viele Anwendungen finden, sondern gegen den Mythos eines einheitlichen Datenformates. Es scheint also, dass die Integration heterogener Informationsquellen nach wie vor ein wesentliches Problem bleibt.

Für diese Integration braucht man eine neue, vermittelnde Softwarekomponente. Diese Komponente wird oft als *Mediator* bezeichnet [40, 42]. Die grobe Ansicht der Mediatorarchitektur ist im UML-Diagramm von Abbildung 1.1 dargestellt.

Die Aufgabe und Eigenschaften der einzelnen Komponenten dieser Architektur werden in dem nächsten Abschnitt näher beschrieben.

[2] `http://www.sap.com`

[3] Die in der Arbeit vorkommenden Abkürzungen werden im Anhang erläutert.

[4] `http://www.w3.org/XML`

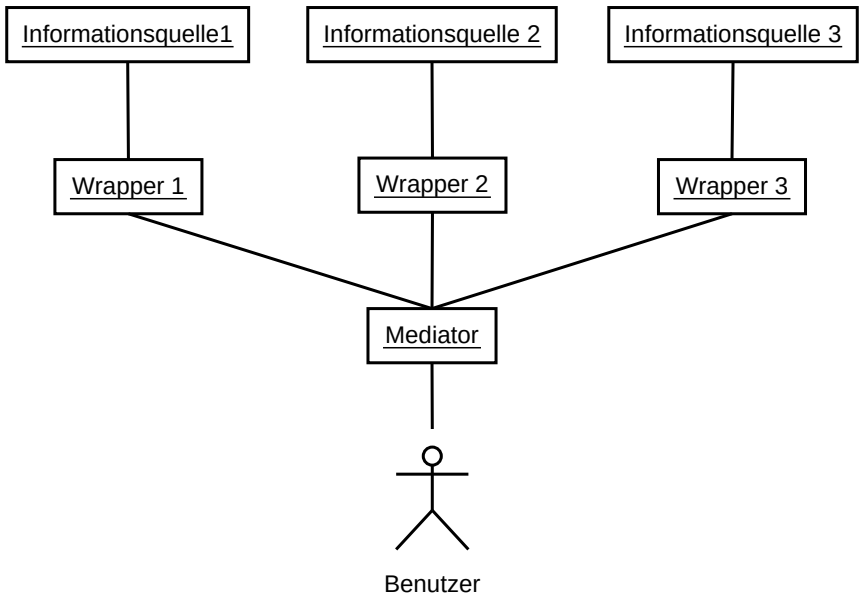

Abbildung 1.1: Die Mediatorarchitektur

1.2 Die Mediatorarchitektur

Die Mediatorarchitektur, wie sie in Abbildung 1.1 dargestellt ist, besteht aus Informationsquellen, Wrapper (auch Übersetzer oder Translator genannt), dem Mediator und der Benutzerschnittstelle. Wie diese Komponenten zur Gesamtarchitektur zusammengestellt werden, hängt von vielen Faktoren ab, wie z. B. die konkrete Aufgabe des Mediators, oder welche Informationsquellen vorhanden sind. Zwei Extremfälle sind die Top-down und die Bottom-up entwicklung. Bei der ersten wird aus der jeweiligen Aufgabe des Mediators ausgegangen, und man sucht passende Informationsquellen, bzw. man passt die vorhandenen Informationsquellen an die Aufgabe an. Bei der zweiten hingegen geht man aus den vorhandenen Informationsquellen aus, und versucht, sie zu einer mächtigeren Wissensquelle zu integrieren [17, 37].

Dabei wurde bewusst das Wort Wissen an Stelle von Information benutzt. Mediatoren können als jene Schicht der Informationsverarbeitung angesehen werden, in der aus den Informationen nützliches Wissen synthetisiert wird [41].

In den folgenden Abschnitten werden die einzelnen Komponenten der Mediatorarchitektur näher erläutert. Ein konkretes Mediatorsystem ist im Abschnitt 1.3.1 beschrieben.

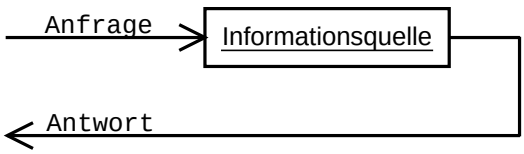

Abbildung 1.2: Das Modell der Informationsquelle

1.2.1 Informationsquelle

Der Mediator integriert verschiedene Informationsquellen. Die wichtigsten Merkmale der Informationsquellen sind ihre Heterogenität und Autonomität.

Heterogenität bedeutet, dass sie durchaus verschieden sein können, sowohl in ihrem Zweck, als auch in ihrem Aufbau, ihrer Benutzerschnittstelle, ihrer Qualität, ihrer Zuverlässigkeit, ihrer Grösse etc. Es wird lediglich angenommen, dass man an die Informationsquellen Anfragen stellen kann, worauf sie mit der entsprechenden Information antworten (siehe Abbildung 1.2). Der eigentliche (technische) Mechanismus, wie die Anfragen gestellt und die Antworten zurückgegeben werden, kann auch stark variieren. Zum Beispiel bei einer über das Web zugänglichen Datenbank werden die Anfragen entweder in das URL eingebettet oder mit HTTP POST an ein CGI-Script geschickt. Die Antwort wird dann in Form einer dynamisch erstellten HTML-Seite zurükgeliefert. (Für weitere Informationen über HTTP und CGI, siehe `http://www.w3.org/Protocols` bzw. `http://hoohoo.ncsa.uiuc.edu/cgi`.)

In dieser Hinsicht kann die Informationsquelle so betrachtet werden, dass sie eine Funktion I verwirklicht: wenn die Anfrage mit Q und die Antwort mit R bezeichnet wird (aus eng. Query und Response), so gilt: $R = I(Q)$. Für manche Mediatoranwendungen ist es aber möglicherweise günstiger, die Informationsquelle in Form von Relationen oder Prädikate zu erfassen (siehe Abschnitt 1.3.1). Die benötigten Umwandlungen werden dann vom jeweiligen Wrapper durchgeführt.

Autonomität bedeutet, dass der Autor bzw. der Benutzer des Mediators keinen unmittelbaren Einfluss auf die Informationsquelle besitzt; man muss sie so nehmen, wie sie ist. Das ist der Preis dafür, dass man schon existierende Informationsquellen benutzt, und die Quelle nicht selber aufbauen und warten muss. Aber das kann zu vielen Problemen führen, z. B. kann die Quelle ohne Weiteres Veränderungen unterworfen sein, oder sogar (temporär oder endgültig) aufhören zu existieren. Manche Aspekte dieser Problematik werden in dieser Arbeit durch die Entwicklung von Validierungsmethoden behandelt.

Informationsquellen können lokal, auf jenem Rechner sein, auf dem der Mediator abläuft, sie können aber auch auf anderen Rechner sein, so dass sie nur über das Nezt erreichbar sind. Typische Informationsquellen sind zum Beispiel relationale und objekt-orientierte Datenbanken, Suchmaschinen (im Internet oder Intranet), Informationssysteme aller Art (z. B. Bibliotheksverzeichnis), Computer Algebra Systeme, spezielle Softwarepakete, Expertensysteme, Text-, HTML-, oder XML-Dateien, oder sogar der Benutzer.

1.2.2 Wrapper

Damit der Mediator die heterogenen Informationsquellen einheitlich behandeln kann, müssen sie an eine einheitliche Darstellung adaptiert werden. Diese Adaptierung kann im Allgemeinen nicht durch die entsprechende Änderung der Informationsquelle erreicht werden, da die Informationsquellen autonom sind. Die Adaptierung könnte theoretisch im Mediator selbst erfolgen. Warum dafür doch meistens separate Komponenten (Wrapper[5])

[5]Das englische Wort Wrapper bedeutet buchstäblich *Hülle*. Der Wrapper erweitert die Funktionalität der Informationsquelle; sie wird praktisch eingehüllt, so dass sie dem Mediator gegenüber eine andere Schnittstelle zur Verfügung stellt. Siehe auch Erläuterungen zu den verwendeten Fremdwörtern im Anhang.

benutzt werden, hat hauptsächlich softwaretechnische Gründe, und zwar erleichtert diese Dekomposition die Optimierung, die Wartung und die Wiederverwendbarkeit [17].

Aus einem funktionalen Sichtpunkt ist die Aufgabe des Wrappers die Übersetzung. Er übersetzt die Anfragen, die der Mediator in einer einheitlichen Sprache formuliert, in die Anfragesprache der jeweiligen Informationsquelle. In ähnlicher Weise übersetzt er die Antworten der Informationsquelle aus ihrer Sprache in die des Mediators. Dadurch wird erreicht, dass sich nur der Wrapper mit der eigentlichen Sprache der Informationsquelle auseinandersetzen muss; diese Details sind dem Mediator verborgen. Ihm scheint, als wären alle Informationsquellen in derselben Sprache anzusprechen.

Andererseits kann die Aufgabe des Wrappers auch so angesehen werden, dass er das Modell bzw. Schema der Informationsquelle in das sog. föderative Modell bzw. Schema des Mediators integriert. Der Wrapper dient als Modellübersetzer, indem er das lokale Modell der Informationsquelle, in dem ihre Daten vorliegen, in ein einheitliches Datenmodell (engl. Common Data Model) übersetzt. Das lokale Schema der Informationsquelle ist im lokalen Modell definiert vorgegeben. Es wird mittels des einheitlichen Datenmodells neu formuliert: das wird vom Modellübersetzer geleistet. Das so entstandene sog. Exportschema, das zum lokalen Schema äquivalent ist, muss innerhalb des einheitlichen Datenmodells in das föderative Schema übersetzt werden: das macht der Schemaübersetzer [17, 42].

Existierende Forschungsergebnisse über Wrapper fokussieren meistens auf die Anfrageübersetzung, und behandeln die Übersetzung der Antworten als einfache Datenkonvertierung. (Diese Arbeiten benutzen auch meistens die Bezeichnung Übersetzer oder Translator an Stelle von Wrapper.) Das ist nur solange korrekt, bis sich die Ausgabe der Informationsquelle nicht ändert. Da aber die Informationsquellen autonom sind, kann das nicht ausgeschlossen werden. In der Tat, besonders Internet-basierte Informationsquellen, die heutzutage immer wichtiger sind, wie z. B. Suchmaschinen, neigen dazu, ihr Layout häufig zu ändern, was die menschlichen Benutzer zwar nicht stört, aber die maschinelle Bearbeitung der Ergebnisse sehr problematisch macht. Näheres dazu siehe in den nächsten Kapiteln.

1.2.3 Mediator

Die Aufgabe des Mediators ist es, die Anfragen des Benutzers in kleinere Unteranfragen aufzuspalten, die für die involvierten Informationsquellen geeignet sind, dann die Unteranfragen an die jeweiligen Informationsquellen zu schicken (mit der Vermittlung der Wrapper), die Antworten der Informationsquellen einzusammeln (wieder mit der Vermittlung der Wrapper), und letztendlich die Antworten auf die Unteranfragen zu einer Antwort auf die ursprüngliche Anfrage zu kombinieren.

Die Komplexität der Anfragen, die der Mediator bearbeiten kann, kann sehr stark variieren. Es ist z. B. möglich, dass er komplexe Verknüpfungen zwischen den Ergebnissen der einzelnen Informationsquellen erstellen kann; aber Verknüpfungen innerhalb einer Domäne wird er wohl der zuständigen Informationsquelle überlassen, falls sie diese Funktionalität unterstützt. (Für ein Beispiel, siehe Abschnitt 1.3.1.)

Es ist auch durchaus möglich, dass die Antworten der Informationsquellen einander widersprechen; der Mediator muss auch das bewältigen können. Zu diesem Zweck können

verschiedene Konfliktauflösungsstrategien definiert werden, z. B. kann der Mediator manche Quellen als zuverlässiger betrachten, als die anderen, oder bei numerischen Werten kann der Durchschnitt gebildet werden usw. Es ist aber auch möglich, dass schon eine einzige Informationsquelle widersprüchliche Informationen zurückliefert.

Für weitere Informationen über Mediatoren, siehe Abschnitt 1.3.1 sowie z. B. [5, 16, 36, 42].

1.2.4 Benutzerschnittstelle

Der Mediator kann entweder direkt von einem menschlichen Benutzer oder von einer anderen Softwarekomponente benutzt werden. Auf jeden Fall bietet er eine Schnittstelle, die die Anfragen annimmt und die Antworten zurückliefert. Meistens ist dafür eine spezielle Sprache, eine grafische Oberfläche (im Falle eines menschlichen Benutzers), oder eine Funktionsbibliothek (im Falle der maschinellen Weiterverarbeitung) definiert.

An dieser Stelle muss erwähnt werden, dass die Mediatorarchitektur meistens noch die Existenz einer weiteren Person voraussetzt. Diese Person wird im Folgenden als Mediatorfachmann bezeichnet. Seine Aufgabe ist es, die benötigten Informationsquellen am Mediator anzubinden, dabei eventuell die benötigten Wrapper zu konstruieren, und sie zu warten. (In Wirklichkeit kann es natürlich durchaus sein, dass diese Aufgaben nicht einer einzigen Person zugeteilt werden, sondern eher einem ganzen Team, aber das spielt für unsere Betrachtungen keine Rolle. Für das Thema dieser Arbeit ist die Abstraktion des Mediatorfachmanns völlig angemessen.)

1.3 KOMET und Metasearch

1.3.1 KOMET

Diese Arbeit ist im Rahmen des Projektes KOMET (Karlsruhe Open MEdiator Technology) entstanden. KOMET wird seit 1994 an dem Institut für Algorithmen und Kognitive Systeme der Universität Karlsruhe entwickelt. Es bietet ein Rahmen für die einfache Implementierung von Mediatorsystemen. KOMET verwendet die deklarative, auf annotierter Logik basierende Sprache KAMEL (KArlsruhe MEdiator Language) [5, 15].

Ein Mediatorprogramm besteht aus Klauseln, die die folgende Form haben [4]:

$$A : \nu \leftarrow C_1 \& \ldots \& C_n \| B_1 : \mu_1 \& \ldots \& B_k : \mu_k$$

Der Kopf (head) $A : \nu$ ist ein annotierter Atom, die $B_i : \mu_i$ sind annotierte Literale. Die C_i sind sog. Constraint-Relationen, die in externen Informationsquellen definiert sind. Diese repräsentieren also die Schnittstelle zu den eigentlichen externen Informationsquellen, die aus unserer Sicht besonders wichtig sind. Da im Constraint-Teil keine Annotationen vorkommen, werden sie hier auch nicht weiter behandelt.

KOMET nimmt also an, dass die Informationsquellen Relationen zur Verfügung stellen. Diese Relationen werden als Prädikate in der Inferenzmaschine von KOMET verwendet.

Für eine tiefere Beschreibung, siehe z. B. [5]. An dieser Stelle wird stattdessen nur ein Beispiel betrachtet [4]:

$$
\begin{aligned}
&stock(Name, Close) : [\{Date\}] \leftarrow \\
&\quad DB :: \{stockname(Name, SID) \& close(SID, Date, Close)\} \| \\
&stock(Name, Close) : [\{today\}] \leftarrow \\
&\quad T :: stock(Name, Close) \| \\
&equal\_closing(Name1, Name2) : [\{Date\}] \leftarrow \\
&\quad \| stock(Name1, Close) : [\{Date\}] \& stock(Name2, Close) : [\{Date\}] \& \\
&\quad Name1 \neq Name2
\end{aligned}
$$

In der ersten Klausel wird eine Relation *stock* basierend auf zwei Relationen (*stockname* und *close*) der Datenbank DB definiert. Die gewonnenen Daten sind mit dem Zeitpunkt annotiert. Die zweite Klausel extrahiert dieselbe Information für den aktuellen Zeitpunkt mit Hilfe eines Ticker T. (*today* ist eine Konstante.) Die dritte Klausel verwendet die bereits definierte Relation *stock*, um eine neue Relation zu definieren.

KOMET unterstützt konjunktive Anfragen und Negation. Wenn eine konjunktive Anfrage nur einer einzigen Informationsquelle bedarf, die ihrerseits auch konjunktive Anfragen unterstützt, dann wird die ganze Anfrage an die Informationsquelle delegiert. Sonst wird die Konjunktion im Mediator verwirklicht. KOMET unterstützt weiterhin eine einfache und schnelle Implementierung von Wrapper sowie die Wiederverwendung von bereits geschriebenem Code [17].

1.3.2 Metasearch

Metasearch[6] ist eine einfache Anwendung von KOMET, die aber viele Eigenschaften und auch einige Probleme von KOMET gut demonstriert. Es ist ein Mediatorprogramm, das eine Anfrage an verschiedene Internet-Suchmaschinen (wie z. B. AltaVista[7] oder Excite[8]) delegiert. Die Suchmaschinen antworten typischerweise mit einer HTML-Seite, in der die eigentliche Antwort auf die Anfrage (typischerweise URL, Titel und ein kurzes Extrakt von etwa 10 Seiten) eingebettet ist. Diese Informationen werden dann mittels spezieller Wrapper aus der HTML-Seite extrahiert, und die Antworten der einzelnen Quellen werden zusammengefügt. Die Extraktion der Informationen aus der HTML-Seite erfolgt zur Zeit mit Hilfe von regulären Ausdrücken [6].

Wenden wir uns jetzt nur den in Metasearch verwendeten Wrappern zu. Z. B. wird der Wrapper für AltaVista folgenderweise definiert:

```
ALTAVISTA = WWWSEARCH(
   'www.altavista.com',
   'cgi_bin/query?q=%Q',
   '<dl><dt><b>*.</b><a href="%U"><b>%T</b></a><dd>'
)
```

[6] http://calmet-pc.ira.uka.de/komet
[7] http://www.altavista.com
[8] http://www.excite.com

`ALTAVISTA` ist eine Spezialisierung des generischen Web-Wrappers `WWWSEARCH`. Die Spezialisierung erfolgt dadurch, dass man drei Parameter angibt, und zwar ist in diesem Fall das Basis-URL eben `www.altavista.com`, der Pfad, mit dem man die Anfragen stellen kann, ist `cgi_bin/query?q=%Q`, (wobei `%Q` die eigentliche Anfrage ist), und der reguläre Ausdruck, mit dessen Hilfe die URLs und die Titel aus der resultierenden HTML-Seite extrahiert werden, ist `<dl><dt><b>*.</b><a href="%U"><b>%T</b></a><dd>`. (Dabei wird das URL immer in die Variable `%U`, der Titel in die Variable `%T` extrahiert.)

Der generische Web-Wrapper `WWWSEARCH` ist in C++ implementiert. Er arbeitet in zwei Schritten: zuerst lädt er die Seite herunter, die durch das Basis-URL, den Pfad des Suchscripts und die eigentliche Anfrage bestimmt ist. Im zweiten Schritt extrahiert er die gewünschten Informationen (URLs und Titel) aus der heruntergeladenen Seite mit Hilfe des regulären Ausdrucks.

Das Problem, das sich bei Metasearch wirklich akut zeigt, aber auch bei anderen Mediatoranwendungen zu finden ist, ist dass das Layout der HTML-Seiten, die die Suchmaschinen zurückgeben, oft verändert wird. Das führt in vielen Fällen dazu, dass der eine oder andere Wrapper plötzlich falsche Informationen (oder auch nichts) extrahiert. Um diese Problematik handhaben zu können, bedarf es Validierungsmöglichkeiten. Genau das ist das Thema dieser Arbeit, und wird im nächsten Abschnitt detaillierter beschrieben.

Bemerkung: der obige reguläre Ausdruch für AltaVista war auch nur etwa bis zum 1. März 2001 gültig. Seitdem steht an Stelle von `<b>` schon `<b class=txt2>`, und die `<a...>` und `<b...>` Tags (und in ähnlicher Weise auch `</a>` und `</b>`) sind umgekehrt. Am 12. März 2001 gab es wieder Veränderungen: zum `<a...>` Tag wurde ein `onMouseOver`-Script zugefügt, ausserdem fangen alle URLs seitdem mit `/r?r=` an (siehe Abschnitt 4.3.1).

1.4 Validierung

Bei Metasearch ist der Bedarf nach Validierung deutlich erkennbar, denn bei den gängigen Suchmaschinen kommt eine Remodellierung oft vor [23, 33].

Das ist aber kein spezielles Problem von Metasearch, sondern es ist ein generelles Problem bei Mediatoranwendungen. Die ganze Problematik ist daraus abzuleiten, dass der Mediator bereits existierende, autonome Informationsquellen benutzt. Andererseits ist es eben die Stärke der Mediatoranwendungen, dass man die benötigten Informationsquellen nicht selbst erstellen und warten muss. Da aber somit die einzelnen Informationsquellen in anderen Händen sind, muss man in Kauf nehmen, dass sie eventuell verändert werden, oder sogar aufhören zu existieren. Da die Informationsquellen autonom sind, kann man also weder erwarten, dass sie ihr Format nicht verändern, noch dass sie bei einer Änderung den Mediator benachrichtigen.

Dies ist also ein intrinsisches Problem von Mediatoranwendungen. Theoretisch gibt es zwei Wege, wie man mit diesem Problem umgehen kann:

1. Man konstruiert den Wrapper so robust, dass er alle möglichen Änderungen der Informationsquelle toleriert, so dass also der Mediator die Änderungen gar nicht mitbekommt.

2. Man validiert den Wrapper, d. h. man versucht zu überprüfen, ob der Wrapper richtig funktioniert. Normalerweise wird der Wrapper wohl korrekt funktionieren, also muss gar nichts gemacht werden. Wenn jedoch festgestellt wird, dass der Wrapper nicht mehr korrekt funktioniert (was meistens auf eine Änderung der Informationsquelle schliessen lässt), dann muss der Wrapper angepasst werden. Dies kann entweder vom Mediatorfachmann gemacht werden, kann aber auch automatisch passieren.

Leider ist weder 1. noch 2. allgemein lösbar, denn es gibt unendlich viele Möglichkeiten, wie die Informationsquelle seine Ausgabe kodieren kann, und ausserdem kann sich auch der Inhalt ändern, d. h. es ist nicht garantiert, dass die Informationsquelle die gleiche Anfrage immer mit der gleichen Information beantwortet. Also werden wir uns bemühen, Methoden zu finden, die in der Praxis mit möglichst hoher Sicherheit funktionieren. Die vorliegende Arbeit befasst sich mit der Validierung von Wrappern – aber auch das kann zur Konstruktion von robusten Wrappern beitragen, denn Wrapper und Validator zusammen können als ein robuster Wrapper angesehen werden.

In Kapitel 2 werden zunächst die möglichen Fehlerursachen untersucht und klassifiziert, die bei der Validierung in Betracht genommen werden müssen. In Kapitel 3 werden verschiedene Lösungsansätze vorgestellt, sowie einige Möglichkeiten, den Validator in die Mediatorarchitektur einzubinden. Kapitel 4 dient als Fallstudie: darin werden die vorhin beschriebenen Methoden in der Praxis, anhand des Metasearch-Beispiels, dargestellt. Am Ende der Arbeit fasst Kapitel 5 die Ergebnisse zusammen, und vergleicht sie mit Ansätzen aus der Fachliteratur.

Kapitel 2

Klassifizierung der möglichen Fehler

Wie schon früher angedeutet, können bei dem Zugriff auf externe Informationsquellen (der meistens mit der Vermittlung eines Wrappers durchgeführt wird) verschiedene Fehler auftreten. Um diese Fehler erfolgreich beseitigen zu können (Kapitel 3 beschreibt mögliche Lösungsstrategien), muss zuerst geklärt werden, mit welchen Arten von Fehlern man überhaupt rechnen muss.

In diesem Kapitel werden die verschiedenen Fehler in folgenden Hinsichten klassifiziert:

- nach Fehlerquellen, d. h. an welcher Stelle der Fehler entsteht;
- nach Symptomen, also welche sichtbare Folgen der Fehler hat;
- nach der Dauer des Fehlers bzw. der Veränderung, die den Fehler verursacht.

In den folgenden Abschnitten werden die möglichen Fehlerquellen (Abschnitt 2.1) und Symptome (Abschnitt 2.2) aufgelistet, jeweils mit einer allgemeinen Beschreibung und an einem konkreten Beispiel, nämlich an Metasearch (siehe 1.3.2), demonstriert. Es wird auch schon angedeutet, wie schwer oder leicht die einzelnen Fehler zu entdecken sind. (Näheres dazu siehe in Kapitel 3.)

2.1 Klassifizierung der Fehlerquellen

Wie man in Abbildung 2.1 sieht, kann ein Fehler entweder in der Informationsquelle, im Wrapper, oder im zugrundeliegenden Kommunikationskanal auftreten. Es ist aber auch möglich, dass ein fehlerhaftes Verhalten dadurch entsteht, dass sich die Informationsquelle ändert, so dass der Wrapper ihre Ausgabe nicht mehr korrekt bearbeiten kann. Es ist theoretisch auch möglich, dass sich das Kommunikationsmedium ändert, aber das scheint nicht häufig der Fall zu sein. Und es ist noch weniger wahrscheinlich, dass sich der Wrapper ändert, ohne eine entsprechende Änderung der Informationsquelle oder des Kommunikationskanals.

Abbildung 2.1: Kommunikationsmodell

2.1.1 Fehler in der Informationsquelle

Auch das zuverlässigste System funktioniert manchmal fehlerhaft, also kann von den eingebundenen Informationsquellen auch nicht erwartet werden, dass sie fehlerfrei arbeiten. Dass diese Quellen sogar autonome Systeme sind, macht die Lage in mancher Hinsicht noch komplizierter:

- Man bekommt nicht unmittelbar mit, wenn ein Fehler auftritt. Wenn z. B. der entfernte Rechner, in dem die Informationsquelle untergebracht ist, ausgeschaltet wird, wird der Mediator wahrscheinlich nicht informiert, also lässt sich das Problem nur durch gewisse (vielleicht unklare) Symptome feststellen. In diesem Fall würde man z. B. bemerken, dass der Wrapper gar nichts mehr extrahiert, was auch andere Gründe haben könnte. Somit wird die Fehlersuche auch wesentlich komplizierter.
- Man kann meistens nicht genau wissen, wie zuverlässig das benutzte System ist. Dass es nicht zuverlässig genug ist, wird meistens nur dadurch festgestellt, dass es viel zu oft Probleme bereitet. Die Zuverlässigkeit des benutzten Systems kann sich auch mit der Zeit verschlechtern, worauf man wahrscheinlich auch keinen Einfluss hat.
- Man kann den Fehler nicht selbst korrigieren. Es ist zwar vorteilhaft, dass man den Fehler nicht selbst korrigieren *muss*, aber man kann auch nichts machen, wenn z. B. die Informationsquelle für zwei Tage ausfällt, obwohl man sie dringend brauchen würde.

Der Ausfall der Informationsquelle ist ein sehr typisches Beispiel, das leider auch relativ oft vorkommt. Diese Fehler könnten eventuell noch weiter unterteilt werden, z. B. nach dem Grund des Ausfalls (Überlastung, Stromausfall, Hardwarefehler etc.), aber das ist aus unserer Sicht jetzt irrelevant.

Es gibt leider auch subtilere Fehlermöglichkeiten, als das bereits erwähnte Ausfallen der Informationsquelle. Einige Beispiele mögen dies schildern:

- Programmierfehler in einer benutzten Funktionsbibliothek. Dies kann z. B. zu schlechten Ergebnissen führen, oder die Informationsquelle selbst kann bei einer gewissen Anfrage in eine endlose Schleife geraten.
- Die Informationsquelle beinhaltet selbst Widersprüche.
- Die Informationsquelle beinhaltet fehlerhafte, falsche Informationen.

Es ist schon an dieser Stelle zu sehen, dass manche Fehlerarten wohl auf der Ebene der Wrapper gut feststellbar sind (z. B. dass die Informationsquelle keine Anfragen annimmt),

manche aber nur sehr schwierig oder kaum zu entdecken sind, z. B. dass die Informationsquelle falsche Informationen beinhaltet. Diese Art von Problemen wird möglicherweise durch das Mediatorprogramm behandelt, z. B. im Zusammenspiel mit anderen Informationsquellen. Abschnitt 3.3 beschreibt Möglichkeiten, wie die Validierungsfunktionalität in die Mediatorarchitektur – möglicherweise in das Mediatorprogramm selbst – integriert werden kann.

Bei Metasearch ist das genauso: dass eine Suchmaschine gerade nicht erreichbar ist (weil sie z. B. überlastet ist), stellt sich schon bei dem ersten Schritt, d. h. beim Herunterladen der HTML-Seite, heraus. Dazu muss die Seite gar nicht verarbeitet werden, sondern das kann aus dem HTTP-Kopfzeile ausgelesen werden. Dass aber die extrahierten URLs z. B. zu nicht relevanten Seiten zeigen, ist schon viel schwieriger feststellbar. Wenn man im Validierungsprozess auch solche Fehler entdecken will, dann wird zusätzliches Wissen und Intelligenz gebraucht.

2.1.2 Kommunikationsfehler

Wie Abbildung 2.1 illustriert, greift der Wrapper typischerweise nicht direkt auf die Informationsquelle zu, sondern über ein gewisses Kommunikationsmedium. Welche Fehler an dieser Stelle verursacht werden können, hängt natürlich stark von der Natur des Kommunikationskanals ab, die wiederum stark variiert.

Am sichersten ist die Kommunikation, wenn die Informationsquelle lokal, also auf dem selben Rechner ist, wie der Wrapper (und der Mediator). Auch in diesem Fall können aber Fehler auftreten, wie z. B. der Überlauf von Puffern, oder Kapazitätprobleme (z. B. beim Shared-Memory-Zugriff, wenn der Speicher voll ist). Die Ursachen können entweder Programmierfehler sein, oder Überlastung von Ressourcen, oder aber auch schlechtes Betriebssystem-Design.

Probleme werden häufiger, wenn die Informationsquelle nicht lokal, sondern entfernt ist, und nur über Netzzugriff erreicht werden kann. Auch da gibt es Unterschiede je nach Netzwerk. Z. B. hat man offensichtlich niedrigere Fehlerraten bei einem gemieteten Privatnetzwerk mit garantierten QoS[1]-Parametern, als bei einem öffentlichen Netz, wie das Internet. Das Internet ist aber sehr wichtig für uns, erstens wegen seiner grossen globalen Bedeutung, und zweitens wegen der konkreten Zielsetzung, Zugriffe beim Metasearch-Beispiel zu validieren.

Es können einerseits technische Fehler auftreten, wie z. B. das Durchbrechen eines Kabels, aber besonders im sich stets wandelnden Internet bilden die Probleme oberhalb der physikalischen Schicht einen grossen Anteil. Typische Beispiele sind temporäre DNS-Ausfälle, IP-Nummer-Konflikte, Proxy-Probleme und das Umziehen von Web-Seiten.

Aus der Sicht des Wrappers kann oft gar nicht festgestellt werden, ob ein konkreter Fehler von der Informationsquelle oder vom Netz verursacht worden ist. Wenn z. B. ein URL nicht gefunden werden kann, dann wird das meistens wohl daran liegen, dass die Seite gar nicht exisitert, aber es kann auch sein, dass sie zwar existiert, aber wegen Netzproblemen nicht erreicht werden kann.

[1] engl. Quality of Service, Dienstleistungsqualität

2.1.3 Fehler im Wrapper

Wrapper werden entweder von Hand, oder automatisch, typischerweise durch maschinelles Lernen konstruiert [14, 22, 34]. In beiden Fällen kann der Wrapper fehlerhaft sein.

Wenn man den Wrapper von Hand erstellt, muss man mit Programmierfehlern rechnen. An dieser Stelle soll auch erwähnt werden, dass das Testen von Wrappern sehr schwierig ist. I. A. ist der Raum der möglichen Anfragen entweder so groß, dass nicht alle möglichen Anfragen erschöpfend getestet werden können, oder sogar unbeschränkt. Also wird man nur einige Anfragen betrachten, und versuchen, die Erfahrungen zu verallgemeinern. Das heisst, man versucht, die möglichen Arten von Verhalten der Informationsquelle mit einem endlichen (möglichst kleinen) Satz von representativen Testfällen zu erschöpfen. Da aber die Informationsquelle autonom ist, kann nicht hundertprozentig sichergestellt werden, ob die Verallgemeinerung bzw. die Klassifizierung tatsächlich korrekt ist.

Andererseits, wenn der Wrapper automatisch erstellt worden ist, kann man nicht sicher sein, ob der Lernalgorithmus wirklich die relevanten und nur die relevanten Eigenschaften gelernt hat. Z. B. kann ein Lernalgorithmus, dem zu wenig Beispiele zur Verfügung stehen, lernen, dass jedes URL mit `http://` beginnt. Das führt natürlich zu Problemen bei einem URL, das mit `ftp://` anfängt. Dieses Problem ist besonders akut, wenn der Lernalgorithmus nur mit positiven Beispielen trainiert werden kann, was bei dem maschinellen Lernen von Wrappern meistens der Fall ist [20, 25]. Das Testen von maschinell erstellten Wrappern ist natürlich genauso schwierig, wie bei den von Menschen geschriebenen. (Mögliche Testverfahren werden in Kapitel 3 beschrieben.)

2.1.4 Änderungen der Informationsquelle

Das wohl häufigste und wichtigste Problem der Wrapper ist, dass ihre Arbeitsweise voraussetzt, dass die Informationsquelle immer auf dieselbe Weise anzusprechen ist, und auf dieselbe Weise antwortet. Aber in Wirklichkeit ist das leider nicht so, denn die Informationsquellen sind autonom, so dass sie hin und wieder verändert werden können, sowohl inhaltlich, als auch was ihre Schnittstelle betrifft.

Bei Internet-Seiten ist es besonders häufig der Fall (und Suchmaschinen bilden auch keine Ausnahme), dass das Layout der Seite verändert wird, um die Schnittstelle zu vereinfachen, neue Werbungen einzubetten, oder neue Funktionen zu unterstützen [33]. Kushmerick [21, 23] untersuchte 27 Sites 6 Monate lang, und beobachtete, dass 44% der Sites mindestens einmal verändert wurde. Die maximale Anzahl von Änderungen einer Site in 6 Monaten war 4 (Metacrawler[2]). Bei AltaVista zum Beispiel gab es 2 Veränderungen in diesen 6 Monaten.

Die Änderungen, die eine Informationsquelle durchläuft, können weiter in inhaltliche Änderungen, Änderungen der Anfragesprache, und Präsentationsänderungen unterteilt werden. Dies entspricht eben dem Bild, dass die Informationsquelle eine Funktion I verwirklicht, so dass $R = I(Q)$ (siehe Abschnitt 1.2.1 und Abbildung 1.2 auf Seite 3). Es kann entweder die Funktion selbst verändert werden, oder die Art und Weise, wie Q oder R gehandhabt werden.

[2] `http://www.metacrawler.com`

2.1.4.1 Inhaltliche Änderungen

Kleinere inhaltliche Änderungen der Informationsquelle können meistens problemlos vom Wrapper verkraftet werden. Umso mehr, als die inhaltlichen Änderungen oft nur eine Vergrößerung des der Informationsquelle zugrundeliegenden Datenbestandes bedeuten, z. B. entdeckt die Suchmaschine neue Webseiten. Auch wenn gelegentlich Daten aus der Informationsquelle verschwinden (z. B. wird eine nicht mehr existierende Seite aus der Suchmaschine gelöscht), verursacht das meistens keine Fehler.

Andererseits können große Veränderdungen durchaus die Funktion eines ganzen Mediatorprogramms beeinträchtigen. Wenn sich z. B. eine Informationsquelle spezialisiert, und nur noch auf ganz spezielle Anfragen antworten kann, dann kann das zu vielen Fehlern führen.

Auch kleine Veränderungen spielen aber eine überaus wichtige Rolle, in dem sie das Testen von Wrapper sehr schwierig machen. Es liegt hauptsächlich an diesen kleinen, alltäglichen Veränderungen, dass man nicht erwarten kann, dass eine Informationsquelle auf dieselbe Anfrage immer mit derselben Antwort reagiert. Das führt dazu, dass gängige Testmechanismen, wie z. B. Regressionstests bei Wrappern nicht direkt angewendet werden können [21].

2.1.4.2 Änderungen der Anfragesprache

Obwohl die Anfragesprache bei weitem nicht so oft verändert wird, wie der Inhalt oder die Präsentation, ist es natürlich auch nicht auszuschliessen[3]. Wenn allerdings die Art und Weise, wie die Anfragen an die Informationsquelle gestellt werden, sich ändert, verursacht es meistens, dass der Wrapper gar nichts mehr extrahieren kann oder völlig falsche Antworten zurückliefert, so dass diese Änderungen meistens relativ einfach feststellbar sind.

Als Beispiel diene an dieser Stelle wieder eine Internet-Suchmaschine. Bei AltaVista werden die Anfragen an ein Script namens `query` (`www.altavista.com/cgi_bin/query`) gestellt. Dieses Script nimmt die Anfrage in einem Parameter Namens `q` entgegen, z. B. so: `www.altavista.com/cgi_bin/query?q=Titanic`. Wenn nun das Script verändert wird, und die Anfrage nun in einem Parameter anderen Namens erwartet wird, oder wenn der Name des Scripts verändert wird, dann werden Anfragen, die früher funktioniert haben, eine Fehlermeldung auslösen. Die Fehlermeldung bedeutet, dass AltaVista eine Seite mit dem Text, dass wahrscheinlich etwas falsch gemacht wurde, zurückliefert. Der Wrapper wird wohl nichts aus dieser Seite extrahieren können, oder, wenn er doch etwas extrahiert, dann wird es wahrscheinlich relativ einfach sein, zu bemerken, dass die extrahierten Informationen falsch sind.

Es sind theoretisch auch subtilere Änderungen der Anfragesprache möglich, die entweder nur bei manchen Anfragen zur Geltung kommen, oder aber die Bearbeitung der Anfragen nur unbedeutend beeinflussen. Solche Änderungen sind natürlich schwieriger zu bekämpfen. Jedoch kommen solche Probleme in der Praxis relativ selten vor, so dass ihre Bedeutung auch entsprechend geringer ist.

[3] Eine Veränderung dieser Art wurde bei AltaVista am 1. Mai 2001 beobachtet. Mehr dazu in Abschnitt 4.3.1.

2.1.4.3 Präsentationsänderungen

Wie schon früher angedeutet, bilden Präsentationsänderungen die häufigste Fehlerquelle, besonders bei Webseiten. Vor allem kommerzielle Seiten (und die meisten Suchmaschinen sind auch kommerziell) neigen dazu, ihr Layout öfters zu verändern. Das kann zahlreiche Gründe haben, wie z. B.:

- die eingebetteten Werbungen werden verändert bzw. es werden neue Werbungen zugefügt
- neue Funktionen sollen unterstützt werden, z. B. wird eine neue Möglichkeit gegeben, nur MP3[4]-Dateien zu suchen
- die Benutzeroberfläche soll vereinfacht werden
- ein Fehler des Schnittstellendesign wird korrigiert

Ob eine Veränderung der Präsentation Extraktionsprobleme im Wrapper verursacht, hängt auch sehr stark vom Wrapper ab. Grundsätzlich sollten Wrapper so konstruiert werden, dass sie jene kleinen und eigentlich unwesentlichen Änderungen der Präsentation, mit denen auch sehr oft zu rechnen ist (wie z. B. die Änderung der Werbung), tolerieren.

Die meisten Wrapper sind jedoch auf gewisse Regularität der Präsentation angewiesen, so dass eine komplette Umstellung der Schnittstelle, die hin und wieder auch passiert, meistens dazu führt, dass der Wrapper entweder gar nichts mehr extrahieren kann, oder aber falsche Antworten extrahiert.

2.2 Klassifizierung der Symptome

Nachdem wir schon wissen, mit welchen Fehlerquellen beim Zugriff auf externe Informationsquellen zu rechnen ist, können nun die durch diese Fehlerquellen verursachte Symptome klassifiziert werden.

Es wurde schon bei der Aufzählung der Fehlerquellen an mehreren Stellen erwähnt, welche Symptome der jeweilige Fehler verursachen kann. In der Praxis wird man aber wohl umgekehrt vorgehen müssen, und aus dem aufgetretenen Symptom auf die Ursache schliessen, falls das überhaupt möglich ist.

Es ist leider auch vorstellbar, dass ein Fehler überhaupt keine wahrnehmbare Symptome produziert, und trotzdem falsche Antworten verursacht. Wenn z. B. die Informationsquelle von einem böswilligen Dritten von aussen verändert wird, so dass sie falsche Antworten zurückgibt, die aber nicht auffallen (und es gibt auch keine anderen Veränderungen), dann haben wir keine Chance, das zu bemerken. Zum Glück kommt das nicht allzu oft vor.

(Da aber die Informationsquellen autonom sind, ist eine ähnliche Wandel eigentlich nie auszuschliessen. Wenn man also ein System erstellen möchte, das zuverlässiger ist, als die benutzten Informationsquellen, dann müssen andere Methoden verwendet werden.)

[4]Beliebtes Format für die Speicherung und Übertragung von Musik

2.2.1 Probleme bei der Verbindung

Viele Probleme der zugrundeliegenden technischen Infrastruktur (Netzwerk, Server, Proxy, DNS-Server etc.) lassen sich schon im ersten Schritt des Zugriffes auf die externe Informationsquelle, nämlich während des Verbindungsaufbaus, erfassen.

Das ist auch sehr vorteilhaft, weil man dann schon weiss, dass der Fehler in der technischen Infrastruktur liegt, und nichts mit der eigentlichen Informationsquelle zu tun hat. Wenn dieses Symptom ausser Acht gelassen wird, kann es dazu führen, dass die Fehlerursache nicht mehr eindeutig identifiziert werden kann, bzw. dass der Fehler völlig übersehen wird. Wenn nämlich die "Antwort" der Informationsquelle als eine leere Seite weiter bearbeitet wird, so wird der Wrapper wohl auch nichts daraus extrahieren können, und am Ende ist es nicht mehr feststellbar, ob es einen Fehler gab, oder die Antwort auf die Anfrage eben die leere Menge war.

2.2.2 Der Wrapper gerät in eine endlose Schleife

Dieses Symptom scheint nicht allzu oft aufzutreten, ist aber theoretisch möglich, und ist theoretisch auch nicht feststellbar (das Haltungsproblem von Turing-Maschinen ist algorithmisch unentscheidbar). Praktisch kann man jedoch eine Schranke angeben, wie lange der Wrapper maximal arbeitet, und wenn diese Schranke überschritten wird, ist der Wrapper höchstwahrscheinlich in eine endlose Schleife geraten.

Der Grund wird meistens ein Programmierfehler sein. Es kann natürlich auch sein, dass dieser Programmierfehler nur nach einer Veränderung der Informationsquelle hervorgerufen wird, z. B. wird eine implizite Annahme des Programmierers nach der Veränderung nicht mehr erfüllt.

2.2.3 Der Wrapper extrahiert nichts

Eines der häufigsten Symptome ist, dass der Wrapper gar nichts extrahieren kann. Das kann von allen möglichen Fehlerquellen verursacht werden. Wenn die Extraktion z. B. mit Hilfe von regulären Ausdrücken vorgenommen wird, kann schon eine kleine Präsentationsänderung genügen, damit der Wraper nichts mehr extrahieren kann. Aber, wie schon angedeutet, kann dieses Symptom auch dadurch zustande kommen, dass die Antwort auf die Anfrage eben die leere Menge ist (in der relationalen Algebra: die Anfrage selektierte kein Tupel).

Um zwischen diesen beiden Fällen zu differenzieren, können z. B. solche Testanfragen verwendet werden, auf die es garantiert eine (nicht nullwertige) Antwort gibt. Näheres dazu siehe in Kapitel 3.

2.2.4 Der Wrapper extrahiert nicht die korrekte Anzahl von Tupeln

Dieses Symptom ist eine Verallgemeinerung des vorherigen. Dass nicht die korrekte Anzahl von Tupeln extrahiert wird (aber mindestens ein Tupel) scheint nicht so häufig zu sein, wie das vorherige Symptom, aber es kommt auch vor. Es kann mehrere Ursachen haben:

- ➢ der Wrapper überspringt einige Tupeln und extrahiert sie nicht
- ➢ der Wrapper bemerkt die Grenze zwischen einigen nacheinanderfolgenden Tupeln nicht
- ➢ der Wrapper verstümmelt die Tupeln in kleinere Teile, die er als Tupeln identifiziert
- ➢ der Wrapper legt die Grenzen zwischen den Tupeln völlig falsch fest

Dieses Symptom lässt sich dann gut feststellen, wenn es bekannt ist, wieviele Tupeln es geben wird. (Oder zumindest, zwischen welchen Grenzen sich die Anzahl der Tupeln halten wird.) AltaVista z. B. gibt fast immer 10 Tupeln zurück. Wenn also eine Anfrage z. B. 100 Tupeln zurückliefert, dann kann man davon ausgehen, dass der Zugriff wahrscheinlich fehlerhaft war.

(Es ist gar nicht einfach, eine Anfrage zu finden, wofür die Anzahl der von AltaVista gelieferten Tupeln weder 0 noch 10 ist. Bei unseren Experimenten fanden wir nach vielen Fehlversuchen das Wort "vorbeigelaufene", wofür es 4 Treffer gab. Aber bei normalen Suchen – d. h. wenn man nicht extra ein solches Wort finden möchte – passiert das selten.)

2.2.5 Die extrahierten Tupeln sind syntaktisch falsch

Wenn die Verbindung problemlos auf- und abgebaut werden konnte, der Wrapper in keine endlose Schleife geriet und eine plausible Anzahl von Tupeln liefert, dann ist die letzte einfache (d. h. ohne viel Zusatzwissen durchführbare) Validierungsmöglichkeit eine syntaktische Analyse. Damit können noch viele Fehler ausgefiltert werden.

Als einfaches Beispiel für eine syntaktisch falsche Antwort kann die folgende Situation dienen: man fragt eine Informationsquelle nach den Namen gewisser Länder, und erhält Antworten, wie "`><tr><td><img src=`".

Kushmerick behauptet [23], dass die meisten Wrapper, die aus HTML-Seiten extrahieren, bei einem Fehler syntaktisch völlig falsche Antworten liefern, so dass also die meisten Fehler mit Methoden, die auf Syntaxebene arbeiten, detektiert werden können.

2.2.6 Die extrahierten Tupeln sind semantisch falsch

Die am schwierigsten detektierbare Fehler sind diejenigen, die zwar falsche, aber syntaktisch richtige Antworten produzieren, d. h. Antworten, die so aussehen, als wären sie korrekt.

Da aber diese Antworten doch semantisch falsch sind, kann auch irgendwie entdeckt werden, dass sie falsch sind. Das Problem ist nur, dass das sehr viel zusätzliches Wissen erfordert: wenn man alle möglichen semantischen Fehler entdecken können will, muss man über mindestens so viel Wissen verfügen, wie die Informationsquelle selbst.

Es gibt aber auch solche semantische Fehler, die mit weniger Zusatzwissen zu entdecken sind. Dies kann wieder am vorherigen Beispiel verdeutlicht werden. Nehmen wir also wieder an, dass man nach gewissen Ländern sucht, z. B. nach solchen Ländern, wo die Währung Dollar heisst. Nun wird aber die Antwort "Sadarfeguk" zurückgeliefert. Diese

Antwort ist zwar syntaktisch korrekt (es wäre ja theoretisch möglich, dass ein Land so heisst), aber semantisch falsch, denn es gibt kein Land mit diesem Namen. Um diesen Fehler zu entdecken, bedarf es lediglich einer Liste mit den Namen aller Länder (oder einer externen Informationsquelle, von der man abfragen kann, ob es ein Land mit einem gegebenen Namen gibt). Zu beachten ist, dass zur Entdeckung des Fehlers viel weniger Wissen ausreichte, als jenes der Informationsquelle: man brauchte dazu keine Information über die Währung. Es ist aber natürlich möglich, dass die Entdeckung gewisser Fehler doch mehr Wissen erfordert.

An dieser Stelle kann auch argumentiert werden, dass die Entdeckung solcher Fehler die Aufgabe des Mediators ist. Es ist aber eine Validierungsaufgabe, die auch im Mediator durchgeführt werden kann. Abschnitt 3.3 beschreibt Möglichkeiten, wie das jeweilige Validierungsverfahren in die Mediatorarchitektur eingebunden werden kann.

2.3 Dauer des Fehlers oder der Veränderung

Eine weitere Möglichkeit, die Fehler zu klassifizieren, ist nach ihrer Dauer. Das ist vor allem aus dem Sichtpunkt der Fehlerbehebung wichtig.

2.3.1 Temporäre Fehler bzw. Veränderungen

Viele Fehler, die von der technischen Infrastruktur verursacht werden, sind temporär. Wenn z. B. ein Server überlastet ist, dann kann es sein, dass eine Wiederholung der Anfrage schon mit der gewünschten Antwort zurückkehrt. Also wiederholt sich der Fehler nicht.

Bei ernsthafteren technischen Fehlern, wie z. B. bei einer Beschädigung des Proxy, kann es stunden- oder sogar tagelang dauern, bis die Verbindung zur Informationsquelle wieder hergestellt werden kann.

Jedenfalls ist die Veränderung des Mediators oder der Wrapper in solchen Situationen meistens nicht nötig, so dass der Mediatorfachmann auch nicht notwendigerweise informiert werden muss.

Bei einer temporären Veränderung ist es ausserdem gar nicht sicher, ob die Mediatoranwendung diese Veränderung überhaupt mitbekommt.

2.3.2 Permanente Fehler bzw. Veränderungen

Bei permanenten Fehlern und Veränderungen müssen in der Regel Gegenmaßnahmen getroffen werden, also muss entweder der Mediatorfachmann benachrichtigt werden, oder die Reparatur muss automatisch durchgeführt werden.

Ein typischer permanenter Fehler ist z. B., dass eine Informationsquelle dauerhaft nicht mehr zugänglich ist. In diesem Fall muss natürlich die Mediatoranwendung entsprechend verändert werden, so dass sie andere Informationsquellen benutzt (wenn das möglich ist).

Eine typische permanente Veränderung ist, dass sich das Layout, also die Präsentation der Antworten der Informationsquelle ändert. In diesem Fall muss der ensprechende Wrapper

auch mitverändert werden. Das kann entweder von Hand gemacht werden, kann aber auch automatisch erfolgen.

Kapitel 3

Lösungsansätze

Nachdem die möglichen Fehler, die bei dem Zugriff auf externe Informationsquellen in der Mediatorarchitektur auftreten können, aufgelistet und in mehrerer Hinsicht klassifiziert wurden, können nun auch die verschiedenen Lösungsmöglichkeiten behandelt werden. Dieses ist das zentrale Kapitel der Arbeit.

Die eigentlichen Lösungsansätze werden in Abschnitt 3.2 präsentiert. Um aber die verschiedenen Validierungsmöglichkeiten miteinander vergleichen zu können (was in Abschnitt 3.2.6 gemacht wird), muss zuerst geklärt werden, was unter der Qualität einer Validierungsmethode zu verstehen ist. Das ist der Zweck von Abschnitt 3.1. Schließlich werden in Abschnitt 3.3 Möglichkeiten zur Einbindung der vorher präsentierten Validierungsmethoden (Validatoren) in die Mediatorarchitektur vorgestellt.

Zu beachten ist, dass in diesem Kapitel alle Verfahren, die in irgendeiner Weise die Korrektheit von Zugriffen auf externe Informationsquelle(n) überprüfen, Validierungsmethoden (oder Validatoren) genannt werden. Also auch die konventionelle Fehlerbehandlung (z. B. im Fall eines Fehlers beim Verbindungsaufbau im Wrapper) gehört in diese Kategorie, aber auch komplizierte Verfahren, die nur im Mediator verwirklicht werden können. In Abschnitt 3.1 und 3.2 werden alle Validierungsmethoden gleich behandelt. Die eigentliche Realisierung und Einbindung in die Mediatorarchitektur wird erst in Abschnitt 3.3 beschrieben.

3.1 Qualität einer Validierungsmethode

Um die Qualität eines Validators bewerten zu können, muss erst geklärt werden, was von einem "guten" Validator zu erwarten ist. Diese Bewertung hat zweierlei Ziele: erstens wird dadurch deutlich, worauf bei dem Design und bei der Implementierung von Validierungsverfahren zu achten ist, und zweitens wird dadurch ein Vergleich der verschiedenen Verfahren möglich. Die Bewertung ist aber gar nicht so einfach, wie es auf den ersten Blick scheint. Es kann auch oft sehr schwierig sein, alle diese Merkmale gleichzeitig zu erfüllen. Die nächsten Abschnitte beschreiben einige wichtige Merkmale, und die Schwierigkeiten, die mit ihrer Bewertung verbunden sind.

3.1.1 Korrektheit, Spezifizität, Sensitivität, prediktive Werte

Selbstverständlich wird erwartet, dass der Validator korrekt ist, d. h. korrekt beurteilen kann, ob der Zugriff auf die externe Informationsquelle fehlerfrei abgelaufen ist, oder Fehler aufgetreten sind.

Leider – wie es sich aus Kapitel 2 herausstellt – gibt es Fehler, die gar nicht, oder nur sehr schwierig zu erkennen sind. Dementsprechend ist die Erwartung, dass der Validator immer hundertprozentig korrekt funktioniert, im Allgemeinen nicht realistisch. Stattdessen bemühen wir uns, Methoden zu finden, die im statistischen Sinne gut funktionieren.

Nun stellt sich die Frage, wie man die Korrektheit statistisch am besten messen kann. Da dieser Punkt zentrale Bedeutung für die nächsten Abschnitte hat, machen wir an dieser Stelle einen kleinen Exkurs in die angewandte Statistik. Zuerst wird das Problem formalisiert.

Es werden N *Tests* mit dem Validator durchgeführt, d. h. der Validator validiert N Zugriffe eines Wrappers auf eine externe Informationsquelle. Von diesen Zugriffen laufen k fehlerfrei ab, in $N - k$ passiert ein Fehler. In jedem der Tests muss der Validator ein Urteil treffen, ob der jeweilige Zugriff fehlerfrei war. Von den N Urteilen entsprechen m der Wahrheit, $N - m$ sind falsch. Die Korrektheit des Validators wird als

$$K = \frac{m}{N}$$

definiert. Sie ist eine Zahl zwischen 0 und 1.

Was sagt uns diese Zahl? Ist z. B. eine Korrektheit von 0.8 gut oder schlecht? Ist eine Korrektheit von 0.999 viel besser als eine Korrektheit von 0.99? Die Antwort auf diese Fragen hängt sehr stark von k, oder genauer formuliert, von k/N ab.

Normalerweise kann man annehmen, dass der Wrapper meistens fehlerfrei funktioniert, und nur ganz selten einen Fehler macht. Das bedeutet $k \approx N$. Für einen ganz "dummen" Validator, der jeden Zugriff als fehlerfrei beurteilt, gilt $m = k$, und somit hat er eine Korrektheit von k/N, was fast 1 ist!

Wie man sieht, ist die Korrektheit allein in solchen extremen Fällen, in denen eines der beiden Urteile eine viel höhere Wahrscheinlichkeit hat als die andere, kein aussagekräftiges Maß der Qualität eines Tests. Leider ist die Validierung von Wrapper so ein extremer Fall. Also muss ein besseres Maß gefunden werden.

Es gibt zwei verschiedene Arten von Fehlurteilen des Validators: entweder beurteilt er einen fehlerfreien Zugriff als fehlerhaft, oder einen fehlerhaften Zugriff als fehlerfrei. Diese beiden Fehlerarten (es geht dabei nicht um Fehler des Wrappers, sondern um Fehler des Validators!) werden in der Statistik normalerweise Fehler 1. Art und Fehler 2. Art genannt (siehe Tabelle 3.1).

Dabei wird der Validator (in der Sprache der Statistik) als Hypothesentester betrachtet. Seine Nullhypothese (H_0) ist, dass der Wrapper korrekt funktioniert und der Zugriff fehlerfrei abgelaufen ist. Die Alternativhypothese ist, dass während des Zugriffes ein Fehler aufgetreten ist. Sein Urteil ist das Ergebnis des Hypothesentests. (Für eine Erläuterung über Hypothesentests, siehe z. B. `http://www.uni-bielefeld.de/~hjawww/glossar/hypothes.htm`.)

	der Zugriff war	
Urteil des Validators	fehlerfrei	fehlerhaft
Zugriff fehlerfrei	richtig	Fehler 2. Art
Zugriff fehlerhaft	Fehler 1. Art	richtig

Tabelle 3.1: Fehler 1. und 2. Art

Wie gesagt, wird die Beurteilung des Tests dadurch schwierig, dass die Wahrscheinlichkeit des einen Ergebnisses viel größer ist, als die des anderen. Ein ähnliches Problem tritt in einigen Zweigen der Medizinwissenschaften (klinische Chemie, Hämatologie) auf: es werden Tests benutzt, um zu beurteilen, ob der Patient eine gewisse seltene Krankheit (z. B. AIDS) hat. Da die meisten Krankheiten nur bei einer kleinen Minderheit der Population auftreten, ist die Wahrscheinlichkeit, dass der Patient an der jeweiligen Krankheit leidet, sehr gering.

Um dieses Problem zu umgehen, werden in der Medizin zwei verschiedene Messgrößen eingeführt: Spezifizität und Sensitivität [38]. Spezifizität ist die Wahrscheinlichkeit eines richtigen negativen Tests bei wirklich nicht vorliegender Krankheit:

$$\text{Spezifizität} = Pr(\text{der Test ist negativ}|\text{der Patient ist gesund})$$

Sensitivität ist die Wahrscheinlichkeit eines richtigen positiven Tests bei wirklich vorliegender Krankheit:

$$\text{Sensitivität} = Pr(\text{der Test ist positiv}|\text{der Patient ist krank})$$

(Der Zusammengang zum Validator wird durch die folgenden Entsprechungen ersichtlich: der Patient ist krank $\leftrightarrow$ der Zugriff war fehlerhaft; der Patient ist gesund $\leftrightarrow$ der Zugriff war fehlerfrei; das Ergebnis des Tests $\leftrightarrow$ das Urteil des Validators.)

Weiterhin gibt es noch

$$\text{Unspezifizität} = 1 - \text{Spezifizität} = Pr(\text{der Test ist (falsch) positiv}|\text{der Patient ist gesund})$$

und

$$\text{Unsensitivität} = 1 - \text{Sensitivität} = Pr(\text{der Test ist (falsch) negativ}|\text{der Patient ist krank})$$

Diese Definitionen sind zwar symmetrisch, aber wegen dem großen Unterschied zwischen den Wahrscheinlichkeiten der zwei Fälle ist die Aussagekraft der beiden Größen nicht gleich. Um das zu erklären, betrachten wir die umgekehrten bedingten Wahrscheinlichkeiten (oft auch positiver bzw. negativer prediktiver Wert genannt):

$$Pr(\text{krank}|\text{positiv}) \text{ und } Pr(\text{gesund}|\text{negativ})$$

Diese zeigen an, wie "glaubwürdig" der Test ist: wenn also z. B. der Validator den Zugriff als fehlerhaft beurteilt hat, mit welcher Wahrscheinlichkeit hat er recht? Aus dem Satz von Bayes folgt:

$$Pr(\text{krank}|\text{positiv}) = Pr(\text{positiv}|\text{krank}) \cdot \frac{Pr(\text{krank})}{Pr(\text{positiv})} = \text{Sensitivität} \cdot \frac{Pr(\text{krank})}{Pr(\text{positiv})}$$

$$Pr(\text{gesund}|\text{negativ}) = Pr(\text{negativ}|\text{gesund}) \cdot \frac{Pr(\text{gesund})}{Pr(\text{negativ})} = \text{Spezifizität} \cdot \frac{Pr(\text{gesund})}{Pr(\text{negativ})}$$

Nehmen wir an, dass sowohl die Sensitivität, als auch die Spezifizität sehr hoch (also fast 1) sind. Auch die Wahrscheinlichkeiten $Pr(\text{gesund})$ und $Pr(\text{negativ})$ sind hoch. Daraus folgt, dass die Wahrscheinlichkeit

$$Pr(\text{gesund}|\text{negativ})$$

also der negative prediktive Wert auch sehr hoch ist, das ist also in Ordnung. Aber der positive prediktive Wert ist nicht unbedingt hoch, denn sowohl $Pr(\text{krank})$ als auch $Pr(\text{positiv})$ ist sehr gering, also kann ihr Quotient auch sehr klein sein! Das passiert, wenn der Test relativ viele falsche positive Antworten gibt, und somit $Pr(\text{positiv}) \gg Pr(\text{krank})$ gilt.

Um zu untersuchen, welche Konsequenzen das für die Sensitivität und Spezifizität hat, formen wir den obigen Ausdruck weiter um:

$$\begin{aligned}
Pr(\text{krank}|\text{positiv}) &= \text{Sensitivität} \cdot \frac{Pr(\text{krank})}{Pr(\text{positiv})} = \\
&= \text{Sensitivität} \cdot \frac{Pr(\text{krank})}{Pr(\text{positiv}|\text{krank})Pr(\text{krank}) + Pr(\text{positiv}|\text{gesund})Pr(\text{gesund})} = \\
&= \frac{\text{Sensitivität} \cdot Pr(\text{krank})}{\text{Sensitivität} \cdot Pr(\text{krank}) + (1 - \text{Spezifizität})(1 - Pr(\text{krank}))} = \\
&= \frac{1}{1 + \frac{(1-\text{Spezifizität})(1-Pr(\text{krank})}{\text{Sensitivität} \cdot Pr(\text{krank})}} = \frac{1}{1 + \frac{(1-x)(1-p)}{yp}} = f_p(x, y)
\end{aligned}$$

(In den letzten beiden Gleichungen wurden lediglich einige Bezeichnungen eingeführt: $x =$ Spezifizität, $y =$ Sensitivität, $p = Pr(\text{krank})$.) Um diese Wahrscheinlichkeit zu maximieren, muss der im Nenner stehende Bruch minimiert werden. (Im idealen Fall wird er 0 sein, sonst ist er positiv.) Dazu muss entweder sein Zähler minimiert, oder sein Nenner maximiert werden. Wie man sieht, ist $Pr(\text{krank})$ eine obere Grenze für seinen Nenner, also kann er mit Hilfe von der Erhöhung der Sensitivität nicht beliebig erhöht werden! Wenn aber die Spezifizität ihren Idealwert, die 1, erreicht, wird auch dieser Bruch 0 sein.

Wenn man diese Funktion auch grafisch darstellt (siehe Abbildung 3.1), dann wird ganz klar, dass aus dieser Hinsicht also die Spezifizität eine viel wichtigere Rolle spielt, als die Sensitivität.

Dieses Ergebnis kann auch analytisch unterstützt werden, indem man die partiellen Ableitungen der in Abbildung 3.1 abgebildeten Funktion $f(x, y)$ bildet:

$$\begin{aligned}
\frac{\partial f}{\partial x} &= \frac{1-p}{yp} \cdot \left(1 + \frac{(1-x)(1-p)}{yp}\right)^{-2} \\
\frac{\partial f}{\partial y} &= \frac{(1-x)(1-p)}{y^2 p} \cdot \left(1 + \frac{(1-x)(1-p)}{yp}\right)^{-2}
\end{aligned}$$

also gilt

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial x} \cdot \frac{1-x}{y} \ll \frac{\partial f}{\partial x}$$

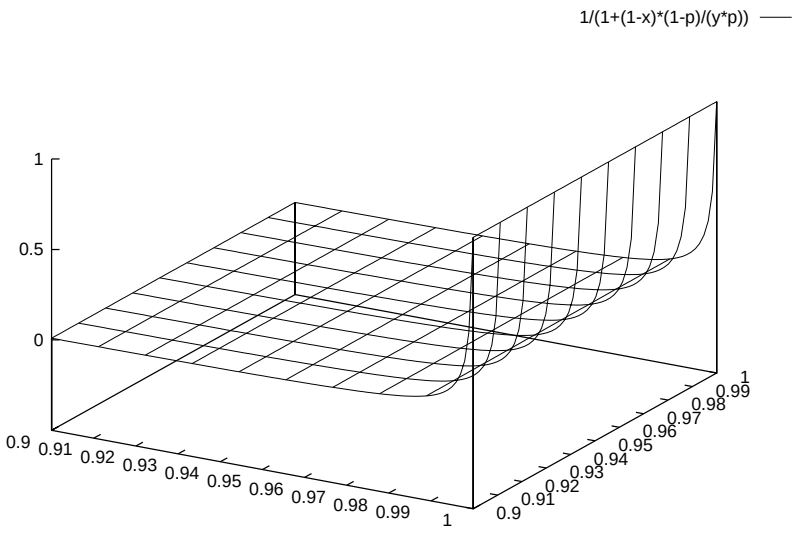

Abbildung 3.1: $Pr(\text{krank}|\text{positiv})$ als die Funktion der Spezifizität (x) und der Sensitivität (y). Der Parameter p ist die Häufigkeit der Kranken, also $p = Pr(\text{krank})$. In diesem Fall wurde $p = 10^{-3}$ angenommen.

(bei der letzten Ungleichung wurde $x \approx 1$ und $y \approx 1$ angenommen). Dieses Ergebnis bedeutet, dass der Funktionswert (also die bedingte Wahrscheinlichkeit $Pr(\text{krank}|\text{positiv})$) viel stärker von x (also von der Spezifizität) als von y (also von der Sensitivität) abhängt.

Was den Validator betrifft, kann man daraus den Schluss ziehen, dass er eine ganz hohe Spezifizität aufweisen muss, damit seine Urteile glaubwürdig sind. Wie hoch die Spezifizität nun wirklich sein muss, hängt stark von der Anwendung bzw. von der Umgebung ab. Wenn nämlich die Spezifizität nicht hoch genug ist, verursacht das viele falsche positive Urteile; und wenn jedes mal der Mediatorfachmann unnötig gerufen wird, dann ist der Test unbrauchbar. Wenn jedoch eine – nicht allzu teuere – automatische Reparatur möglich ist (z. B. wenn der Wrapper sowieso automatisch erstellt wurde, und in solchen Fällen nur neu erstellt werden muss), ist das kein großes Problem.

3.1.2 Nicht-funktionale Qualitätsmerkmale

Neben der Anforderung, dass der Validator möglichst korrekte Urteile geben soll, gibt es weitere Aspekte, die beachtet werden müssen. Bei diesen geht es grundsätzlich darum, dass der Preis, den wir für die automatische Validierung bezahlen müssen, minimiert werden soll. Diese Merkmale sind meistens nicht direkt quantifizierbar, so dass ihre Beurteilung noch schwieriger ist, als die der Korrektheit.

Im Folgenden wird eine Aufzählung und kurze Beschreibung der wichtigsten nicht-funktionalen Qualitätsmerkmale eines Validators gegeben.

Effizienz. Die Validierung soll möglichst schnell sein, und soll die Resourcen (Prozessor,

Hauptspeicher, Hintergrundspeicher, etc.) des ausführenden Systems so wenig wie möglich belasten. Wichtig ist, dass die Effizienz des Zugriffes auf die externe Informationsquelle durch den Validator nicht wesentlich beeinflusst werden darf. Daraus folgt auch, dass online, also während der Arbeit des Mediatorsystems nur relativ einfache Tests durchgeführt werden können. Wenn auch kompliziertere Tests benötigt werden, müssen sie off-line, wenn das System gerade nicht benutzt wird, gemacht werden.

Programmieraufwand. Der Validator bedeutet extra Programmieraufwand, der in Abhängigkeit von der Komplexität des Validators stark variieren kann. Dabei muss natürlich nicht nur die Erstellung des Validators, sondern auch seine Wartung in Betracht genommen werden. Der Gesamtaufwand soll also möglichst klein gehalten werden.

Allgemeinheit. Der Validator sollte möglichst wenig Wrapper-spezifisch sein, d. h. er sollte sich nicht auf die Funktionsweise des Wrappers verlassen. Sonst muss nämlich bei jeder Veränderung des Wrappers auch der Validator entsprechend verändert werden.

Wiederverwendbarkeit. Die Wiederverwendbarkeit des Validators hängt auch mit der Allgemeinheit zusammen, geht jedoch noch darüber hinaus. Sie bedeutet, dass der Validator, oder Komponenten des Validators im Zusammenhang mit ganz anderen Wrappern und Informationsquellen auch benutzt werden können.

Selbstanpassungsfähigkeit. Eines der wohl am schwierigsten definierbaren (und erreichbaren) Merkmale ist die Selbstanpassungsfähigkeit. Sie bedeutet, dass sich der Validator an eine Veränderung der Ausgabe des Wrappers, die korrekt ist, anpassen kann. Zur Verdeutlichung stehe an dieser Stelle ein Beispiel. Stellen wir uns vor, dass ein Wrapper Namen von Personen extrahiert. Bis einem Zeitpunkt waren die Namen, die die Informationsquelle produzierte, voll ausgeschrieben, danach wird aber die Informationsquelle so verändert, dass die Vornamen abgekürzt werden. Ein Validator, der auf syntaktischer Ebene arbeitet, kann diese Veränderung als einen Fehler wahrnehmen, so dass also der Validator neu programmiert werden muss, damit er nicht ständig Fehlermeldungen produziert. Ein selbstanpassungsfähiger Validator würde sich jedoch an die neuen Verhältnisse anpassen.

Gesamtkosten. In einer kommerziellen Umgebung wird der Einsatz von verschiedenen Validatorlösungen wohl anhand ihrer Gesamtkosten beurteilt. Sie ergeben sich aus den Kosten des Design, der Implementierung und der Wartung des Validators, sowie aus den Kosten der eventuellen Änderungen, die am ganzen Mediatorsystem durchgeführt werden müssen, um den Validator anbinden zu können (siehe Abschnitt 3.3).

3.2 Ansätze

In diesem Abschnitt werden verschiedene Validierungsmöglichkeiten vorgestellt. Die Methoden werden nach dem Grundprinzip ihrer Arbeitsweise klassifiziert. Am Ende des Abschnittes wird auch ein Vergleich der präsentierten Methoden gegeben. Wie diese Methoden in der Praxis eingesetzt werden können, wird an der Fallstudie von Kapitel 4 demonstriert.

3.2.1 Validierung während der Arbeit des Wrappers

In die erste Gruppe gehören Validierungsmethoden, die die Arbeitsweise des Wrappers validieren (und nicht nur seine Ausgabe). Diese Methoden können als *White-Box-Testmethoden* verstanden werden.

Der große Vorteil dieser Verfahren ist, dass während der Arbeit des Wrappers noch sämtliche Informationen über den Zugriff verfügbar sind. Später, wenn man nur die Ausgabe des Wrappers analysieren kann, ist schon ein Informationsverlust in Kauf zu nehmen. Dieses kann an dem folgenden (schon erwähnten) Beispiel gesehen werden: wenn die Ausgabe eines Wrappers leer ist, kann man im Allgemeinen nicht mehr entscheiden, ob das durch einen Fehler verursacht worden ist, oder aber die gültige Antwort – dass eben kein Tupel selektiert wurde – ist.

Der Nachteil von diesem Ansatz ist, dass es sehr Wrapper-spezifisch ist, also von der Arbeitsweise des Wrappers absolut abhängig ist. Daraus folgt, dass nach einer Umprogrammierung oder sonstigen Veränderung des Wrappers wohl auch der Validator verändert werden muss, wenn das überhaupt noch möglich ist. Also wird dadurch die Wartung schwierig, was auch die Kosten einer solchen Lösung drastisch erhöhen kann. Wenn der Wrapper und der Validator praktisch zusammengebaut sind, kann das auch dazu führen, dass der Validator überhaupt nicht wiederverwendbar sein wird, nicht einmal seine Komponenten.

Wegen dieser hohen Wrapper-Spezifizität ist es auch nicht einfach, allgemeine Methoden dieser Klasse zu nennen. Stattdessen werden einige Beispiele genannt, die im Metasearch-Beispiel Verwendung finden könnten:

- Erkennung eines rein technischen Fehlers, dass z. B. die Seite nicht heruntergeladen werden konnte.
- Erkennung, wenn das RegExp Engine (also das System, dass die regulären Ausdrücke verarbeitet, und die entsprechenden Informationen extrahiert) fehlerhaft funktioniert, z. B. dadurch, dass es ungewöhnlich lange arbeitet.
- Wenn auch das RegExp Engine selber erstellt wurde, kann vielleicht sogar erkannt werden, dass der zugrundeliegende endliche Automat sich in einer ungewöhnlichen Weise verhält (weil z. B. ungewöhnliche Zustandfolgen vorkommen).

Es ist möglich, einige dieser Wrapper-internen Informationen im Validator zu benutzen, und gleichzeitig eine relativ hohe Wrapper-Unabhängigkeit zu erreichen. Dazu bedarf es einer definierten Schnittstelle zwischen Wrapper und Validator, die für den Wrapper spezifiziert, wie er diese Informationen dem Validator zur Verfügung stellen soll. Dann kann nämlich der Validator diese Informationen genauso benutzen, als wären sie ein Teil der Ausgabe des Wrappers. Somit kann also der Wrapper doch als schwarzer Kasten (engl. black box) betrachtet werden.

3.2.2 Validierung der Ausgabe des Wrappers

Im Gegensatz zu den Methoden des vorherigen Abschnitts dienen in diesen Verfahren nicht die internen Vorgänge des Wrappers zur Validierung, sondern ausschließlich seine Ausgabe. Also geht es in diesem Fall mehr um Black-Box-Testverfahren.

Die Vor- und Nachteile sind hier eben umgekehrt, als vorher. Vorteil ist die vollständige Unabhängigkeit von den Implementierungsdetails des Wrappers. Nachteil ist, dass Informationen über den inneren Zustand des Wrappers bei der Validierung nicht benutzt werden können.

Die Grundlage dieser Methoden ist, dass einige Ausgaben des Wrappers bei normalem Verlauf des Zugriffes nicht vorkommen können. Sei das Alphabet der Wrapper-Ausgaben Σ; dann sind die möglichen Ausgaben Elemente der Menge Σ^*. (Wenn die Länge der Ausgabe durch eine Zahl z begrenzt werden kann, dann können die Ausgaben auch so angesehen werden, wie Elemente des Raumes Σ^z.) Die Menge der plausiblen Ausgänge (also derjenigen, die im normalen Verlauf des Zugriffes vorkommen können) sei $L \subseteq \Sigma^*$. Es kann auch sein, dass die Menge L in manchen Fällen eher als eine unscharfe Menge [19] definiert werden kann, da die Grenze zwischen plausiblen und nicht-plausiblen Ausgaben unscharf sein kann: möglicherweise sind einige Ausgaben zwar sehr unwahrscheinlich, aber theoretisch möglich.

Die Aufgabe des Validators kann also folgenderweise formuliert werden: gegeben eine Ausgabe a des Wrappers; ist $a \in L$? Wenn die Menge L klein ist, kann der Validator einfach eine Liste mit den Elementen von L enthalten. Wenn aber L groß (vielleicht sogar unendlich) ist, kann dieses einfache Verfahren nicht benutzt werden. Um dieses Problem zu lösen, können Ergebnisse und Methoden der Formalen Sprachen [2] verwendet werden. Oft kann z. B. die Aufgabe mittels endlicher Automaten, oder in äquivalenter Weise mittels regulärer Ausdrücken gelöst werden.

Oft sind die Methoden der Formalen Sprachen aber unzureichend. Die wichtigsten Probleme sind:

- ➢ Ungewissheit und unscharfe Grenzen müssen gehandhabt werden
- ➢ Es ist oft sehr schwierig, eine Grammatik oder einen Automaten für eine Sprache zu konstruieren
- ➢ Es kann auch sein, dass sich die Sprache gar nicht durch eine Grammatik definieren lässt
- ➢ Der resultierende Automat kann zu viel Resourcen (CPU-Zeit oder Speicher) brauchen

In solchen Problemfällen können verschiedene andere Methoden helfen. Die Aufgabe kann als ein Klassifizierungsproblem gestellt werden. Es gibt im Wesentlichen zwei verschiedene Ansätze:

- ➢ Man verwirklicht eine Methode, die in irgendeiner Weise die Elemente von L spezifiziert. Dabei können z. B. binäre Entscheidungsdiagramme (Binary Decision Diagram, BDD), unscharfe Logik, oder verschiedene Annäherungen verwendet werden. Möglicherweise ist L zwar schwierig zu spezifizieren, aber eine ähnliche Sprache L' kann vielleicht schon einfacher beschrieben werden. Z. B. kann L durch eine relativ kleine Änderung kontext-frei gemacht werden (obwohl L selbst kontext-sensitiv ist). Dadurch können die Gesamtkosten der Validierung drastisch gesenkt werden, und

wenn der Unterschied zwischen L und L' nicht allzu groß ist, dann wird dadurch die Korrektheit der Validierung nur unwesentlich verschlechtert.

➢ Das Klassifikationsproblem kann auch automatisch, durch maschinelles Lernen, gelöst werden. Normalerweise werden dafür positive und negative Beispiele benötigt, d. h. Elemente aus L und Elemente aus $\Sigma^* \setminus L$, damit der Lernalgorithmus die Grenze zwischen den beiden Mengen lernen kann. Im Falle des Mediators bedeutet das, dass der Lernalgorithmus mit möglichen und unmöglichen Wrapper-Ausgaben trainiert werden muss (und man muss natürlich auch bei jedem Beispiel angeben, zu welcher Menge es gehört). Es gibt aber auch Lernverfahren, die aus ausschließlich positiven Beispielen lernen können [25]. Meistens reichen schon um die 100 Beispiele (diese Zahl kann aber natürlich je nach Lernverfahren variieren), damit der Lernalgorithmus dann mit relativ guter Sicherheit das Klassifizierungsproblem lösen kann. Noch mehr Beispiele helfen meistens nicht viel weiter. In Abbildung 3.2 ist eine typische Lernkurve dargestellt.

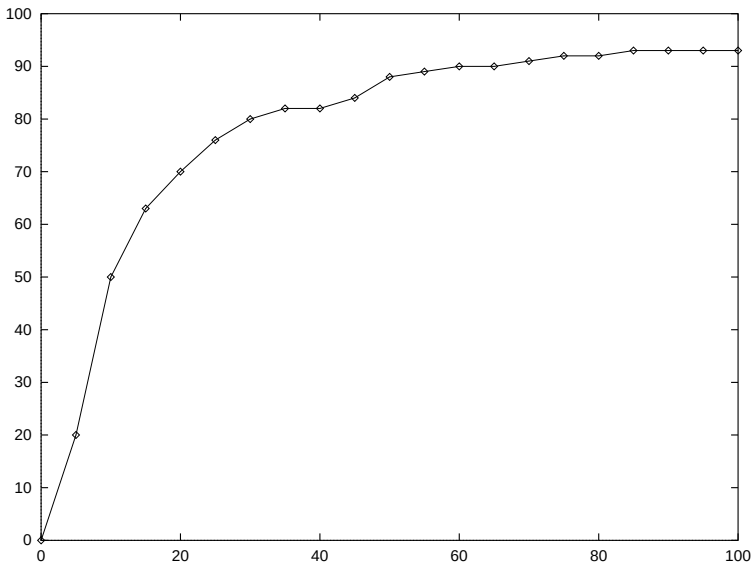

Abbildung 3.2: Eine typische Lernkurve. Auf der horizontalen Koordinatenachse ist die Anzahl der Beispiele, mit denen der Lernalgorithmus trainiert wurde, aufgetragen. Die vertikale Koordinatenachse zeigt die erzielte (prozentuale) Korrektheit der Klassifizierung an. Gut erkennbar sind der anfangs sehr steile Anstieg und die Sättigung bei einer Korrektheit von unter 100%.

Beide Methoden können hauptsächlich syntaktische Fehler ausfiltern. Beide Methoden verlassen sich auf gewisse Regularitäten der Menge L: wenn die Elemente von L keine Regularität aufweisen würden, gäbe es praktisch keine andere Möglichkeit, als sie aufzuzählen.

Die automatisch gelernte Klassifizierung hat auf jeden Fall den Vorteil des geringeren Aufwandes. Das zeigt sich vor allem in der Wartung: wenn sich die Sprache L ändert und der Validator angepasst werden muss, dann muss der lernende Validator nur neu trainiert werden (was unter Umständen auch automatisch erfolgen kann, siehe z. B. [20]), während der von Hand implementierte Validator umprogrammiert werden muss. Daraus folgt auch,

dass ein mit maschinellem Lernen arbeitender Validator flexibler und allgemeiner ist, als ein von Hand implementierter. Andererseits hat man keine Garantie, dass maschinelles Lernen fehlerfrei funktionieren würde, und das tut es meistens auch nicht, zumindest nicht mit einer endlichen Menge von Trainingsdaten (siehe Abbildung 3.2). Also sind von Hand implementierte Validatoren in der Regel zuverlässiger. (Diese Vor- und Nachteile gelten allgemein für maschinell gelernte bzw. von Hand erstellte Lösungen, also z. B. auch für den Vergleich von automatisch und von Hand erstellten Wrapper.)

Die Klassifizierung kann oft durch eine gut gewählte Transformation erleichtert werden. Sei $T : \Sigma^* \to X$ eine Transformation; das Bild von L wird mit $T(L)$ bezeichnet. T ist dann nützlich, wenn die Elemente von $T(L)$ eihe höhere Regularität aufweisen, als die von L, und damit die Klassifizierung in X einfacher wird als in Σ^*.

Ein typisches Beispiel für eine solche Transformation ist die Verwendung von sog. *Features*, also charakteristischen Eigenschaften. Dadurch wird meistens auch eine starke Dimensionsreduktion erreicht, denn es werden in der Regel nur wenig (z. B. 5 oder 10) Features benutzt. Wenn die Anzahl der verwendeten Features k ist, dann ist X eben ein k-dimensionaler Raum. Wenn X klein ist, können die Elemente von $T(L)$ sogar aufgezält werden. Aber meistens ist der Zweck der Dimensionsreduktion nicht, dass der resultierende Raum so klein wie möglich sei, sondern dass eben die Regularität dadurch erhöht wird.

Kushmerick benutzt für seinen Algorithmus RAPTURE [21, 23] statistische Features der Wrapperausgabe, wie z. B. die Anzahl der extrahierten Tupeln, durchschnittliche Länge der extrahierten Texte, durchschnittliche Anzahl von Wörter usw. Das sind numerische Features, also kann der resultierende Raum als $\mathbb{R}^k$ angesehen werden. Die implizite Annahme ist, dass $T(L)$ ein konvexer Körper dieses Raumes ist. Ein wichtiger Vorteil ist, dass diese Features nicht problem-spezifisch sind, so dass RAPTURE in den verschiedensten Domänen eingesetzt werden kann.

Zum Schluss dieses Abschnittes werden nun die Quellen der Ungenauigkeiten dieser Methoden zusammengefasst:

- ➢ Aus der Ausgabe des Wrappers ist es nicht notwendigerweise ersichtlich, ob der Zugriff fehlerhaft oder fehlerfrei war
- ➢ Einfachheitshalber wird man vielleicht von L zu einer anderen, ähnlichen Sprache L' übergehen, um die Klassifizierung zu erleichtern
- ➢ Programmierfehler
- ➢ Die durch maschinelles Lernen gelernte Grenze zwischen L und $\Sigma^* \setminus L$ enspricht nicht genau der Wahrheit
- ➢ Durch Dimensionsreduktion geht Information verloren

3.2.3 Einbeziehen anderer Informationsquellen

Wie schon in Kapitel 2 beschrieben wurde, ist für die Behebung semantischer Fehler zusätzliches Wissen erforderlich. In einer Mediatorumgebung ist die natürliche Methode,

Wissen einzubringen, eben die Anbindung externer Informationsquellen. Diese Informationsquellen können dann benutzt werden, um festzustellen, ob die Ausgabe des Wrappers im gegebenen Umfeld einen Sinn macht. Damit wird eine Art Redundanz erreicht, was die Zuverlässigkeit des Systems stark erhöhen kann[31].

Mögliche Informationsquellen, die in der Validierung Verwendung finden können:

- Informationsquellen, die sowieso schon Bestandteile der jeweiligen Mediatoranwendung sind. Diese Art von Wiederverwendung hilft, die Kosten der Validierung niedrig zu halten.
- Informationsquellen, die extra für die Validierung eingebunden werden. Die entsprechenden Wrapper müssen natürlich auch implementiert (und theoretisch auch mit einem Validator versehen) werden. Das steigert offensichtlich die Implementierungs- und Wartungskosten.
- Der Benutzer selbst. Er kann entweder direkt gefragt werden, ob die Ausgabe des Wrappers Sinn macht, oder kann man aus seinem Verhalten folgern (z. B. bei Metasearch: ob er auf die angebotenen URLs klickt). Wenn die Mensch-Maschine-Schnittstelle das ermöglicht, kann auch sein Gesichtsausdruck erfasst und analysiert werden. Heutzutage ist das noch nicht üblich, aber die Möglichkeit besteht [35]. Die Meinung des Benutzers kann das Urteil des Validators entweder einfach ausser Kraft setzen, oder aber in einem komplexeren Lernverfahren (z. B. reinforcement learning [18]) kann es als "Belohnung" für den Validator dienen.

3.2.4 Testanfragen

Beim Testen von herkömmlicher Software spielen Regressionstests [21] eine sehr wichtige Rolle. Dabei wird die Ausgabe der Software für bestimmte Eingaben überprüft, indem die jeweilige Ausgabe mit einer vordefinierten korrekten Ausgabe verglichen wird.

Dieses Verfahren kann in Mediatoranwendungen meistens nicht unmittelbar verwendet werden, denn es besteht keine Garantie, dass die externen, autonomen Informationsquellen auf dieselbe Anfrage immer dieselbe Antwort geben. Anders formuliert, wenn die Ausgabe des Wrappers nicht genau das gleiche ist, wie vorher – was beim Regressionstest als Fehler identifiziert wird –, folgt daraus noch nicht, dass der Zugriff in der Tat fehlerhaft ist. Dieses Verfahren funktioniert also nur für Informationsquellen, die in dieser Hinsicht statisch sind. Kushmerick fand [23], dass 19% der von ihm untersuchten Web-basierten Informationsquellen diese Eigenschaft hatten (darunter war z. B. die Bibel und die Verfassung der Vereinigten Staaten).

Für die restlichen 81% ist die Methode unmittelbar nicht einsetzbar. Aber ähnliche Verfahren können durchaus einen Dienst erweisen:

- Wie schon angedeutet, ist ein wichtiges Problem, dass man bei einer leeren Ausgabe des Wrappers nicht entscheiden kann, ob es um einen Fehler geht, oder aber die Antwort in der Tat die leere Menge ist. In solchen Fällen können Testanfragen benutzt werden, für die die Informationsquelle bestimmt eine (nicht nullwertige) Antwort geben wird. Z. B. gibt jede Internet-Suchmaschine für Anfragen wie "Internet" mehrere Tupeln zurück.

- ➢ Solche Anfragen, für die es bestimmt eine Antwort gibt, können auch während des Betriebes gesammelt werden. Es ist nämlich relativ unwahrscheinlich, dass eine Anfrage, für die es noch vor kurzer Zeit mehrere Tupeln zurückgegeben wurden, plötzlich keine Antwort mehr produziert.

- ➢ Auch wenn die Informationsquelle autonom ist, kann man in manchen Fällen erreichen, dass sie für eine gewisse Anfrage eine konkrete Antwort gibt. Beispielsweise, wenn man eine HTML-Seite mit einem unsinnigen Titel erstellt, und sicherstellt, dass die Suchmaschinen diese Seite auch finden (entweder durch Links von anderen Seiten, oder mit der "Submit a site" Funktion, die bei vielen Suchmaschinen zur Verfügung steht), dann werden die Suchmaschinen auf das Unsinnswort als Anfrage mit dem URL dieser Seite antworten.

- ➢ Der normale Regressionstest-Mechanismus kann auch insofern geändert werden, dass die Ausgabe nicht unbedingt vollständig mit der vordefinierten übereinstimmen muss, sondern nur eine gewisse Ähnlichkeit erwartet wird. Die Grundidee ist dabei, dass kleine Änderungen der Informationsquelle jederzeit passieren können, große Änderungen aber relativ unwahrscheinlich sind (die Wahrscheinlichkeit eines Fehlers ist höher). Es bleibt natürlich zu definieren, was unter Ähnlichkeit zu verstehen ist.

Wie man sieht, können diese Methoden meistens nur zusammen mit anderen Verfahren effizient eingesetzt werden. Aber in vielen Fällen sind sie sehr nützlich.

3.2.5 Validierung des Mediatorprogramms

Alle oben genannten Methoden (Validierung während der Arbeit des Wrappers, Validierung der Ausgabe des Wrappers, Einbeziehung anderer Informationsquellen, Testanfragen) können auch auf einer höheren Ebene verwendet werden: nämlich zur Validierung des ganzen Mediatorprogramms. In vielen Fällen ist es praktischer, nur das Endergebnis zu überprüfen, wie man z. B. auch bei einer Überschlagsrechnung oft nur die Plausibilität des Endergebnisses überprüft.

Wenn q Informationsquellen in die Mediatoranwendung eingebunden sind, dann verringert sich damit die Zahl der benötigten Validierungen von q auf 1. Dadurch wird die Validierung effizienter, billiger und auch robuster, weil der Validator bei der Veränderung einer Informationsquelle bzw. eines Wrappers nicht angepasst werden muss.

Oft ist es gar nicht (oder nur kaum) möglich, jeden Zugriff zu validieren, weil das dazu benötigte Zusatzwissen nicht vorhanden ist. Z. B. könnte eine Mediatoranwendung dazu benutzt werden, herauszufinden, wieviel gewisse Autos in Spanien kosten. Dafür werden zwei Informationsquellen verwendet: die eine gibt an, wieviel Pesetas der jeweilige Wagen kostet; die zweite gibt den Kurs von Peseta an, so dass man das Endergebnis in DM bekommt. Wenn kein Zusatzwissen über den Kurs oder über die Preise von Autos in Spanien aufgebracht werden kann, dann können die Ergebnisse der einzelnen Zugriffe nicht validiert werden; es ist wahrscheinlich einfacher, das Endergebnis zu validieren. Wenn sich z. B. herausstellt, dass ein neuer Opel Astra in Spanien 20DM kostet, dann gab es höchstwahrscheinlich einen Fehler. Z. B. kann die Ursache sein, dass jene Informationsquelle, die den Kurs liefert, insofern verändert wurde, dass der Wert in DM nicht mehr für 1000 Pesetas,

sondern für 1 Peseta angegeben wird, aber der Wrapper – oder das Mediatorprogramm – dividiert diese Zahl noch mit 1000.

Der größte Nachteil dieser Methode ist, dass sich nicht unbedingt jeder Fehler so drastisch im Endergebnis zeigt, so dass viele Fehler mit dieser Methode nicht feststellbar sind.

3.2.6 Vergleich der Ansätze

Alle präsentierten Validierungsverfahren eignen sich dazu, bestimmte Fehler zu entdecken, aber keines von ihnen ist perfekt. Wie gut die einzelnen Methoden in einer konkreten Anwendung einsetzbar sind, hängt natürlich sehr stark von der jeweiligen Anwendung und vor allem von den benutzten Informationsquellen und Wrappern ab. Deswegen ist ein allgemeingültiger Vergleich kaum möglich. Um möglichst viele Fehler ausfiltern zu können, werden die Methoden am besten wohl zusammen eingesetzt. Das kann allerdings sehr aufwändig sein.

In den meisten Anwendungen, in denen die Kommunikation mit den Informationsquellen textuell passiert (wie z. B. im WWW), ist wahrscheinlich eine syntaktische Analyse der Ausgabe des Wrappers die beste Lösung[1], weil sie meistens relativ einfach ist, von den Implementierungsdetails des Wrappers nicht abhängt, und viele Fehler ausfiltern kann. In manchen Fällen sind aber diese Methoden sehr ungenau. Das gilt sowohl für die Validierung einzelner Zugriffe, als auch für die Validierung des ganzen Mediatorprogramms.

Wenn diese Art von Validierung nicht ausreicht, können stärkere, aber aufwändigere Verfahren eingesetzt werden. In erster Linie sollte die Einbeziehung anderer Informationsquellen bedacht werden, weil man damit schon die meisten semantischen Fehler entdecken kann. Allerdings bedeutet diese Methode einen hohen zuzüglichen Aufwand und Overhead.

Die anderen Methoden (Testanfragen und die Validierung während der Arbeit des Wrappers) sind ziemlich speziell, aber es gibt auch Fälle, in denen sie sehr nützlich sind. Testanfragen sollten eher nur in Kombination mit anderen Verfahren benutzt werden, weil sie bei weitem nicht alle Fehler ausfiltern können. Die Validierung während der Arbeit des Wrappers kann zwar sehr effektiv sein, aber es macht die Wartung sehr schwierig, so dass sie eigentlich nur in gut begründeten Fällen benutzt werden sollte.

Zum Schluss dieses Abschnitts fasst Tabelle 3.2 die Vor- und Nachteile der hier präsentierten Validierungsverfahren zusammen.

3.3 Einbindung in die Mediatorarchitektur

In den letzten Abschnitten wurden zahlreiche Validierungsmethoden präsentiert. Es bleibt noch zu klären, wie und an welcher Stelle der Validator in die Mediatorarchitektur eingebunden werden kann. Wenn man sich Abbildung 1.1 auf Seite 3 über die Mediatorarchitektur wieder anschaut, sieht man, dass es dafür viele Möglichkeiten gibt. In den folgenden Abschnitten werden einige von diesen näher beschrieben. Man muss jedoch beachten, dass die Einbindung des Validators in die Mediatorarchitektur und die gewählte Validierungsmethode nicht unabhängig sind.

[1] Es ist ein wichtiger Punkt, dass die Kommunikation im Web textuell ist. Wegen der so entstandenen Redundanz kann ein syntaktischer Validator viele Fehler ausfiltern.

Validator	Vorteile	Nachteile
Validierung während der Arbeit des Wrappers	Alle nötigen Informationen stehen zur Verfügung	Sehr Wrapper-spezifisch
Validierung der Ausgabe des Wrappers	Ziemlich allgemein; viele Fehler können mit relativ einfachen Methoden ausgefiltert werden	Nicht alle Fehler sind aus der Ausgabe ersichtlich; Die Spezifikation der plausiblen Ausgaben ist manchmal schwierig
Einbeziehen anderer Informationsquellen	Auch semantische Symptome können zur Validierung verwendet werden	Die Validierung wird sehr aufwändig
Testanfragen	Etablierte Methode; In manchen fraglichen Fällen praktisch die einzige Lösung	In vielen Fällen nicht geeignet
Validierung des Mediatorprogramms	Kostengünstige Methode	Manche Fehler können verborgen bleiben

Tabelle 3.2: Zusammenfassung der präsentierten Klassen von Validierungsverfahren

3.3.1 Validator im Wrapper

Da der Validator die Zugriffe auf die externen Informationsquelle validieren muss, und diese im Wrapper stattfinden, ist es logisch, den Validator auch als einen Teil des Wrappers anzusehen. Wie das schon in Abschnitt 1.4 angedeutet wurde, wird dadurch praktisch die Zuverlässigkeit des Wrappers erhöht. Benutzt der Validator Informationen über den inneren Zustand des Wrappers, ist es auch naheliegend, die beiden Funktionalitäten in derselben Komponente zu verwirklichen.

Es stellt sich jedoch die Frage, was passieren soll, wenn der Validator einen Zugriff fehlerhaft gefunden hat. Es gibt viele Möglichkeiten:

- Der Validator benachrichtigt den Benutzer, dass der Zugriff fehlgeschlagen hat. Das sollte eigentlich nur dann getan werden, wenn der Mediator und die Wrapper zusammengebaut sind, denn es sollte eigentlich nur der Mediator mit dem Benutzer interagieren.
- Der Validator benachrichtigt den Mediator, dass der Zugriff fehlgeschlagen hat. Danach kann der Mediator entscheiden, was zu tun ist: ob weitere Tests notwendig sind, ob der Mediatorfachmann oder der Benutzer benachrichtigt werden soll etc. Die Benachrichtigung des Mediators kann meistens nicht mit den gewöhnlichen Kommunikationsmechanismen zwischen Wrapper und Mediator erfolgen, sondern z. B. mittels Fehlercodes (siehe Abschnitt 3.3.5).
- Eine automatische Reparatur des Wrappers wird durchgeführt (z. B. wird der Wrapper neu trainiert). Es kann natürlich auch sein, dass der Fehler somit nicht beseitigt werden kann, so dass dann doch jemand benachrichtigt werden muss.
- Der Validator benachrichtigt den Mediatorfachmann, damit er das Problem behebt.

- ➢ Der Validator lässt den Wrapper den Zugriff wiederholen.
- ➢ Der Validator führt weitere Tests durch, um auf die Fehlerursache zu kommen, oder um die Sicherheit des Urteils zu erhöhen.

Wenn der Validator für eine semantische Validierung zusätzliches Wissen benötigt, muss es entweder im Validator selbst untergebracht werden, oder eine entsprechende Informationsquelle soll in die Mediatoranwendung eingebunden werden, und über den Mediator (und einen Wrapper, der theoretisch auch validiert werden soll) mit den üblichen Methoden erreicht werden.

3.3.2 Validator im Mediator

Die Validierungsfunktionalität kann auch im Mediator untergebracht werden. Das scheint dann logisch zu sein, falls das ganze Mediatorprogramm validiert werden soll. Sonst ist es softwaretechnisch wahrscheinlich besser, den Validator vom Mediator abzukoppeln.

Wenn die Validierung im Mediator durchgeführt wird, ist die Kommunikation zwischen dem Validator und dem Benutzer oder dem Mediatorfachmann durch den Mediator gegeben. Allerdings muss dann auf Wrapper-interne Informationen verzichtet werden.

Wenn der Validator eine externe Informationsquelle braucht, kann auf sie ohne weiteres von dem Mediator aus zugegriffen werden.

3.3.3 Validator als separate Komponente

Aus softwaretechnischen Gründen ist wahrscheinlich die beste Lösung, den Validator als eine getrennte Komponente zu verwirklichen. Allerdings müssen dann verschiedene Kommunikationskanäle ausgebaut werden: erstens muss der Validator die benötigten Informationen (Wrapper-Ausgabe, Wrapper-interne Informationen, Ausgabe des Mediatorprogramms usw.) erhalten, ausserdem muss er sein Urteil weitergeben können (an den Mediator, an den Benutzer, an den Mediatorfachmann, usw.). Wenn der Validator auch externe Informationsquellen benötigt, kann er auf sie über den Mediator zugreifen.

Diese Kommunikationsmechanismen auszubauen bedeutet natürlich zusätzlichen Programmieraufwand, wenn aber die entsprechenden Kommunikationsprotokolle definiert sind, wird eine hohe Modularität gewährleistet, die die Wartung erleichtert und die Wiederverwendbarkeit erhöht.

3.3.4 Validator als Informationsquelle

Ein wichtiger Spezialfall der gerade beschriebenen Methode ist, wenn der Validator nicht nur als separate Komponente, sondern gleich als Informationsquelle eingebunden wird.

Die Idee ist, dass der Validator eine ähnliche Funktion verwirklicht, wie es bei den Informationsquellen angenommen wird (siehe Abschnitt 1.2.1 und Abbildung 1.2 auf Seite 3). Diesmal ist die Eingabe eben die Ausgabe des Wrappers, Wrapper-interne Informationen,

oder aber die Ausgabe des ganzen Mediatorprogramms. Die Ausgabe ist im einfachsten Fall das Urteil, ob der Zugriff (bzw. das Mediatorprogramm) fehlerfrei war oder nicht (weitere Ausgabemöglichkeiten siehe in Abschnitt 3.3.5). Die verwirklichte Funktion ist eben die Validierung.

Der große Vorteil dieser Methode ist, dass damit die Einbindung des Validators die Architektur nicht komplizierter macht. Der Validator ist nur eine spezielle Informationsquelle, auf die genauso zugegriffen werden kann, wie auf jede andere Informationsquelle. Der Unterschied ist nur, dass der Validator eben nicht einen Teil der Anfrage des Benutzers als Eingabe bekommt, sondern das, was die anderen Informationsquellen bzw. Wrapper ausgeben. Die Validierung kann automatisch vom Mediator, oder manuell vom Autor des Mediatorprogramms aufgerufen werden. Der Validator ist auch keine autonome Quelle im gewöhnlichen Sinne, so dass es keines speziellen Wrappers oder Validators für diese Informationsquelle bedarf.

Ein Nachteil ist, dass in diesem Fall der Validator nicht so einfach auf andere Informationsquellen zugreifen kann, denn es wird in der Mediatorarchitektur keine Interaktion unter den Informationsquellen unterstützt. Wenn er also andere Informationsquellen braucht, müssen diese auf andere Weise eingebunden werden.

3.3.5 Fehlercodes

Wie schon mehrmals angedeutet, kann der Validator seine Ausgabe in Form von Fehlercodes erzeugen und weitergeben. Ob der Code wirklich numerisch kodiert wird, oder textuell ist, kann von der jeweiligen Anwendung abhängen.

Der Fehlercode kann z. B. folgende Informationen enthalten:

- ➢ Ob der Zugriff fehlerfrei war oder nicht
- ➢ Wie sicher ist der Validator in seinem Urteil?
- ➢ Möglicher Grund bzw. mögliche Gründe des Fehlers
- ➢ Empfohlene Gegenmaßnahme
- ➢ Wie könnte noch weiter getestet werden, um die Fehlerursache herauszufinden, oder die Zuverlässigkeit des Urteils zu erhöhen?
- ➢ Wieviel würden diese weitere Tests kosten (Zeit, Geld, Ressourcen usw.)?

3.3.6 Diagnostikfunktion

Bis jetzt ging es hauptsächlich um Online-Validierung, d. h. die Validierung erfolgte direkt nach dem Zugriff auf die externe Informationsquelle. Dieselben Methoden können natürlich auch in einem Offline-Szenario benutzt werden, indem die Validierung getrennt, in Phasen, in denen Systemressourcen zur Verfügung stehen, wie z. B. in der Nacht, erfolgt. Die Validierungsmethode ist eigentlich dieselbe, nur die Anbindung ist etwas anders: tagsüber werden die Testdaten (entweder im Wrapper oder im Mediator, oder aber auch an beiden

Stellen) gesammelt, um dann in der Nacht die Validierung durchzuführen. (Es kann natürlich auch einen vordefinierten Testdatensatz geben, nicht nur die am Tag gesammelten Daten.) Das Validierungsverfahren wird also nicht nach jedem Zugriff, sondern nur einmal am Tag aufgerufen. Das ist natürlich dann sinnvoll, wenn die Validierung länger dauert, als was man bei der normalen Benutzung tolerieren kann.

Wenn dieser Offline-Validator im Wrapper eingebaut ist, kann es so angesehen werden, dass der Wrapper praktisch eine Diagnostikfunktion zur Verfügung stellt: man kann abfragen, ob der Wrapper noch ordnungsgemäß funktioniert. In ähnlicher Weise bietet ein Offline-Validator der ganzen Mediatoranwendung eine Diagnostikfunktion, mit der man abfragen kann, ob die Anwendung noch ordnungsgemäß funktioniert.

3.3.7 Mehrstufige Validierung

Es wurde schon erwähnt, dass auch mehrere von den verschiedenen Validierungsmethoden von Abschnitt 3.2 gleichzeitig einsetzbar sind, und dass das in vielen Fällen auch sinnvoll ist. In ähnlicher Weise kann es auch sein, dass die verschiedenen Validierungsverfahren nicht in einer einzigen Validatorkomponente verwirklicht sind, sondern mehrere Validatorstufen formen.

Es ist z. B. möglich, dass jeder Wrapper mit einem Validator verknüpft ist, der bei jedem Zugriff einen schnellen, oberflächlichen Test durchführt, und mit einem anderen, der einen langsameren, aber gründlicheren Test als Diagnostikfunktion zur Verfügung stellt. Daneben kann noch ein weiterer Validator, der das ganze Mediatorprogramm validiert, als externe Informationsquelle eingebunden sein usw.

Kapitel 4

Fallstudie

Dieses Kapitel demonstriert anhand des Metasearch-Beispiels (siehe Abschnitt 1.3.2), wie die in Kapitel 3 präsentierten Validierungsmethoden in die Praxis umgesetzt werden können. Programmteile, sowie qualitative und quantitative Ergebnisse werden präsentiert und diskutiert.

Die Auswertung anhand des Metasearch-Beispiels ist in mehrerer Hinsicht sehr wichtig. Erstens, wie schon früher angedeutet, ist Metasearch zwar eine sehr einfache Anwendung, aber die meisten Probleme der Mediator-Anwendungen kommen dabei schon zum Vorschein. Zweitens, mit der immer wachsenden Bedeutung des Internet wird auch die Anzahl der über das Web eingebundenen Informationsquellen immer größer; Metasearch kann auch als der Prototyp von rein Internet-basierten Mediatoranwendungen angesehen werden. Ausserdem sind in der Fachliteratur schon einige Ergebnisse über die Validierung von ähnlichen Web-basierten Anwendungen vorhanden, was einen Vergleich ermöglicht (siehe Abschnitt 5.1).

Als erstes wird in Abschnitt 4.1 die konkrete Aufgabenstellung der Fallstudie geklärt und präzisiert. In Abschnitt 4.2 werden Details der Implementierung präsentiert, sowie einige interessantere Programmteile. Abschnitt 4.3 diskutiert die Ergebnisse, die mit dem implementierten System erzielt wurden. Schließlich beschreibt Abschnitt 4.4 einige weitere Versuche und Überlegungen, die die Gültigkeit der Ergebnisse unterstützen.

4.1 Konkrete Aufgabenstellung

Die wichtigste Aufgabe der Fallstudie war, die praktische Anwendbarkeit der in Kapitel 3 vorgeschlagenen Methoden zu testen. Die Erfahrungen sollten es ermöglichen, die Methoden miteinander und mit den Methoden aus der Fachliteratur zu vergleichen.

Es war kein Ziel der Arbeit, einen perfekten, überall einsetzbaren Validator zu konstruieren, der direkt in KOMET eingebaut werden kann. Stattdessen wurde ein vereinfachter Prototyp von Metasearch erstellt. Um das einfache Testen der Validierungsverfahren zu ermöglichen, enthält dieser Prototyp nur die aus der Sicht der Validierung wirklich wichtigen Komponenten von Metasearch. Also enthält er keine Inferenzmaschine, und wird auch nicht deklarativ programmiert, damit die Möglichkeiten der Validierung völlig unbeschränkt sind.

Folgende Funktionalität wurde im Prototyp beibehalten:

- ➢ Ins Programm wurden verschiedene Internet-Suchmaschinen mittels spezieller Wrapper eingebunden
- ➢ Der Benutzer (es kann auch ein anderes Programm sein) kann Anfragen angeben
- ➢ Der Benutzer kann eine Suchmaschine auswählen
- ➢ Die Anfrage des Benutzers wird an die gewählte Suchmaschine gestellt
- ➢ Die Antwort der Suchmaschine wird im Wrapper bearbeitet, und die URLs, Titel und Extrakte der gefundenen Seiten werden angezeigt

Als zusätzliche Funktion wird die Antwort der Informationsquelle mit verschiedenen Verfahren validiert.

4.2 Implementierung

Für die Implementierung des Prototyps wurde die Programmiersprache Perl [39] gewählt, denn

- ➢ sie unterstützt sehr stark die Verwendung von regulären Ausdrücken, die sowohl zur Extraktion von Informationen aus HTML-Seiten, als auch zur Validierung von Text verwendet werden können;
- ➢ das Herunterladen einer Web-Seite ist in Perl mittels des LWP[1] Moduls – auch über Proxy und Firewall – sehr einfach;
- ➢ sie eignet sich hervorragend zur schnellen Implementierung von Prototyplösungen (engl.: *rapid prototyping*), weil auch komplexere Perl-Programme mit relativ niedrigem Programmieraufwand erstellt werden können. Sie müssen auch nicht nach jeder Änderung neu kompiliert werden.

Die Altervative wäre die Programmiersprache Java [1] mit dem Modul `gnu.regexp`[2] für die Unterstützung von regulären Ausdrücken gewesen. Die ersten Versuche sind auch in Java durchgeführt worden. Aber es hat sich schnell gezeigt, dass Java die oben aufgeführten Eigenschaften nicht besitzt.

Für manche Teilaufgaben wurden auch Unix Shell-Scripts[3] verwendet. Eine grafische Oberfläche wurde mit Hilfe von Tk erstellt. (Die grafische Bibliothek Tk wurde ursprünglich für die Programmiersprache Tcl[4] entwickelt, wurde aber inzwischen auch an Perl angebunden[5].)

[1] `http://www.cpan.org/modules/by-module/LWP/`
[2] `http://www.cacas.org/~wes/java/`
[3] Siehe z. B. `http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Unix/CShellII.html`
[4] `http://dev.scriptics.com/software/tcltk/`
[5] `http://w4.lns.cornell.edu/~pvhp/ptk/doc/`

```
1  #!/usr/local/bin/perl -w
2
3  use LWP::Simple;
4
5  die "Usage: $0 query_string\n" unless $#ARGV>=0;
6
7  $urlbase="http://www.altavista.com/cgi_bin/query?q=";
8  $"="+";
9  $query="@ARGV";
10
11 $input=get("$urlbase$query");
12 die "Can't fetch html page: $urlbase$query\n" unless defined
       $input;
13
14 $input=~s/[\n\r]//gms;
15
16 while ($input=~m#<dl><dt><b class=txt2>\d+\.</b><b><a href=
       "(.+?)">(.+?)</a></b>#gi) {
17   push @results, [($1,$2)];
18 }
19
20 for $i (0..$#results) {
21   printf "%d.\nURL:\n$results[$i][0]\nTITLE:\n$results[$i][1]\n"
         ,$i+1;
22 }
```

Programmliste 4.1: Ein in Perl geschriebener, einfacher Wrapper für AltaVista

In erster Linie wurde ein Wrapper für AltaVista implementiert; siehe Programmliste 4.1. Aus diesem kleinen Programm ist schon ersichtlich, dass man in Perl sehr kompakt, nur auf die wichtigen Elemente konzentrierend programmieren kann. Die Funktionsweise dieses Programms wird im Folgenden kurz erläutert.

Zeile 1 dient nur dazu, das Programm einfach starten zu können. In Zeile 3 wird das Modul `LWP::Simple` geladen, das den einfachen Zugriff auf Webseiten ermöglicht. In Zeile 5 wird überprüft, ob das Programm mit mindestens einem Argument – nämlich mit der Anfrage – gestartet wurde; wenn nicht, wird eine Fehlermeldung ausgegeben, und das Programm wird unterbrochen. In Zeile 7 wird das Basis-URL von AltaVista eingestellt. In Zeilen 8-9 werden die Argumente des Programms mit Pluszeichen zusammengefügt. D. h. wenn das Programm mit den Argumenten "KOMET" und "knowledge" aufgerufen wird, erhält die Variable `$query` den Wert "KOMET+knowledge", was eben dem von AltaVista erwarteten Format entspricht. In Zeile 11 wird die gewünschte Seite von AltaVista mit der von LWP bereitgestellten Methode `get` heruntergeladen. Sollte das nicht gelingen, wird in Zeile 12 eine Fehlermeldung ausgeschrieben, und das Programm wird unterbrochen. In Zeile 14 werden Zeilenumbrüche aus der HTML-Seite entfernt. In Zeilen 16-18 erfolgt die eigentliche Informationsextraktion: der Text der Seite wird iterativ mit einem regulären Ausdruck durchsucht, und die gefundenen URLs und Titel werden in eine Liste Namens `@results` gesammelt. In Zeilen 20-22 werden die Elemente dieser Liste ausgeschrieben.

Dieses einfache Programm diente als Grundlage für die weiteren Experimente, wurde jedoch entsprechend den verschiedenen Ansprüchen in mehreren Richtungen weiterentwickelt. Es wurden im Wesentlichen zwei verschiedene Anwendungen erstellt: ein interaktiver Metasearch-Prototyp und ein stapelverarbeitetes Testprogramm. Ausserdem wurden verschiedene Validierungsmethoden implementiert, die in beiden Anwendungen benutzt werden. Diese Entwicklungen werden in den nächsten Abschnitten näher beschrieben.

4.2.1 Interaktiver Metasearch-Prototyp

Mit der grafischen Bibliothek Tk wurde eine einfache Benutzeroberfläche erstellt, die es für den Benutzer ermöglicht, verschiedene Anfragen an verschiedene Suchmaschinen zu stellen, und die aus der Antwort extrahierten Informationen anzeigt (siehe Abbildung 4.1). Einfachheitshalber werden in dieser Anwendung nur URLs und Titel extrahiert und angezeigt, die Extrakte nicht.

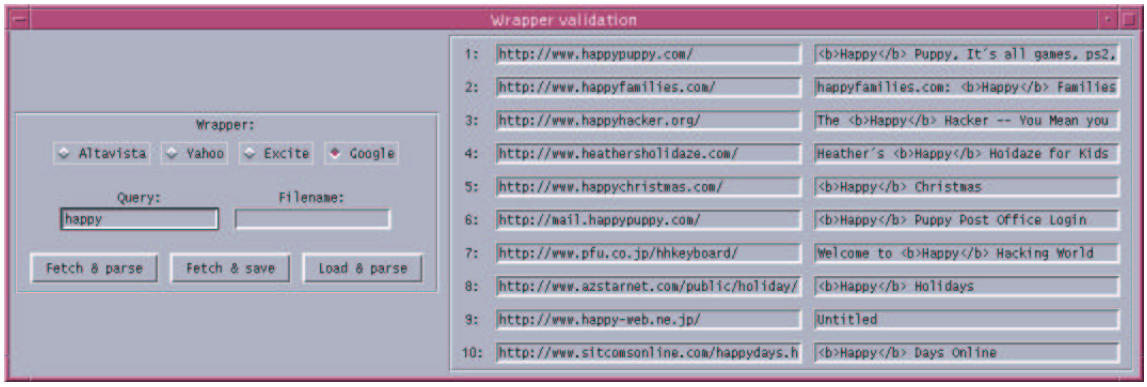

Abbildung 4.1: Die Benutzeroberfläche des in Perl/Tk geschriebenen interaktiven Metasearch-Prototyps

Um das Experimentieren zu ermöglichen, bietet diese Anwendung drei verschiedene Funktionen:

1. Fetch & parse: die durch Suchmaschine und Anfrage bestimmte Seite wird heruntergeladen, und die Informationen werden extrahiert und angezeigt
2. Fetch & save: die Seite wird heruntergeladen, und im HTML-Format in der angegebenen Datei gespeichert; Informationen werden nicht extrahiert
3. Load & parse: die angegebene Datei wird eingelesen, und es wird versucht, die Informationen mit dem Wrapper für die angegebene Suchmaschine zu extrahieren; die extrahierten Informationen werden angezeigt

Die erste Funktion entspricht der "normalen" Nutzung; die beiden anderen Funktionen waren beim Experimentieren, vor allem bei der Analyse von Änderungen der Suchmaschinenausgabe und bei der Analyse derer Auswirkung nützlich.

Der Quellcode dieser Anwendung wurde in zwei Module geteilt: `main` und `Wrapper`. Das hat softwaretechnische Gründe: das Modul `Wrapper` konnte auch in der anderen Anwendung wiederverwendet werden. In diesem Modul wird die Klasse `Wrapper` definiert (obwohl in Perl der Unterschied zwischen einem Modul und einer Klasse nicht allzu groß ist). Instanzen dieser Klasse entsprechen den einzelnen Wrappern, die – durch Angabe des Basis-URLs und eines regulären Ausdrucks zur Extraktion der Informationen – auf eine bestimmte Suchmaschine spezialisiert sind. Ausserdem definiert diese Klasse bestimmte Primitive, wie z. B. `load_html` oder `parse`, aus denen in `main` die obigen Funktionen zusammengestellt sind.

```
1  package Wrapper;
2
3  use LWP::Simple;
4  use Validator;
5
6  sub init_results {
7    for $i (0..9) {
8      $results[$i][0]="";
9      $results[$i][1]="";
10   }
11 }
12
13 sub create {
14   my ($class,$urlbase,$regexp)=@_;
15   my $self={urlbase => $urlbase , regexp => $regexp};
16   bless $self,$class;
17   return $self;
18 }
19
20 sub fetch {
21   my ($self,$query)=@_;
22   my $urlbase=$self->{urlbase};
23   my $html=get("$urlbase$query");
24   die "Can't fetch html page: $urlbase$query\n" unless defined
       $html;
25   return $html;
26 }
27
28 sub parse {
29   my ($self,$html)=@_;
30   $html=~s/[\r\n]//gms;
31   my $regexp=$self->{regexp};
32   my $i=0;
33   init_results;
34   my $expr='
35     while ($i<10 && $html=~m#'."$regexp".'#gi) {
36       $results[$i][0]=$1;$results[$i][1]=$2;$i++;
37     }';
38   eval $expr;
39   $number_of_results=$i;
40   Validator::validate($number_of_results,@results);
41 }
42
43 sub save_html {
44   my ($html,$filename)=@_;
45   open(OUT,"> $filename");
46   print OUT "$html";
47   close(OUT);
48 }
49
50 sub load_html {
51   my $filename=shift;
52   my $html=`cat $filename`;
53   return $html;
54 }
55
56 sub dump_results {
57   my @results=@_;
58   my $result="";
59   for my $i (0 .. $#results) {
60     my $num=$i+1;
61     $result .= "$num.\nURL:\n$results[$i][0]\nTITLE:\n$results[
         $i][1]\n";
62   }
63   return $result;
64 }
65
66 init_results;
67
```

```
68 1;
```

Programmliste 4.2: Quellcode von Wrapper.pm

Programmliste 4.2 zeigt den Quellcode von `Wrapper.pm`. Das Modul `Wrapper` benutzt auch schon das Modul `Validator` (Zeile 4 bzw. 40), weil die extrahierten Tupeln gleich auch validiert werden. Darauf wird im nächsten Abschnitt näher eingegangen. Die Funktion `init_results` (Zeile 6-11) dient dazu, die 10-elementige Liste `@results` zu initialisieren. Diese Funktion wird am Anfang des Programmablaufs (Zeile 66) und vor jeder Durchführung von `parse` (Zeile 33) aufgerufen. Die Funktion `create` (Zeile 13-18) ist der Konstruktor der Wrapper. Mit der Funktion `fetch` (Zeile 20-26) wird eine Webseite heruntergeladen. Die Funktion `parse` (Zeile 28-41) extrahiert die Informationen aus einer HTML-Zeichenkette. Mit `save_html` bzw. `load_html` (Zeile 43-48 und 50-54) kann die HTML-Datei gespeichert und geladen werden. `dump_results` (Zeile 56-64) schreibt die Elemente der Liste `@results` in eine Variable aus.

Im Modul `main` wird nur die grafische Oberfläche erstellt, und die Wrapper werden konstruiert. Die weitere Funktionalität wird durch die Anbindung von Aktionen an die Elemente der Benutzeroberfläche geregelt.

```
1  #!/usr/local/bin/perl -w
2
3  use Wrapper;
4  use Tk;
5
6  $main_window=new MainWindow(-title => "Wrapper validation");
7  $frame1=$main_window->Frame(-relief=>"groove",-borderwidth=>"2"
       )->pack(-side=>"left",-padx=>5,-pady=>5);
8  $frame1->Label(-text=>"Wrapper:")->pack();
9  $frame11=$frame1->Frame(-borderwidth=>"2")->pack(-padx=>5,-pady
       =>5);
10 $frame11->Radiobutton(-value=>"altavista",-text=>"Altavista",-
       command=>sub{$selectedWrapper=$altavista})->pack(-side=>"left
       ",-padx=>5)->invoke();
11 $frame11->Radiobutton(-value=>"yahoo",-text=>"Yahoo",-command=>
       sub{$selectedWrapper=$yahoo})->pack(-side=>"left",-padx=>5);
12 $frame11->Radiobutton(-value=>"excite",-text=>"Excite",-command
       =>sub{$selectedWrapper=$excite})->pack(-side=>"left",-padx
       =>5);
13 $frame11->Radiobutton(-value=>"google",-text=>"Google",-command
       =>sub{$selectedWrapper=$google})->pack(-side=>"left",-padx
       =>5);
14
15 $frame12=$frame1->Frame(-borderwidth=>"2")->pack(-padx=>5,-pady
       =>5);
16 $frame121=$frame12->Frame(-borderwidth=>"2")->pack(-side=>"left"
       ,-padx=>5,-pady=>5);
17 $frame121->Label(-text=>"Query:")->pack();
18 $frame121->Entry(-textvariable=>\$query)->pack();
19 $frame122=$frame12->Frame(-borderwidth=>"2")->pack(-side=>"left"
       ,-padx=>5,-pady=>5);
20 $frame122->Label(-text=>"Filename:")->pack();
21 $frame122->Entry(-textvariable=>\$filename)->pack();
22
23 $frame13=$frame1->Frame(-borderwidth=>"2")->pack(-padx=>5,-pady
       =>5);
24 $frame13->Button(-text=>"Fetch & parse",-command=>sub{
       $selectedWrapper->parse($selectedWrapper->fetch($query))})->
       pack(-side=>"left",-padx=>5);
```

```
25 $frame13->Button(-text=>"Fetch & save",-command=>sub{Wrapper::
       save_html($selectedWrapper->fetch($query),$filename)})->pack
       (-side=>"left",-padx=>5);
26 $frame13->Button(-text=>"Load & parse",-command=>sub{
       $selectedWrapper->parse(Wrapper::load_html($filename))})->
       pack(-side=>"left",-padx=>5);
27
28 $frame2=$main_window->Frame(-relief=>"groove",-borderwidth=>"2"
       )->pack(-side=>"left",-padx=>5,-pady=>5);
29
30 for my $i (0..$#Wrapper::results) {
31   my $framex=$frame2->Frame()->pack(-padx=>5,-pady=>5);
32   my $num=$i+1;
33   $framex->Label(-text=>"$num:",-width=>3)->pack(-side=>"left",-
         padx=>2);
34   $framex->Entry(-textvariable=>\$Wrapper::results[$i][0],-width
         =>40)->pack(-side=>"left",-padx=>5);
35   $framex->Entry(-textvariable=>\$Wrapper::results[$i][1],-width
         =>40)->pack(-side=>"left",-padx=>5);
36 }
37
38 $urlbase="http://www.altavista.com/sites/search/web?q=";
39 $regexp='<b><a href="/r\?ck_sm=.*?&ref=.*?&r=(.+?)" .+?>(.+?)</a
       ></b>';
40 $altavista=Wrapper->create($urlbase,$regexp);
41 $urlbase="http://search.yahoo.com/bin/search?h=s&p=";
42 $regexp='<li><a href=".+?\*(.+?)">(.+?)</a>';
43 $yahoo=Wrapper->create($urlbase,$regexp);
44 $urlbase="http://search.excite.com/search.gw?search=";
45 $regexp=';pos=\d+;(.+?) onMouseOver=.+?>(.+?)</a>';
46 $excite=Wrapper->create($urlbase,$regexp);
47 $urlbase="http://www.google.com/search?q=";
48 $regexp='<p><a href=(.+?)>(.+?)</a>';
49 $google=Wrapper->create($urlbase,$regexp);
50
51 $selectedWrapper=$altavista;
52
53 MainLoop;
```

Programmliste 4.3: Quellcode von `main.pl`

Der Quellcode von `main` ist in Programmliste 4.3 dargestellt. Damit der Code verständlicher ist, wird in Abbildung 4.2 die Anordnung der Frames gezeigt.

Mit der Hilfe dieses interaktiven Programms wurden viele Experimente durchgeführt. Die wichtigsten Beobachtungen sind in Abschnitt 4.3.1 beschrieben.

4.2.2 Validierungsmethoden

Parallel zu den Experimenten mit dem interaktiven Metasearch-Prototyp wurden auch mehrere Validierungsverfahren implementiert. Wie schon bei der Erläuterung von `Wrapper.pm` angesprochen, werden die Informationen gleich nach ihrer Extraktion validiert. Dieselben Validierungsverfahren wurden später auch im stapelverarbeiteten Testprogramm verwendet.

Die folgenden Validierungsmethoden wurden implementiert:

- Eine syntaktische Validierungsmethode für die URLs. Es wäre theoretisch möglich

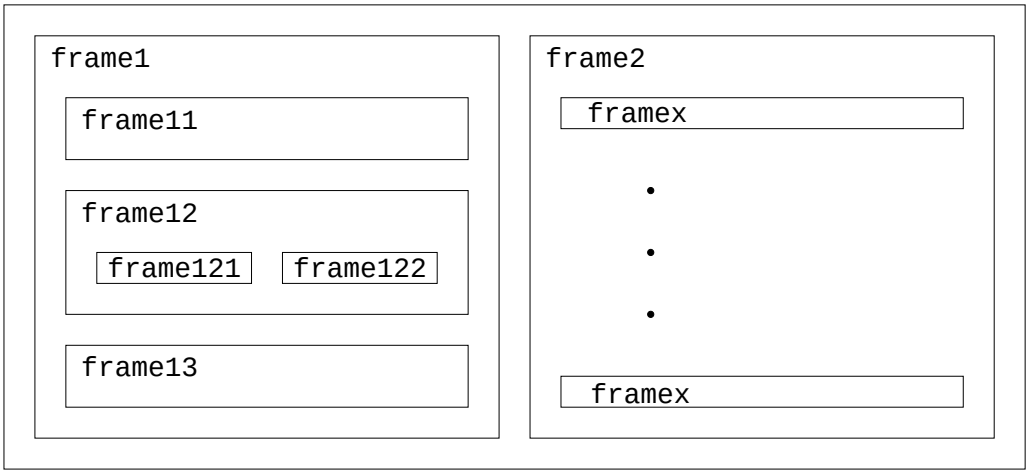

Abbildung 4.2: Anordnung der Benutzeroberfläche

gewesen, einen Zerteiler für die Sprache der URLs[6] zu konstruieren. Stattdessen wurde aber ein regulärer Ausdruck konstruiert, der zwar nicht perfekt ist, aber in der Praxis meistens gut funktioniert, und viel kompakter und einfacher zu erstellen ist. Als weiterer Vorteil kann man durch die Präzisierung bzw. Lockerung des regulären Ausdrucks seine Spezifizität und Sensitivität fein einstellen. In den Tests wurde folgender Ausdruck benutzt:

```
^((http\://)|(ftp\://))?[-\+\w]+(\.[-\+\w]+)+(\:\d+)?\
(/\~[-\+\.%\w]+)?(/[-\+\.\,\:%\w]+)*/?(\?([-\+\.%\w]+\
=[-\+\.%\w/]+&)*[-\+\.%\w]+=[-\+\.%\w/]+)?$
```

➢ Eine syntaktische Validierungsmethode für die Titel. Auch da wurde eine sehr einfache, aber in der Praxis gut funktionierende Methode gewählt: der Titel einer HTML-Seite kann alles sein, ist jedoch meistens nicht allzu lang. In der offiziellen Spezifikation von HTML[7] steht zwar nicht, dass die Länge des Titels begrenzt wäre, aber der Titel wird normalerweise in der Titelleiste des Browser-Fensters angezeigt, so dass ein sehr langer Titel keinen Sinn macht, weil es nicht angezeigt werden kann; deshalb werden lange Titel auch nicht benutzt. Tim Berners-Lee[8] empfiehlt [3], dass der Titel maximal 64 Zeichen lang sein soll. Im Validator wird überprüft, ob der Titel weniger als 100 Zeichen lang ist.
Andererseits wird in der Dokumentation von HTML festgelegt, dass der Titel keine HTML-Formatierungen (also keine Tags) enthalten darf. Also könnte der Validator das eventuell auch überprüfen. Das Problem ist, dass der von der Suchmaschine angezeigte Titel schon durchaus HTML-Tags enthalten kann. Z. B. zeigen einige Suchmaschinen die Wörter der ursprünglichen Anfrage mit fetten oder kursiven Buchstaben an, wenn sie im Titel oder im Extrakt vorkommen.

➢ Eine syntaktische Validierungsmethode für den Extrakt. Auch da wurde eine ähnli-

[6] `http://www.cs.ucl.ac.uk/staff/jon/book/node166.html`
[7] `http://www.w3.org/TR/html401`
[8] Vater des World Wide Web und Direktor des World Wide Web Consortium, siehe `http://www.w3.org/People/Berners-Lee`

che, einfache Methode gewählt: der Extrakt ist kurz – genauer gesagt, er muss kürzer sein, als 300 Zeichen. Diese Grenze konnte natürlich nicht der HTML-Spezifikation entnommen werden, denn Syntax und Semantik des Extraktes wird von den einzelnen Suchmaschinen definiert. Diese Definition ist aber nicht öffentlich zugänglich, also musste dieser Grenzwert empirisch ermittelt werden. Es wird auch verlangt, dass der Extrakt aus mindestens einem Zeichen bestehe.

➢ Ein zeitaufwändigerer, semantischer Test der Wrapper-Ausgabe. Dabei wurde als zusätzliche Informationsquelle das Internet selbst einbezogen. Für einen semantischen Test muss natürlich überlegt werden, was eben die Semantik der extrahierten Tupeln ist. In dem Fall einer Suchmaschine kann die Semantik etwa so ausgedrückt werden: "Die URLs zeigen zu Seiten, die wahrscheinlich noch existieren, zumindest schon mal existiert haben. Diese Seiten haben wahrscheinlich eben die angegebenen Titel, zumindest hatten sie schon mal diese Titel. Aus diesen Seiten wurden die entsprechenden Extrakte extrahiert. Die Seiten enthalten wahrscheinlich Informationen, die mit der ursprünglichen Anfrage in enger Verbindung stehen, wahrscheinlich kommen sogar die Wörter der Anfrage in den Seiten vor."
In einem semantischen Test müssen eben diese Eigenschaften überprüft werden. Dafür kann wirklich das Internet selbst als Informationsquelle dienen: man kann überprüfen, ob die angegebenen URLs tatsächlich zu existierenden Seiten zeigen, ob diese Seiten wirklich die angegebenen Titel haben und die Wörter des Extraktes und die der Anfrage enthalten. Allerdings gibt es dabei – wie es auch aus der obigen Beschreibung der Semantik klar ist – eine große Unsicherheit: es ist nicht sicher, dass die Seite immer noch existiert und immer noch dieselben Informationen enthält.
Um diese Unsicherheit zu minimieren, wurde im Validator der Test insofern vereinfacht, dass nur überprüft wird, ob das URL tatsächlich zu einer existierender Seite zeigt. Die anderen Teile des Tests wurden weggelassen, weil sie die Ergebnisse des Tests nicht verbesserten.

Im Validator (`Validator.pm`) wurden diese vier Validierungsverfahren implementiert; die zugehörigen Funktionen sind: `is_URL_OK`, `is_Title_OK`, `is_Excerpt_OK`, `is_Tuple_OK`.

```
1  package Validator;
2
3  use strict;
4
5
6  sub validate_searchengine($@) {
7    my $tmp_dir=shift;
8    my @results=@_;
9    my $number_of_invalid_URLs=0;
10   my $number_of_invalid_Titles=0;
11   my $number_of_invalid_Excerpts=0;
12   my $number_of_invalid_Tuples=0;
13   system "rm $tmp_dir/*";
14   print "Starting timeout_wget's...\n";
15   for my $i (0..$#results) {
16     my $url=$results[$i]->{"URL"};
17     my @command=("timeout_wget_starter.sh","20","$tmp_dir/tmp$i.
         html","$url");
18     system(@command);
19   }
20   print "Sleeping for 30 secs...\n";
21   sleep 30;
22   print "Ckecking results...\n";
```

```
23    for my $i (0..$#results) {
24
25      my $url=$results[$i]->{URL};
26      main::print_log("$url\n");
27      if (is_URL_OK($url)) {
28        main::print_log("Correct URL\n");
29      } else {
30        main::print_log("Erroneous URL\n");
31        $number_of_invalid_URLs++;
32      }
33
34      my $title=$results[$i]->{Title};
35      main::print_log("$title\n");
36      if (is_Title_OK($title)) {
37        main::print_log("Correct Title\n");
38      } else {
39        main::print_log("Erroneous Title\n");
40        $number_of_invalid_Titles++;
41      }
42
43      my $excerpt=$results[$i]->{Excerpt};
44      main::print_log("$excerpt\n");
45      if (is_Excerpt_OK($excerpt)) {
46        main::print_log("Correct Excerpt\n");
47      } else {
48        main::print_log("Erroneous Excerpt\n");
49        $number_of_invalid_Excerpts++;
50      }
51
52      my $tuple=$results[$i];
53      if (is_Tuple_OK($tmp_dir,$i)) {
54        main::print_log("Correct Tuple\n");
55      } else {
56        main::print_log("Erroneous Tuple\n");
57        $number_of_invalid_Tuples++;
58      }
59    }
60
61    main::print_result("@{[$#results+1]}\t"."
          $number_of_invalid_URLs\t"."$number_of_invalid_Titles\t"."
          $number_of_invalid_Excerpts\t"."$number_of_invalid_Tuples\n
          ");
62  }
63
64  sub is_URL_OK($) {
65    my $url=shift;
66    $url=~m#^((http\://)|(ftp\://))?[-\+\w]+(\.[-\+\w]+)+(\:\d
          +)?(/\~[-\+\.%\w]+)?(/[-\+\.\,\:%\w]+)*/?(\?([-\+\.%\w
          ]+=[-\+\.%\w/]+&)*[-\+\.%\w]+=[-\+\.%\w/]+)?$#i;
67  }
68
69  sub is_Title_OK($) {
70    my $title=shift;
71    length($title)<100;
72  }
73
74  sub is_Excerpt_OK($) {
75    my $excerpt=shift;
76    length($excerpt)<300 && length($excerpt)>0;
77  }
78
79  sub is_Tuple_OK($$) {
80    my $tmp_dir=shift;
81    my $index=shift;
82    my $page=`cat $tmp_dir/tmp$index.html`;
83    defined $page && $page ne "";
84  }
85
```

```
86 1;
```

Programmliste 4.4: Quellcode von Validator.pm

Der Quellcode von `Validator.pm` ist in Programmliste 4.4 dargestellt. Neben den schon erwähnten einfachen Funktionen beinhaltet dieses Modul noch `validate_searchengine`. Sie wird von ausserhalb des Moduls aufgerufen, und sie ruft dann die anderen Funktionen auf. Dabei wird die Anzahl von positiven Testergebnissen, also von „fehlerhaft“ bezeichneten URLs etc. gezählt. Die Arbeitsweise ist ziemlich unkompliziert, bis auf die Durchführung des semantischen Tests. Diese wird durch zwei Faktoren komplizierter:

```
1 #!/usr/bin/sh
2
3 wget -O $2 $3 >/dev/null &
4 sleep $1
5 kill $! 2>/dev/null 1>/dev/null
```

Programmliste 4.5: Quellcode von timeout_wget.sh

- ➢ Konnte das URL nicht gefunden werden, kann es noch lange dauern, bis `get` (aus `LWP::Simple`) aufgibt. Damit der Test nicht zu zeitaufwändig wird, soll also dieser Vorgang nach einer gewissen Zeit unterbrochen werden. Es ist zwar möglich, bei LWP einen Timeout-Wert anzugeben, aber das funktioniert in vielen Fällen nicht[9]. Also wurde das von Hand, mit Hilfe des Unix-Programmes `wget` implementiert. Diese Funktionalität wird im Script `timeout_wget.sh` (siehe Programmliste 4.5) verwirklicht.

- ➢ Wenn 10 Tupeln extrahiert werden, dann müssen auch 10 URLs überprüft werden. Das sollte natürlich parallel gemacht werden. Deswegen wird das Script `timeout_wget.sh` 10-mal gestartet; jedes Script startet einen `wget`-Prozess im Hintergrund, und wartet eine bestimmte Zeit (20 Sekunden), danach wird der Prozess – wenn er noch läuft – abgebrochen.

Jedes `wget` versucht, die jeweilige Seite in eine temporäre Datei (`$tmp_dir/tmp$i.html`) herunterzuladen. Nach einer gewissen Zeit (im Programm sind das 30 Sekunden), wenn alle `wget`-Prozesse fertig sind, bzw. abgebrochen wurden, werden die heruntergeladenen Seiten im Unterprogramm `is_Tuple_OK` überprüft. Wie schon angedeutet, wurde diese Untersuchung lediglich darauf begrenzt, ob die Seite heruntergeladen werden konnte.

4.2.3 Stapelverarbeitetes Testprogramm

Um die praktische Anwendbarkeit der oben aufgeführten Verfahren zu demonstrieren, bedarf es einer umfangreicheren Evaluierung. Da die Veränderungen der Suchmaschinen nicht sehr oft stattfinden, reichte die zur Verfügung stehende Zeit nicht aus, genügend Daten zu sammeln. Zum Glück hat Dr. Kushmerick die von ihm im Zeitraum Mai – Oktober 1998

[9] Siehe z. B. `http://www.ics.uci.edu/pub/websoft/libwww-perl/archive/1999h2/0260.html` oder `http://archive.develooper.com/libwww@perl.org/msg01318.html`

gesammelten Daten [21, 23] zur Verfügung gestellt. Dabei hat er an 27 Sites 6 Monate lang verschiedene Anfragen gestellt, und die Ausgaben gespeichert.

Für die Evaluierung vom Meteasearch-Prototyp wurden davon die Ausgaben von AltaVista, Lycos[10] und MetaCrawler auf die Anfrage "happy" verwendet. Das sind insgesamt etwa 200 HTML-Seiten. In dem oben genannten Zeitraum gab es zwei Veränderungen bei AltaVista, eine bei Lycos und zwei bei MetaCrawler. Also wurden drei Wrapper für AltaVista, zwei für Lycos und drei für MetaCrawler erstellt. Jede Seite wurde sowohl mit dem zugehörigen korrekten Wrapper bearbeitet, als auch mit dem/den anderen für dieselbe Suchmaschine: somit wurde getestet, wie die Wrapper-Ausgabe bei Änderungen der Suchmaschine aussieht, und wie die einzelnen Validierungsmethoden damit zurecht kommen. Zu beachten war auch, dass die zweite Veränderung von MetaCrawler nur die erste rückgängig gemacht macht, so dass Seiten vor der ersten Änderung fehlerfrei mit dem dritten Wrapper bearbeitet werden konnten. Also wurden diese Testfälle nicht bei der zweiten Testgruppe benutzt. Insgesamt ergaben sich etwa 440 Testfälle.

Es gab einige kleinere Veränderungen in den Modulen `Wrapper` und `Validator`, z. B. wurde `Wrapper` in `SearchEngineWrapper` umbenannt. Viel wichtiger ist aber, dass statt `main` drei Module geschrieben wurden: `main_altavista`, `main_lycos` und `main_metacrawler`. Statt der Erstellung der Benutzeroberfläche war ihre Aufgabe eben die Steuerung des Testvorgangs.

```
1  #!/usr/local/bin/perl -w
2
3  use SearchEngineWrapper;
4
5
6  $result_file="results_altavista.txt";
7  $log_file="log_altavista.txt";
8
9  sub print_result($) {
10   open(OUT,">>$result_file");
11   print OUT $_[0];
12   close OUT;
13 }
14
15 sub print_log($) {
16   open(OUT,">>$log_file");
17   print OUT $_[0];
18   close OUT;
19 }
20
21 sub print_log_result($) {
22   print_result $_[0];
23   print_log $_[0];
24 }
25
26
27 $urlbase="http://www.altavista.com/sites/search/web?q=";
28
29 $regexp1='\. </b><a href="(.+?)"><b>(.+?)</b></a><dd>(.*?)<i>';
30 $regexp2='\. </b><a href="(.+?)"><b>(.+?)</b></a>.*?</font><br
       >(.*?)<font size=-2>';
31 $regexp3='\. </b><a href="(.+?)"><b>(.+?)</b></a><dd>(.*?)<b>URL
       ';
32
33 $altavista1=SearchEngineWrapper->create($urlbase,$regexp1,"
       altavista1");
```

[10] www.lycos.com

```
34 $altavista2=SearchEngineWrapper->create($urlbase,$regexp2,"
       altavista2");
35 $altavista3=SearchEngineWrapper->create($urlbase,$regexp3,"
       altavista3");
36
37 $task[0][0]="altavista_pages/pages1";$task[0][1]=$altavista1;
38 $task[1][0]="altavista_pages/pages2";$task[1][1]=$altavista2;
39 $task[2][0]="altavista_pages/pages3";$task[2][1]=$altavista3;
40
41 $task[3][0]="altavista_pages/pages1";$task[3][1]=$altavista2;
42 $task[4][0]="altavista_pages/pages1";$task[4][1]=$altavista3;
43 $task[5][0]="altavista_pages/pages2";$task[5][1]=$altavista1;
44 $task[6][0]="altavista_pages/pages2";$task[6][1]=$altavista3;
45 $task[7][0]="altavista_pages/pages3";$task[7][1]=$altavista1;
46 $task[8][0]="altavista_pages/pages3";$task[8][1]=$altavista2;
47
48 system("rm $result_file");
49 system("rm $log_file");
50
51 print_log_result "Korrekt:\n";
52
53 for my $i (0..2) {
54   $dir=$task[$i][0];
55   $wrapper=$task[$i][1];
56
57   print_log_result $wrapper->{name};
58   print_log_result ", $dir\n";
59
60   opendir DIR,"$dir";
61   (undef,undef,@files)=readdir DIR;
62   closedir DIR;
63   @files=sort {lc($a) cmp lc($b)} @files;
64   foreach $file (@files) {
65     $lcfile=lc $file;
66     print_log_result "$lcfile\t";
67     my $html=SearchEngineWrapper::load_html("$dir/$file");
68     $wrapper->parse("tmp_altavista",$html);
69   }
70 }
71
72 print_log_result "\nInkorrekt:\n";
73
74 for my $i (3..8) {
75   $dir=$task[$i][0];
76   $wrapper=$task[$i][1];
77
78   print_log_result $wrapper->{name};
79   print_log_result ", $dir\n";
80
81   opendir DIR,"$dir";
82   (undef,undef,@files)=readdir DIR;
83   closedir DIR;
84   @files=sort {lc($a) cmp lc($b)} @files;
85   foreach $file (@files) {
86     $lcfile=lc $file;
87     print_log_result "$lcfile\t";
88     my $html=SearchEngineWrapper::load_html("$dir/$file");
89     $wrapper->parse("tmp_altavista",$html);
90   }
91 }
```

Programmliste 4.6: Quellcode von main_altavista.pl

In Programmliste 4.6 ist `main_altavista` dargestellt; die anderen sehen ähnlich aus. Die Ergebnisse des Tests werden in eine Datei geschrieben; diese Ergebnisse werden in Anhang A präsentiert und in Abschnitt 4.3 diskutiert. Weniger wichtige Informationen werden in eine andere Datei geschrieben, die beim Testen der Anwendung benutzt wurde. Diese

beiden Dateien werden vor jedem Schreibzugriff geöffnet und danach wieder geschlossen, damit die Ausgabe des Programms sofort in den Dateien erscheint und nicht im Cache bleibt. Somit konnte der Testvorgang besser beobachtet werden. Das Programm wurde damit auch nicht wesentlich langsamer, weil die meiste Zeit sowieso mit dem Herunterladen von HTML-Seiten vergeht.

Es werden drei Wrapper erstellt, mit verschiedenen regulären Ausdrücken. Die HTML-Seiten sind auch in drei Verzeichnissen zu finden. Es ergeben sich 9 Testfallgruppen, die in der Liste `@task` definiert sind: `$task[$i][0]` ist das jeweilige Verzeichnis mit den Testfällen und `$task[$i][1]` ist der zu verwendende Wrapper. Die Tests werden dann in zwei Schleifen bearbeitet: in der ersten Schleife die ersten drei Testfallgruppen, in denen der richtige Wrapper verwendet wird, in der zweiten Schleife der Rest, in denen also ein inkorrekter Wrapper verwendet wird. In den Schleifen werden einfach alle Dateien der jeweiligen Verzeichnisse dem jeweiligen Wrapper übergeben. Die Validierung wird automatisch vom Wrapper aus aufgerufen.

4.3 Ergebnisse der Fallstudie

4.3.1 Beobachtungen mit dem interaktiven Metasearch-Prototyp

Während der Benutzung des interaktiven Metasearch-Prototyps wurden einige Änderungen der Suchmaschinenausgaben entdeckt, insbesondere bei AltaVista:

- Beim Beginn der Arbeit (Ende Februar 2001) beschrieb folgender reguläre Ausdruck die zu extrahierenden Tupeln: (in Perl-Syntax)
  `<dl><dt><b>\d+\.  </b><a href="(.+?)"><b>(.+?)</b></a><dd>`

- Anfang März wurde die Ausgabe von AltaVista in mehreren Hinsichten verändert. Das erste `<b>`-Tag wurde mit einem Argument versehen: `<b class=txt2>`. Ausserdem wurde die Reihenfolge der `<a...>`- und `<b>`-Tags bzw. der `<\a>`- und `<\b>`-Tags um den Titel umgekehrt.

- Mitte März gab es zwei weitere Änderungen. Erstens wurde zum `<a...>`-Tag ein `onMouseOver`-Script zugefügt, und zweitens bekamen die URLs ein `/r?r`-Präfix. Das heisst, dass der Anwender seitdem durch ein lokales Script von AltaVista geführt wird, wenn man auf ein Link klickt. Diese beiden Änderungen hängen auch stark miteinander zusammen: mit dem `onMouseOver`-Script wird nämlich erreicht, dass in der Statusleiste das richtige URL angezeigt wird, so dass der Benutzer gar nicht mitbekommt, dass er nicht direkt zu diesem URL kommt.

- Ungefähr am 1. Mai gab es wieder drei Änderungen. Das Script, das die Anfragen entgegennahm, war früher `http://www.altavista.com/cgi_bin/query`; seit der Veränderung ist es `http://www.altavista.com/sites/search/web`. Auch das Script `r`, wodurch man beim Anklicken eines Links geführt wird, bekam neue Argumente: `ck_sm` und `ref`. Ausserdem sind die Antworten nicht mehr nummeriert sondern nur mit einem Symbol gegliedert.

➢ Zur Zeit (Mitte Mai 2001) ist der zu benutzende reguläre Ausdruck:
```
<dl><dt>&#149;<b><a href="/r?ck_sm=.*?&ref=.*?&r=(.+?)" \
onMouseOver=.*?>(.+?)</a></b>
```

4.3.2 Ergebnisse des stapelverarbeiteten Tests

Die vollständigen Ergebnisse sind in Anhang A aufgelistet. An dieser Stelle werden die Ergebnisse diskutiert, und die entsprechenden Schlüsse gezogen.

Bei AltaVista (siehe A.1) werden in jenen Fällen, in denen der korrekte Wrapper benutzt wird, immer 10 Tupeln extrahiert. Meistens sind alle Tupeln syntaktisch völlig richtig: alle drei syntaktische Tests finden 0 Fehler. Es gibt nur einige Fälle, in denen einer der Extrakte leer ist, was auch als Fehler angezeigt wird. Die Anzahl der nicht erreichbaren URLs ist aber ziemlich hoch: sie variiert von 10% bis 70%, der Durchschnitt ist etwa 40%. An dieser Stelle soll aber darauf hingewiesen werden, dass das "historische Daten" sind: es ist kein Wunder, dass fast die Hälfte solcher Seiten, die vor drei Jahren noch existiert haben, jetzt nicht mehr erreichbar sind [24]. Es ist zu erwarten, dass diese Anzahl bei aktuellen Daten viel geringer ist (siehe Abschnitt 4.4).

Als die Seiten der ersten Gruppe mit dem zweiten Wrapper bearbeitet werden, passiert etwas sehr Interessantes: es wird immer genau ein Tupel extrahiert, in dem das URL sowohl syntaktisch als auch semantisch korrekt ist, der Titel ist auch korrekt, aber der Extrakt nicht. Das liegt daran, dass der Extrakt in die Seiten der zweiten Gruppe anders eingebettet ist, so dass der reguläre Ausdruck des ersten Wrappers das URL und den Titel des ersten Tupels korrekt extrahiert, danach wird aber praktisch der ganze Rest der Seite fälschlicherweise als Extrakt interpretiert. Deswegen ist das URL und der Titel des ersten Tupels korrekt, aber der Extrakt ist zu lang, und deswegen werden auch keine weiteren Tupeln extrahiert.

In den anderen Fällen, in denen eine Seite von AltaVista mit einem falschen Wrapper bearbeitet wird, wird gar nichts extrahiert. Das liegt daran, dass die Ausgabe in einer solchen Weise verändert wurde, dass es keine Übereinstimmungen mit dem regulären Ausdruck gibt.

Die Ergebnisse mit Lycos (siehe A.2) sind sehr ähnlich. Wenn der korrekte Wrapper benutzt wird, ist die Anzahl der extrahierten Tupeln immer 10, alle Tupeln sind aus jeder syntaktischen Hinsicht korrekt, aber ungefähr die Hälfte der URLs sind nicht mehr erreichbar. Wenn ein falscher Wrapper benutzt wird, ist die Anzahl der extrahierten Tupeln immer 0.

Die Ergebnisse mit MetaCrawler (siehe A.3) sind viel bunter. Der Grund ist, dass die Ausgabe von MetaCrawler bei weitem nicht so strukturiert und regulär ist, wie die der beiden anderen Suchmaschinen. Dementsprechend gibt es auch bei der Anwendung des korrekten Wrappers Tupeln, die vom Validator als fehlerhaft bezeichnet werden: die Anzahl der syntaktisch fehlerhaft gefundenen URLs variiert von 5% bis 15%, die Anzahl der fehlerhaft gefundenen (also zu langen) Titel von 5% bis 20% und die Anzahl der fehlerhaft gefundenen (also zu langen) Extrakte von 0% bis 40%. Diese hohe Anzahl von falsch gefundenen Extrakten liegt daran, dass MetaCrawler die von den verschiedenen Suchmaschinen stammenden Extrakte – wenn die Seite von mehreren Suchmaschinen gefunden wurde, und die

einzelnen Extrakte verschieden sind – zusammenfügt. Die Anzahl der extrahierten Tupeln variiert auch von 14 bis 21. Andererseits ist die Anzahl der nicht mehr erreichbaren URLs niedriger als bei den anderen Suchmaschinen: zwischen 15% und 40%, im Durchschnitt etwa 30%. Das kann daran liegen, dass MetaCrawler solche Sites bevorzugt, die bei vielen anderen Suchmaschinen gefunden wurden, die also wahrscheinlich größer und stabiler sind.

Wenn die Seiten der ersten und dritten Gruppe mit dem zweiten Wrapper bearbeitet werden, können entweder keine Tupeln extrahiert werden, oder es gibt eine rein zufällige Übereinstimmung mit etwas, was in jeder Hinsicht falsch ist, wie es auch aus der Ausgabe der Tests ersichtlich ist. Wenn man aber umgekehrt, die Seiten der zweiten Gruppe mit dem ersten oder dritten Wrapper – die, wie schon angedeutet, gleich sind – bearbeitet, werden 20 Tupeln extrahiert, die meistens korrekte Titel und Extrakte, aber lauter sowohl syntaktisch als auch semantisch falsche URLs haben. Das liegt eben daran, dass die Einbettung der URLs verändert wurde, die der Titel und Extrakte aber nicht, so dass die Extraktion weiterhin möglich ist, aber die extrahierten URLs völlig falsch sind.

4.3.3 Zusammenfassung der Ergebnisse

Insgesamt kann gesagt werden, dass die implementierten, relativ einfachen Validierungsmethoden zusammen sehr effizient eingesetzt werden können. Bei korrekten Ausgaben war die Anzahl der fehlerhaft gefundenen Tupeln höchstens 70%, und auch das liegt höchstwahrscheinlich daran, dass die Seiten alt sind (siehe Abschnitt 4.4). Bei normaler Nutzung ist diese Zahl gewiss viel geringer. Die Anzahl der syntaktisch fehlerhaft gefundenen Tupeln war zwischen 0 und 40% (und zwar meistens 0). Andererseits, wenn die Ausgabe falsch war, dann konnten entweder keine Tupeln extrahiert werden – in diesem Fall kann eine Testanfrage, auf die es bestimmt eine nicht nullwertige Antwort gibt, benutzt werden, um zu überprüfen, ob die Ausgabe wirklich falsch ist –, oder aber mindestens ein Test (oft aber auch alle) hat 100% der Tupeln fehlerhaft gefunden. Wegen diesem großen Unterschied im Verhalten kann man also aufgrund der Ergebnisse dieser Tests mit sehr großer Sicherheit feststellen, ob die Ausgabe falsch ist. Dadurch kann gleichzeitig eine fast hundertprozentige Spezifizität und Sensitivität erreicht werden.

Es ist auch interessant zu vergleichen, wieviel die einzelnen Verfahren genützt haben. Aus den Ergebnissen sieht man, dass eigentlich schon zwei von den implementierten vier Methoden – nämlich die syntaktischen Tests für das URL und für den Extrakt – ausreichen würden, zwischen den fehlerhaften und den fehlerfreien Zugriffen zu unterscheiden. Aber es sind natürlich auch solche Fälle denkbar, wo eben die anderen Tests nötig wären. Sie haben auch nicht geschadet, und mit einer größeren Anzahl von Validierungsmethoden ist die Wahrscheinlichkeit, dass auch zukünftige Fehler erfolgreich ausgefiltert werden können, auf jeden Fall höher. Es scheint auch wichtig zu sein, mindestens eine Validierung für jedes Element der Tupeln zu haben, weil sonst Fehler, die nur dieses Element betreffen, verborgen bleiben.

In der Theorie zuverlässiger Systeme (siehe z. B. [31]) wird es bewiesen, dass durch Redundanz, und durch eine entsprechende Logik zur Kombination mehrerer unzuverlässiger Systeme, ein zuverlässiges System erstellt werden kann (siehe z. B. TMR, Triple Modular Redundancy). Auch hier führt die Kombination von mehreren unzuverlässigen, aber sehr einfachen Validierungsverfahren zu einem sehr zuverlässigen Validator. (Bemerkung: das

Wort „Redundanz“ wurde schon an zwei Stellen in Abschnitt 3 verwendet. Damals ging es um redundante Informationsquellen bzw. Redundanz in der Ausgabe der Informationsquelle. Diesmal handelt es sich um redundante Validierungsmethoden. Alle dieser drei Arten von Redundanz helfen aber, das Mediatorsystem so zuverlässig wie möglich zu gestalten.)

Alle implementierten Verfahren sind zwar sehr einfach, aber es gab größere Unterschiede im mit ihnen verbundenen Aufwand. Die Valadierungsmethoden für Titel und Extrakt waren sehr einfach zu erstellen, und bedeuten auch sehr wenig Overhead. Die syntaktische Validierung für die URLs war etwas schwieriger zu implementieren, verursacht auch etwas mehr Overhead, aber auch nicht bedeutend, denn der Zugriff auf die externe Informationsquelle dauert meistens viel länger. Bei dem semantischen Validator ist das Overhead schon deutlich; die Implementierung wäre einfach gewesen, und wurde nur dadurch komplizierter, dass die Tests parallel durchgeführt wurden.

Es ist noch zu erwähnen, dass MetaCrawler und Metasearch eine sehr ähnliche Funktionalität haben. Daraus folgt, dass die Validierung der Zugriffe auf MetaCrawler auch die Validierung einer ganzen Mediatoranwendung gut modelliert. Also können die obigen einfachen Methoden auch in der Validierung einer ganzen Mediatoranwendung sehr gut eingesetzt werden.

4.4 Weitere Versuche

Es bleibt noch zu bedenken, wie gut die Ergebnisse, die anhand von Daten aus 1998 gewonnen wurden, auf die heutige und zukünftige Lage übertragbar sind. Aus dieser Hinsicht ist es sogar vorteilhaft, dass ältere Daten benutzt wurden, denn damit wird es möglich, durch den Vergleich mit dem heutigen Stand auch Tendenzen für die Zukunft zu identifizieren.

Als erstes kann man beobachten, dass die typischen Änderungen der Ausgabe der Suchmaschinen im Jahre 1998 (siehe Abschnitt 4.3) und im Jahre 2001 (siehe Abschnitt 4.3.1) sehr ähnlich sind. Also kann man annehmen, dass das auch in der Zukunft so sein wird. Das ist einerseits schlecht, denn das bedeutet, dass die Wrapper nach wie vor Validierung und Reparatur brauchen. Andererseits ist das vorteilhaft, weil es die Erstellung von robusten, für längere Zeit einsetzbaren Validatoren ermöglicht.

Ein anderer wichtiger Punkt wurde schon angedeutet: die alten Seiten beinhalteten viele URLs, die nicht mehr erreichbar waren. Es ist zu erwarten, dass das bei aktuellen Seiten nicht mehr der Fall ist, und zwar aus zwei Gründen. Erstens, es ist gut möglich, dass einige URLs bei der Erstellung der Suchmaschinenausgabe noch erreichbar waren, inzwischen aber nicht mehr. Zweitens, heutzutage legen auch die Suchmschinen größeren Wert darauf, möglichst nur existierende URLs zurückzugeben [33].

Um das zu untersuchen, wurden einige weitere Versuche von Hand durchgeführt: die URLs in der Ausgabe der untersuchten drei Suchmaschinen auf die Anfrage “happy” wurden überprüft. Die Ergebnisse sind in Tabelle 4.1 dargestellt.

Wie man sieht, hat sich die Lage drastisch verbessert. Also kann man auch die oben beschriebene semantische Validierung gut verwenden.

Suchmaschine	Anzahl von falschen URLs
AltaVista	0%
Lycos	0%
MetaCrawler	10%

Tabelle 4.1: Ergebnisse der aktuellen Versuche

Kapitel 5

Diskussion

In diesem abschließenden Kapitel werden zuerst einige mit der vorliegenden Arbeit vergleichbare Arbeiten präsentiert (Abschnitt 5.1), danach werden die Ergebnisse der Arbeit zusammengefasst (Abschnitt 5.2). Am Ende wird ein Ausblick über mögliche Weiterentwicklungen der Arbeit bzw. der Verhältnisse der Informationsintegration und der Wrapperkonstruktion gegeben (Abschnitt 5.3).

5.1 Vergleichbare Arbeiten

Die Validierung von Zugriffen auf externe Informationsquellen in einer Mediatorumgebung ist in der bisherigen Literatur nach unserem Kenntnisstand, so gut wie nicht beachtet worden. Es gibt viele Arbeiten, die sich mit Mediatoren und ähnlichen Architekturen zur Informationsintegration befassen (z. B. [16, 36, 40, 42]), aber diese vernachlässigen meistens die Validierung. Daher sind die aus der Sicht der vorliegenden Arbeit relevantesten Ergebnisse nicht in der Mediatorforschung, sondern in der Forschung webbasierter Informationssysteme entstanden.

In vielen dieser Arbeiten geht es um die Erstellung von Wrappern, wobei in diesem Fall unter „Wrapper“ nicht mehr die allgemeine Verbindungskomponente zur Einbindung externer Informationsquellen zu verstehen ist, sondern ein Spezialfall davon, nämlich Web-Wrapper: Programme, die Informationen aus HTML extrahieren. Die meisten solche Arbeiten (z. B. [14, 22, 28, 34]) fokussieren auf die Minimierung des mit der Erstellung von Wrappern verbundenen Aufwands. Dazu werden Methoden wie maschinelles Lernen zur automatischen Erstellung der Wrapper vorgeschlagen.

Es gibt auch eine Arbeit, die sich ausschließlich mit der Validierung von (Web-)Wrappern beschäftigt: die schon an mehreren Stellen erwähnte Arbeit von Kushmerick, die er in zwei Artikeln präsentiert [21, 23]. Sein Algorithmus RAPTURE arbeitet mit statistischen Features, wie z. B. Anzahl der extrahierten Tupeln, Länge der einzelnen Elemente, Anzahl der Wörter, Anteil von HTML-Sonderzeichen („<“, „>“) usw. Der Algorithmus lernt anhand positiver Beispiele die Parameter von Normalverteilungen, die die Verteilungen dieser Features am besten beschreiben. Mittels dieser gelernten Verteilungen wird dann die Wahrscheinlichkeit, dass eine neue Wrapperausgabe hinsichtlich eines Features korrekt ist,

abgeschätzt. Diese Wahrscheinlichkeiten bezüglich der einzelnen Features werden dann zu einer Gesamtwahrscheinlichkeit kombiniert. Diese Wahrscheinlichkeit wird dann mit jenen Wahrscheinlichkeiten verglichen, die dasselbe Verfahren für die Trainigsbeispiele geliefert hat. Auch dieser letzte Schritt wird mittels einer Normalverteilung gemacht: die Parameter einer Normalverteilung, die die Wahrscheinlichkeiten der Trainingsbeispiele gut beschreibt, werden auch gelernt, und es wird anhand dieser Verteilung bestimmt, wie wahrscheinlich es ist, dass die aktuelle Wrapperausgabe korrekt ist.

Kushmerick vergleicht in seinen Artikeln die Leistung von RAPTURE mit der des herkömmlichen Regressionstests, und findet, dass RAPTURE viel bessere Ergebnisse liefert. Er macht auch viele Versuche, um zu bestimmen, welche Features die beste Leistung produzieren. Das Ergebnis ist, dass die Verwendung von mehreren Features meistens nicht besser ist, als die Verwendung eines einzigen. Das beste Feature war bei weitem der Anteil von HTML-Sonderzeichen. Mit der Verwendung dieses Features konnte eine Korrektheit von etwa 99.99% erreicht werden, der positive prediktive Wert war aber wesentlich niedriger. (Eine genaue Angabe steht leider nicht zur Verfügung, weil er andere Maße verwendet hat. Aber aus den Daten ist auf jeden Fall ersichtlich, dass der Algorithmus viele falsche positive Ergebnisse produziert.)

Die Leistung von RAPTURE ist zwar sehr gut, besonders im Vergleich zu Regressionstests, aber es gibt damit auch einige Probleme:

- Die Normalverteilung ist zwar in vielen Fällen zutreffend für die Verteilung der Features, in anderen Fällen aber eben nicht. Das kann zu Fehlurteilen führen, besonders dann, wenn nur ein einziges Feature benutzt wird. Die Normalverteilung gibt eine positive Wahrscheinlichkeit für jeden Wert, auch für offensichtlich unmögliche Werte, wie z. B. für eine negative Länge. Ausserdem gibt die Normalverteilung nur für den Mittelwert 1, für alle anderen Werte weniger als 1, so dass also auch ganz plausible Ausgaben eine Wahrscheinlichkeit weniger als 1 bekommen. Das ist vor allem dann problematisch, wenn es korrekte Ausgaben gibt, die weit vom Mittelwert liegen, aber selten vorkommen, somit die Standardabweichung nicht genügend beeinflussen können, und dadurch eine geringe Wahrscheinlichkeit erhalten. Das kann zu falschen positiven Testergebnissen und dadurch zu einem niedrigem positiven prediktiven Wert führen.

- Die Benutzung mehrerer Features ist problematisch, und trägt zur Steigerung der Leistung nicht bei, obwohl man intuitiv vermutet, dass das so sein müsste. Das liegt wahrscheinlich daran, dass die von Kushmerick für diesen Zweck vorgesehenen und untersuchten Methoden – wie z. B. das Multiplizieren der einzelnen Wahrscheinlichkeiten, oder die Bildung des Mittelwertes der einzelnen Wahrscheinlichkeiten – gegenüber der Unsicherheit, die die einzelnen Wahrscheinlichkeiten aufweisen, nicht robust genug ist, so dass diese Unsicherheit nur vergrößert wird.

- Im Endeffekt entstehen sowohl für korrekte als auch für inkorrekte Wrapperausgaben sehr niedrige Wahrscheinlichkeiten, so dass es sehr schwierig ist, die Grenze zwischen den beiden Gruppen zu ziehen.

- In seiner gegenwärtigen Form, in der nur die Mittelwerte und Standardabweichungen für die einzelnen Features gespeichert werden, kann RAPTURE nicht inkrementell

trainiert werden, und somit kann er sich auch nicht an ändernde Verhältnisse anpassen. Das könnte natürlich verbessert werden, indem alle Trainingsdaten gespeichert werden, aber auch dann könnte er nur neu, aber nicht inkrementell trainiert werden.

Kapitel 3 und 4 der vorliegenden Arbeit können als die Lösung auf diese Probleme betrachtet werden. Die wichtigsten Ergebnisse in dieser Hinsicht sind die folgenden:

- ➢ Die Benutzung eines angemessenen, speziell für die konkrete Anwendung definierten Validators an Stelle einer allgemeinen, automatischen Methode ist vorteilhaft für die Qualität der Validierung.
- ➢ Ein Intervall beschreibt die korrekten Ausgaben meistens viel besser als eine Normalverteilung.
- ➢ Wie die Ergebnisse der Validierung der einzelnen Features zu einem Gesamtergebnis kombiniert werden, soll auch durch eine angemessene Logik definierbar sein, damit eine höhere Zuverlässigkeit erreicht werden kann.

Es ergibt sich ein viel zuverlässigerer Validator. Der Preis dafür ist, dass der Validator von Hand implementiert werden muss. Wie es sich aber in Kapitel 4 herausstellt, sind oft sehr einfache Validierungsverfahren ausreichend, deren Implementierung keinen allzu großen Aufwand bereitet; auch sind diese Validatoren über längere Zeit brauchbar, so dass sich der relativ kleine und einmalige Aufwand auf jeden Fall lohnt.

Cohen [9] präsentiert Methoden zur allgemeinen Strukturerkennung in HTML-Seiten. Dazu benutzt er die von ihm entwickelten „soft“ Logik WHIRL [8], die sowohl einen „soft“ Universalquantor („*many*“), als auch einen Begriff der textuellen Ähnlichkeit unterstützt. Letztere wird durch die von Salton [32] eingeführte „Dokumentvektor“-Notation und TF-IDF Schema definiert. Cohen hat sein in WHIRL geschriebenes Programm für eine begrenzte Klasse von HTML-Seiten ausprobiert, und hat festgestellt, dass jene Struktur, die vom Programm als wahrscheinlichste für die gegebene Seite vorgeschlagen wurde, in etwa 70% der Fälle richtig war. Er erwähnt auch, dass diese Methode auch für die Erkennung von Strukturänderungen und dadurch für die Wrappervalidierung verwendet werden kann. Allerdings hat er diesen Ansatz nicht an richtigen Webseiten ausprobiert.

Eine andere verwandte Forschungsrichtung ist das maschinelle Lernen von Grammatiken. Carrasco und Oncina [7] haben den Algorithmus ALGERIA konstruiert, der aus einer Menge von Trainingsbeispielen eine stochastische reguläre Grammatik für die Sprache der Beispiele generiert. Ist n die Anzahl der Beispiele, so braucht ALGERIA $O(n^3)$ Schritte. Es ist weiterhin bewiesen, dass die gelernte Grammatik bei $n \to \infty$ zu einer richtigen Grammatik der Sprache konvergiert, angenommen, dass die Beispiele wirklich mit einer stochastischen regulären Grammatik generiert werden. Goan, Benson und Etzioni [11] entwickelten diesen Ansatz weiter, indem sie mit WIL (*Web Information Learner*) einen Algorithmus konstruiert haben, der speziell für Daten im Web sehr gut geeignet ist. Mit starker Nutzung der verfügbaren Typ- und Strukturinformationen, sowie einer neuen Ordnungstechnik (*More-Data-First Search*) kann WIL bei gleichem Beispielbestand eine deutlich höhere Korrektheit erzielen als ALGERIA. WIL braucht allerdings schon $O(n^4)$ Schritte; die Konvergenz ist genauso beweisbar. Im Artikel von Goan, Benson und Etzioni wird auch erwähnt, dass diese

Techniken in einem System zur Integration von Informationssystemen verwendet werden könnten. Es wurde nicht erwähnt, dass sie auch zur Validierung benutzt werden könnten, aber es ist in Abschnitt 3.2 der vorliegenden Arbeit beschrieben, wie das ginge.

Am *Information Sciences Institute* der *University of Southern California* laufen mehrere Projekte zur Integration heterogener Informationsquellen. Es wurden auch Ansätze zur Validierung und automatischen Reparatur von Web-Wrappern entwickelt [20, 25]. Mit seinen fortgeschrittenen Techniken, wie z. B. *co-testing* [27], ist der hierarchische Wrapperkonstruktionsalgorithmus STALKER [20] ein sehr effektives Verfahren zur automatisierten Erstellung von Wrappern. Andererseits wurde der Algorithmus DATAPRO [25] direkt zur Validierung und Reparatur von Wrappern entwickelt. Im Gegensatz zu den obigen Ansätzen wird die Wrapperausgabe bei DATAPRO weder durch globale statistische Features noch durch Dokumentvektore, sondern durch sogenannte *Tokens* beschrieben. Tokens können entweder konkrete Wörter, oder aber Verallgemeinerungen von Wörter sein. DATAPRO baut einen Präfixbaum der Tokens auf, in dem es am Anfang nur konkrete Wörter gibt, die dann später verallgemeinert werden, um allgemeine signifikante Tokenfolgen zu erreichen. "Signifikant" bedeutet, dass die Tokenfolge in den Trainingsbeispielen deutlich öfter vorkommt, als es durch Zufall zu erwarten wäre. DATAPRO kann offensichtlich auch zur Validierung von Wrappern benutzt werden, indem er die signifikanten Tokens, die die zu extrahierenden Informationen beschreiben, lernt, und die Signifikanz dieser auch bei neuen Ausgaben überprüft. Eine praktische Evaluierung ergab, dass DataPro eine sehr hohe Sensitivität besitzt, sogar höher als RAPTURE, aber der positive prediktive Wert ist auch hier ziemlich niedrig: aus 443 Tests gab es 40 falsche positive Ergebnisse.

Insgesamt kann man feststellen, dass die Fachliteratur für die Validierung von Zugriffen relativ spärlich ist. Die meisten Arbeiten fokussieren auf die automatische Erstellung von Wrappern und Validatoren fürs Web. Mit solchen Methoden kann zwar eine hohe Sensitivität erreicht werden, eine gleichzeitig hohe Spezifizität wurde jedoch noch nicht berichtet. Diese Verfahren sind auf die Validierung der Wrapperausgabe begrenzt.

5.2 Zusammenfassung

In der vorliegenden Arbeit wurde die Validierung von Zugriffen auf externe Informationsquellen in einer Mediatorumgebung behandelt, mit spezieller Betonung auf Internetbasierte Informationssysteme:

- Zuerst wurden die bei den Zugriffen auftretenden typischen Fehler beschrieben und in mehrerer Hinsicht – nach der Quelle des Fehlers, nach den Symptomen und nach der Dauer des Fehlers – klassifiziert.

- Es wurde geklärt, welche die wichtigsten Qualitätsmerkmale eines Validators sind. Dabei kamen Methoden aus der Statistik und aus den Medizinwissenschaften zum Einsatz. Als wichtigste funktionale Merkmale wurden die Korrektheit, die Sensitivität, die Spezifizität und der positive und negative prediktive Wert definiert. Es wurde gezeigt, dass die Spezifizität und der positive prediktive Wert besonders kritisch sind. Ausserdem wurden diverse nicht-funktionale Merkmale beschrieben.

- Verschiedene Arten von Validierungsverfahren wurden präsentiert, wie die Validierung während der Arbeit des Wrappers, die syntaktische und semantische Validierung der Ausgabe des Wrappers, die Verwendung von Testanfragen, das Einbeziehen anderer Informationsquellen und die Validierung des ganzen Mediatorprogramms. Es wurde untersucht, welche Arten von Fehlern die einzelnen Methoden entdecken können, und welche Kosten sie haben.

- Die verschiedenen Einbindungsmöglichkeiten des Validators in die Mediatorarchitektur wurden beschrieben.

- Einige Validierungsmethoden wurden auch in der Praxis, anhand der Validierung des Metasearch-Beispiels demonstriert. Ein interaktiver Metasearch-Prototyp und ein Testprogramm wurden in der Programmiersprache Perl implementiert. Die Testergebnisse zeigen, dass die Verwendung von angemessenen Validierungsverfahren gleichzeitig eine sehr hohe Sensitivität und Spezifizität ermöglicht, was mit den bisher in der Fachliteratur beschriebenen Methoden nicht möglich ist.

5.3 Ausblick

Es gibt einige Punkte, wie diese Arbeit noch weiterentwickelt werden könnte:

- Der in Kapitel 4 beschriebene Ansatz könnte zu einem allgemeinen System zur Validierung von Webwrappern verallgemeinert werden, ähnlich wie RAPTURE (Abschnitt 5.1). Allerdings werden im Gegensatz zu RAPTURE Intervalle statt Normalverteilungen, angemessene Features und eine definierbare Logik zur Kombination der Testergebnisse verwendet.

- Die Anwendbarkeit eines solchen Systems müsste mit weiteren Tests untersucht werden. Dabei sollten sowohl historische als auch aktuelle Daten benutzt werden.

- Die Validierungsfunktionalität sollte in KOMET integriert werden.

Es ist auch wichtig zu bemerken, dass es neben XML, das schon in Abschnitt 1.1 erwähnt wurde, auch weitere – teilweise auf XML aufsetzende – Ansätze gibt, die die Zukunft der Informationsintegration stark beeinflussen können:

- Wenn die Benutzung von intelligenten Softwareagenten eine weite Verbreitung findet, werden wohl viele Sites ihre Informationen nicht mehr nur in für Menschen bequeme Formaten darbieten, sondern auch in anderen Formaten, die speziell für Maschinen geeignet sind. Oder es ist auch denkbar, dass Sites einige Hilfsfunktionen für Wrapper oder Validatoren bereitstellen, z. B. eine Diagnostikfunktion. Es kann auch sein, dass einfach die Redundanz in der Ausgabe der Informationsquellen erhöht wird, was die Validierung von Zugriffen erleichtert. Z. B. könnten dafür spezielle Fehlerkorrekturcodes verwendet werden.

- Simple Object Access Protocol[1] ist ein in XML definiertes Protokoll zum Zugriff auf verteilte, heterogene Objekte. Es besteht aus der Definition eines *Envelope*s zur Beschreribung des Inhalts, Regeln zur Kodierung von Datentypen und einer Konvention für die Benutzung in einem *Remote Procedure Call* (RPC) Szenario. Wenn sich solche Standards durchsetzen könnten, wäre die Integration heterogener Informationsquellen auch weniger fehleranfällig.

- Web Services Description Language[2] basiert auch auf XML, und erlaubt die Definition von abstrakten Endpunkten, die in XML kodierte Nachrichten empfangen und bearbeiten können, und zu abstrakten Diensten zusammengebunden werden können.

- PEDGUIs (Printed Embedded Data Graphical User Interfaces [12]) stellen einen ganz anderen Ansatz dar. Hierbei geht es um die Einbettung von für Maschinen lesbaren Informationen – z. B. Strichcodes – in Bilder und andere graphische Elemente, die eigentlich für die menschliche Wahrnehmung gedacht sind. Dabei kann eine Informationsdichte von etwa 400 Bytes per Quadratzoll erreicht werden, ohne die menschliche Wahrnehmung zu stören. Solche Methoden könnten benutzt werden, um Informationen gleichzeitig für Mensch und Maschine zu liefern.

- Model Driven Architecture[3] ist die von der Object Management Group[4] vorgeschlagene Lösung auf das Problem der Heterogenität. MDA setzt auf frühere Standards der OMG (wie z. B. UML[5] und CORBA[6]) auf, und ermöglicht eine völlig platformunabhängige Beschreibung von Systemen. Auch so eine Standardisierung würde die Integration heterogener Informationsquellen stark vereinfachen.

Da es kein Ziel der Arbeit war, soll es an dieser Stelle nur noch angedeutet werden, dass es statt der in Metasearch benutzten regulären Ausdrücke auch bessere Methoden zur Extraktion von Informationen gibt:

- Die Veränderungen der Sites bestehen oft nur aus Änderungen gewisser Parameter von Tags. (Z. B. wenn ein `onMouseOver`-Script zu einem `<a>` Tag zugefügt wird.) Auch diese Änderungen führen meistens dazu, dass mit dem regulären Ausdruck nichts mehr extrahiert werden kann. Wenn der Wrapper aber nicht unmittelbar am Text der HTML-Datei, sondern am HTML-Syntaxbaum arbeiten würde, könnte er solche Änderungen ohne weiteres tolerieren. Damit könnte die Anzahl von fehlerhaften Zugriffen drastisch vermindert werden.

- Die Theorie der Informationsextraktion hat eine Menge verschiedene Algorithmen entwickelt, die überhaupt zur Extraktion von gewissen Informationen aus Text oder semi-strukturierten Quellen benutzt werden können. Für einen Vergleich dieser Algorithmen mit Betonung auf die Verwendung in der Wrappererstellung siehe [26].

[1] `http://www.w3.org/TR/SOAP`
[2] `http://www.w3.org/TR/wsdl`
[3] `http://www.omg.org/mda`
[4] `http://www.omg.org`
[5] `http://www.omg.org/technology/uml`
[6] `http://www.corba.org`

- Viele Änderungen, die die reguläre Ausdrücke nicht mehr behandeln können, sind dem menschlichen Benutzer sogar verborgen, weil sie die Anordnung der Seite nicht beeinflussen. Deswegen ist es zu erwarten, dass eine Verallgemeinerung von regulären Ausdrücken, die an der 2-dimensionalen Anordnung der resultierenden Seite, und nicht an dem 1-dimensionalen Quelltext arbeitet, viel besser funktioniert.

Anhang A

Testergebnisse

In den folgenden Abschnitten werden die Ergebnisdateien der stapelverarbeiteten Tests präsentiert.

Erklärung der Tabellen:

Spalte 1:	Name der Testseite
Spalte 2:	Anzahl der extrahierten Tupeln
Spalte 3:	Anzahl der fehlerhaft gefundenen URLs
Spalte 4:	Anzahl der fehlerhaft gefundenen Titeln
Spalte 5:	Anzahl der fehlerhaft gefundenen Extrakte
Spalte 6:	Anzahl der nicht erreichbaren Seiten

A.1 Ergebnisse bei Altavista (`results_altavista.txt`)

```
Korrekt:
altavista1, altavista_pages/pages1
alta-happy-1998-04-12-11-08.html     10     0      0      0      4
alta-happy-1998-04-15-11-08.html     10     0      0      0      1
alta-happy-1998-04-18-11-08.html     10     0      0      0      3
alta-happy-1998-04-21-16-19.html     10     0      0      0      3
alta-happy-1998-04-23-11-06.html     10     0      0      0      3
alta-happy-1998-04-26-11-08.html     10     0      0      0      5
alta-happy-1998-04-29-11-13.html     10     0      0      0      5
altavista2, altavista_pages/pages2
alta-happy-1998-04-30-11-10.html     10     0      0      0      5
alta-happy-1998-05-02-11-26.html     10     0      0      0      5
alta-happy-1998-05-05-13-42.html     10     0      0      0      5
alta-happy-1998-05-08-11-11.html     10     0      0      0      5
alta-happy-1998-05-11-11-09.html     10     0      0      0      5
alta-happy-1998-05-16-10-05.html     10     0      0      0      5
alta-happy-1998-05-23-13-24.html     10     0      0      0      5
alta-happy-1998-05-26-17-09.html     10     0      0      0      5
alta-happy-1998-05-29-11-13.html     10     0      0      0      5
alta-happy-1998-06-02-11-57.html     10     0      0      0      5
alta-happy-1998-06-05-11-09.html     10     0      0      0      5
alta-happy-1998-06-08-11-14.html     10     0      0      0      5
alta-happy-1998-06-11-11-08.html     10     0      0      0      5
alta-happy-1998-06-14-11-14.html     10     0      0      0      5
alta-happy-1998-06-17-11-12.html     10     0      0      0      5
alta-happy-1998-06-20-08-58.html     10     0      0      0      5
alta-happy-1998-06-21-11-11.html     10     0      0      0      5
alta-happy-1998-06-24-11-11.html     10     0      0      0      5
alta-happy-1998-06-27-11-07.html     10     0      0      0      4
```

```
alta-happy-1998-06-30-11-17.html    10    0    0    0    4
alta-happy-1998-07-04-16-33.html    10    0    0    0    5
alta-happy-1998-07-07-11-17.html    10    0    0    0    7
alta-happy-1998-07-10-11-12.html    10    0    0    0    5
alta-happy-1998-07-13-11-09.html    10    0    0    0    2
alta-happy-1998-07-16-11-08.html    10    0    0    0    4
alta-happy-1998-07-18-11-22.html    10    0    0    0    6
alta-happy-1998-07-21-11-08.html    10    0    0    0    2
alta-happy-1998-07-24-11-13.html    10    0    0    0    5
alta-happy-1998-07-27-11-23.html    10    0    0    0    5
alta-happy-1998-07-30-11-07.html    10    0    0    0    6
alta-happy-1998-08-02-11-15.html    10    0    0    0    5
alta-happy-1998-08-05-11-07.html    10    0    0    0    5
alta-happy-1998-08-08-11-12.html    10    0    0    0    5
alta-happy-1998-08-11-11-09.html    10    0    0    0    5
alta-happy-1998-08-15-11-11.html    10    0    0    0    5
alta-happy-1998-08-18-11-09.html    10    0    0    0    5
alta-happy-1998-08-21-12-27.html    10    0    0    0    5
alta-happy-1998-08-24-11-14.html    10    0    0    0    5
alta-happy-1998-08-27-11-05.html    10    0    0    0    5
alta-happy-1998-08-30-11-34.html    10    0    0    0    2
alta-happy-1998-09-02-11-13.html    10    0    0    0    2
altavista3, altavista_pages/pages3
alta-happy-1998-09-05-11-15.html    10    0    0    1    2
alta-happy-1998-09-09-11-40.html    10    0    0    1    2
alta-happy-1998-09-12-11-31.html    10    0    0    1    2
alta-happy-1998-09-15-11-24.html    10    0    0    1    2
alta-happy-1998-09-18-11-12.html    10    0    0    1    2
alta-happy-1998-09-21-11-31.html    10    0    0    1    2
alta-happy-1998-09-24-11-12.html    10    0    0    1    3
alta-happy-1998-09-27-11-35.html    10    0    0    1    3
alta-happy-1998-09-30-11-29.html    10    0    0    1    3

Inkorrekt:
altavista2, altavista_pages/pages1
alta-happy-1998-04-12-11-08.html    1    0    0    1    0
alta-happy-1998-04-15-11-08.html    1    0    0    1    0
alta-happy-1998-04-18-11-08.html    1    0    0    1    0
alta-happy-1998-04-21-16-19.html    1    0    0    1    0
alta-happy-1998-04-23-11-06.html    1    0    0    1    0
alta-happy-1998-04-26-11-08.html    1    0    0    1    0
alta-happy-1998-04-29-11-13.html    1    0    0    1    0
altavista3, altavista_pages/pages1
alta-happy-1998-04-12-11-08.html    0    0    0    0    0
alta-happy-1998-04-15-11-08.html    0    0    0    0    0
alta-happy-1998-04-18-11-08.html    0    0    0    0    0
alta-happy-1998-04-21-16-19.html    0    0    0    0    0
alta-happy-1998-04-23-11-06.html    0    0    0    0    0
alta-happy-1998-04-26-11-08.html    0    0    0    0    0
alta-happy-1998-04-29-11-13.html    0    0    0    0    0
altavista1, altavista_pages/pages2
alta-happy-1998-04-30-11-10.html    0    0    0    0    0
alta-happy-1998-05-02-11-26.html    0    0    0    0    0
alta-happy-1998-05-05-13-42.html    0    0    0    0    0
alta-happy-1998-05-08-11-11.html    0    0    0    0    0
alta-happy-1998-05-11-11-09.html    0    0    0    0    0
alta-happy-1998-05-16-10-05.html    0    0    0    0    0
alta-happy-1998-05-23-13-24.html    0    0    0    0    0
alta-happy-1998-05-26-17-09.html    0    0    0    0    0
alta-happy-1998-05-29-11-13.html    0    0    0    0    0
alta-happy-1998-06-02-11-57.html    0    0    0    0    0
alta-happy-1998-06-05-11-09.html    0    0    0    0    0
alta-happy-1998-06-08-11-14.html    0    0    0    0    0
alta-happy-1998-06-11-11-08.html    0    0    0    0    0
alta-happy-1998-06-14-11-14.html    0    0    0    0    0
alta-happy-1998-06-17-11-12.html    0    0    0    0    0
alta-happy-1998-06-20-08-58.html    0    0    0    0    0
alta-happy-1998-06-21-11-11.html    0    0    0    0    0
alta-happy-1998-06-24-11-11.html    0    0    0    0    0
alta-happy-1998-06-27-11-07.html    0    0    0    0    0
```

```
alta-happy-1998-06-30-11-17.html    0    0    0    0    0
alta-happy-1998-07-04-16-33.html    0    0    0    0    0
alta-happy-1998-07-07-11-17.html    0    0    0    0    0
alta-happy-1998-07-10-11-12.html    0    0    0    0    0
alta-happy-1998-07-13-11-09.html    0    0    0    0    0
alta-happy-1998-07-16-11-08.html    0    0    0    0    0
alta-happy-1998-07-18-11-22.html    0    0    0    0    0
alta-happy-1998-07-21-11-08.html    0    0    0    0    0
alta-happy-1998-07-24-11-13.html    0    0    0    0    0
alta-happy-1998-07-27-11-23.html    0    0    0    0    0
alta-happy-1998-07-30-11-07.html    0    0    0    0    0
alta-happy-1998-08-02-11-15.html    0    0    0    0    0
alta-happy-1998-08-05-11-07.html    0    0    0    0    0
alta-happy-1998-08-08-11-12.html    0    0    0    0    0
alta-happy-1998-08-11-11-09.html    0    0    0    0    0
alta-happy-1998-08-15-11-11.html    0    0    0    0    0
alta-happy-1998-08-18-11-09.html    0    0    0    0    0
alta-happy-1998-08-21-12-27.html    0    0    0    0    0
alta-happy-1998-08-24-11-14.html    0    0    0    0    0
alta-happy-1998-08-27-11-05.html    0    0    0    0    0
alta-happy-1998-08-30-11-34.html    0    0    0    0    0
alta-happy-1998-09-02-11-13.html    0    0    0    0    0
altavista3, altavista_pages/pages2
alta-happy-1998-04-30-11-10.html    0    0    0    0    0
alta-happy-1998-05-02-11-26.html    0    0    0    0    0
alta-happy-1998-05-05-13-42.html    0    0    0    0    0
alta-happy-1998-05-08-11-11.html    0    0    0    0    0
alta-happy-1998-05-11-11-09.html    0    0    0    0    0
alta-happy-1998-05-16-10-05.html    0    0    0    0    0
alta-happy-1998-05-23-13-24.html    0    0    0    0    0
alta-happy-1998-05-26-17-09.html    0    0    0    0    0
alta-happy-1998-05-29-11-13.html    0    0    0    0    0
alta-happy-1998-06-02-11-57.html    0    0    0    0    0
alta-happy-1998-06-05-11-09.html    0    0    0    0    0
alta-happy-1998-06-08-11-14.html    0    0    0    0    0
alta-happy-1998-06-11-11-08.html    0    0    0    0    0
alta-happy-1998-06-14-11-14.html    0    0    0    0    0
alta-happy-1998-06-17-11-12.html    0    0    0    0    0
alta-happy-1998-06-20-08-58.html    0    0    0    0    0
alta-happy-1998-06-21-11-11.html    0    0    0    0    0
alta-happy-1998-06-24-11-11.html    0    0    0    0    0
alta-happy-1998-06-27-11-07.html    0    0    0    0    0
alta-happy-1998-06-30-11-17.html    0    0    0    0    0
alta-happy-1998-07-04-16-33.html    0    0    0    0    0
alta-happy-1998-07-07-11-17.html    0    0    0    0    0
alta-happy-1998-07-10-11-12.html    0    0    0    0    0
alta-happy-1998-07-13-11-09.html    0    0    0    0    0
alta-happy-1998-07-16-11-08.html    0    0    0    0    0
alta-happy-1998-07-18-11-22.html    0    0    0    0    0
alta-happy-1998-07-21-11-08.html    0    0    0    0    0
alta-happy-1998-07-24-11-13.html    0    0    0    0    0
alta-happy-1998-07-27-11-23.html    0    0    0    0    0
alta-happy-1998-07-30-11-07.html    0    0    0    0    0
alta-happy-1998-08-02-11-15.html    0    0    0    0    0
alta-happy-1998-08-05-11-07.html    0    0    0    0    0
alta-happy-1998-08-08-11-12.html    0    0    0    0    0
alta-happy-1998-08-11-11-09.html    0    0    0    0    0
alta-happy-1998-08-15-11-11.html    0    0    0    0    0
alta-happy-1998-08-18-11-09.html    0    0    0    0    0
alta-happy-1998-08-21-12-27.html    0    0    0    0    0
alta-happy-1998-08-24-11-14.html    0    0    0    0    0
alta-happy-1998-08-27-11-05.html    0    0    0    0    0
alta-happy-1998-08-30-11-34.html    0    0    0    0    0
alta-happy-1998-09-02-11-13.html    0    0    0    0    0
altavista1, altavista_pages/pages3
alta-happy-1998-09-05-11-15.html    0    0    0    0    0
alta-happy-1998-09-09-11-40.html    0    0    0    0    0
alta-happy-1998-09-12-11-31.html    0    0    0    0    0
alta-happy-1998-09-15-11-24.html    0    0    0    0    0
alta-happy-1998-09-18-11-12.html    0    0    0    0    0
```

```
alta-happy-1998-09-21-11-31.html    0    0    0    0    0
alta-happy-1998-09-24-11-12.html    0    0    0    0    0
alta-happy-1998-09-27-11-35.html    0    0    0    0    0
alta-happy-1998-09-30-11-29.html    0    0    0    0    0
altavista2, altavista_pages/pages3
alta-happy-1998-09-05-11-15.html    0    0    0    0    0
alta-happy-1998-09-09-11-40.html    0    0    0    0    0
alta-happy-1998-09-12-11-31.html    0    0    0    0    0
alta-happy-1998-09-15-11-24.html    0    0    0    0    0
alta-happy-1998-09-18-11-12.html    0    0    0    0    0
alta-happy-1998-09-21-11-31.html    0    0    0    0    0
alta-happy-1998-09-24-11-12.html    0    0    0    0    0
alta-happy-1998-09-27-11-35.html    0    0    0    0    0
alta-happy-1998-09-30-11-29.html    0    0    0    0    0
```

A.2 Ergebnisse bei Lycos (results_lycos.txt)

```
Korrekt:
lycos1, lycos_pages/pages1
lyco-happy-1998-04-12-11-08.html    10    0    0    0    7
lyco-happy-1998-04-15-11-08.html    10    0    0    0    6
lyco-happy-1998-04-18-11-08.html    10    0    0    0    6
lyco-happy-1998-04-21-16-20.html    10    0    0    0    6
lyco-happy-1998-04-23-11-06.html    10    0    0    0    6
lyco-happy-1998-04-26-11-08.html    10    0    0    0    6
lyco-happy-1998-04-29-11-13.html    10    0    0    0    6
lyco-happy-1998-04-30-11-10.html    10    0    0    0    6
lyco-happy-1998-05-02-11-26.html    10    0    0    0    6
lyco-happy-1998-05-05-13-42.html    10    0    0    0    6
lyco-happy-1998-05-08-11-11.html    10    0    0    0    6
lyco-happy-1998-05-11-11-09.html    10    0    0    0    6
lyco-happy-1998-05-16-10-05.html    10    0    0    0    5
lyco-happy-1998-05-23-13-25.html    10    0    0    0    5
lyco-happy-1998-05-26-17-10.html    10    0    0    0    6
lyco-happy-1998-05-29-11-13.html    10    0    0    0    6
lyco-happy-1998-06-02-11-57.html    10    0    0    0    6
lyco-happy-1998-06-05-11-09.html    10    0    0    0    6
lyco-happy-1998-06-08-11-14.html    10    0    0    0    6
lyco-happy-1998-06-11-11-09.html    10    0    0    0    6
lyco-happy-1998-06-14-11-15.html    10    0    0    0    6
lyco-happy-1998-06-17-11-12.html    10    0    0    0    6
lyco-happy-1998-06-20-10-19.html    10    0    0    0    5
lyco-happy-1998-06-21-11-12.html    10    0    0    0    5
lyco-happy-1998-06-24-11-11.html    10    0    0    0    5
lyco-happy-1998-06-27-11-07.html    10    0    0    0    5
lyco-happy-1998-06-30-11-17.html    10    0    0    0    5
lycos2, lycos_pages/pages2
lyco-happy-1998-07-04-16-33.html    10    0    0    0    5
lyco-happy-1998-07-07-11-17.html    10    0    0    0    5
lyco-happy-1998-07-10-11-13.html    10    0    0    0    5
lyco-happy-1998-07-13-11-09.html    10    0    0    0    5
lyco-happy-1998-07-16-11-08.html    10    0    0    0    5
lyco-happy-1998-07-18-11-23.html    10    0    0    0    5
lyco-happy-1998-07-21-11-08.html    10    0    0    0    5
lyco-happy-1998-07-24-11-13.html    10    0    0    0    5
lyco-happy-1998-07-27-11-23.html    10    0    0    0    5
lyco-happy-1998-07-30-11-07.html    10    0    0    0    5
lyco-happy-1998-08-02-11-15.html    10    0    0    0    5
lyco-happy-1998-08-05-11-07.html    10    0    0    0    5
lyco-happy-1998-08-08-11-12.html    10    0    0    0    5
lyco-happy-1998-08-11-11-09.html    10    0    0    0    5
lyco-happy-1998-08-15-11-11.html    10    0    0    0    5
lyco-happy-1998-08-18-11-09.html    10    0    0    0    5
lyco-happy-1998-08-21-12-29.html    10    0    0    0    5
lyco-happy-1998-08-24-11-14.html    10    0    0    0    5
lyco-happy-1998-08-27-11-05.html    10    0    0    0    5
lyco-happy-1998-08-30-11-36.html    10    0    0    0    5
lyco-happy-1998-09-02-11-14.html    10    0    0    0    5
lyco-happy-1998-09-05-11-15.html    10    0    0    0    5
```

```
lyco-happy-1998-09-09-11-40.html    10    0    0    0    5
lyco-happy-1998-09-12-11-31.html    10    0    0    0    5
lyco-happy-1998-09-15-11-24.html    10    0    0    0    5
lyco-happy-1998-09-18-11-13.html    10    0    0    0    5
lyco-happy-1998-09-21-11-33.html    10    0    0    0    5
lyco-happy-1998-09-24-11-12.html    10    0    0    0    5
lyco-happy-1998-09-27-11-36.html    10    0    0    0    5
lyco-happy-1998-09-30-11-29.html    10    0    0    0    5

Inkorrekt:
lycos2, lycos_pages/pages1
lyco-happy-1998-04-12-11-08.html    0    0    0    0    0
lyco-happy-1998-04-15-11-08.html    0    0    0    0    0
lyco-happy-1998-04-18-11-08.html    0    0    0    0    0
lyco-happy-1998-04-21-16-20.html    0    0    0    0    0
lyco-happy-1998-04-23-11-06.html    0    0    0    0    0
lyco-happy-1998-04-26-11-08.html    0    0    0    0    0
lyco-happy-1998-04-29-11-13.html    0    0    0    0    0
lyco-happy-1998-04-30-11-10.html    0    0    0    0    0
lyco-happy-1998-05-02-11-26.html    0    0    0    0    0
lyco-happy-1998-05-05-13-42.html    0    0    0    0    0
lyco-happy-1998-05-08-11-11.html    0    0    0    0    0
lyco-happy-1998-05-11-11-09.html    0    0    0    0    0
lyco-happy-1998-05-16-10-05.html    0    0    0    0    0
lyco-happy-1998-05-23-13-25.html    0    0    0    0    0
lyco-happy-1998-05-26-17-10.html    0    0    0    0    0
lyco-happy-1998-05-29-11-13.html    0    0    0    0    0
lyco-happy-1998-06-02-11-57.html    0    0    0    0    0
lyco-happy-1998-06-05-11-09.html    0    0    0    0    0
lyco-happy-1998-06-08-11-14.html    0    0    0    0    0
lyco-happy-1998-06-11-11-09.html    0    0    0    0    0
lyco-happy-1998-06-14-11-15.html    0    0    0    0    0
lyco-happy-1998-06-17-11-12.html    0    0    0    0    0
lyco-happy-1998-06-20-10-19.html    0    0    0    0    0
lyco-happy-1998-06-21-11-12.html    0    0    0    0    0
lyco-happy-1998-06-24-11-11.html    0    0    0    0    0
lyco-happy-1998-06-27-11-07.html    0    0    0    0    0
lyco-happy-1998-06-30-11-17.html    0    0    0    0    0
lycos1, lycos_pages/pages2
lyco-happy-1998-07-04-16-33.html    0    0    0    0    0
lyco-happy-1998-07-07-11-17.html    0    0    0    0    0
lyco-happy-1998-07-10-11-13.html    0    0    0    0    0
lyco-happy-1998-07-13-11-09.html    0    0    0    0    0
lyco-happy-1998-07-16-11-08.html    0    0    0    0    0
lyco-happy-1998-07-18-11-23.html    0    0    0    0    0
lyco-happy-1998-07-21-11-08.html    0    0    0    0    0
lyco-happy-1998-07-24-11-13.html    0    0    0    0    0
lyco-happy-1998-07-27-11-23.html    0    0    0    0    0
lyco-happy-1998-07-30-11-07.html    0    0    0    0    0
lyco-happy-1998-08-02-11-15.html    0    0    0    0    0
lyco-happy-1998-08-05-11-07.html    0    0    0    0    0
lyco-happy-1998-08-08-11-12.html    0    0    0    0    0
lyco-happy-1998-08-11-11-09.html    0    0    0    0    0
lyco-happy-1998-08-15-11-11.html    0    0    0    0    0
lyco-happy-1998-08-18-11-09.html    0    0    0    0    0
lyco-happy-1998-08-21-12-29.html    0    0    0    0    0
lyco-happy-1998-08-24-11-14.html    0    0    0    0    0
lyco-happy-1998-08-27-11-05.html    0    0    0    0    0
lyco-happy-1998-08-30-11-36.html    0    0    0    0    0
lyco-happy-1998-09-02-11-14.html    0    0    0    0    0
lyco-happy-1998-09-05-11-15.html    0    0    0    0    0
lyco-happy-1998-09-09-11-40.html    0    0    0    0    0
lyco-happy-1998-09-12-11-31.html    0    0    0    0    0
lyco-happy-1998-09-15-11-24.html    0    0    0    0    0
lyco-happy-1998-09-18-11-13.html    0    0    0    0    0
lyco-happy-1998-09-21-11-33.html    0    0    0    0    0
lyco-happy-1998-09-24-11-12.html    0    0    0    0    0
lyco-happy-1998-09-27-11-36.html    0    0    0    0    0
lyco-happy-1998-09-30-11-29.html    0    0    0    0    0
```

A.3 Ergebnisse bei MetaCrawler (results_metacrawler.txt)

```
Korrekt:
metacrawler1, metacrawler_pages/pages1
meta-happy-1998-04-12-11-08.html     20     1      1      2      7
meta-happy-1998-04-15-11-08.html     20     2      2      3      6
meta-happy-1998-04-18-11-08.html     20     1      1      2      6
meta-happy-1998-04-21-16-20.html     20     1      1      2      6
meta-happy-1998-04-23-11-06.html     20     1      1      2      5
meta-happy-1998-04-26-11-08.html     20     1      1      1      5
meta-happy-1998-04-29-11-13.html     20     2      2      3      5
meta-happy-1998-04-30-11-10.html     20     1      1      2      6
meta-happy-1998-05-02-11-26.html     20     1      1      2      6
meta-happy-1998-05-05-13-42.html     20     2      2      2      4
meta-happy-1998-05-08-11-11.html     20     2      2      1      6
meta-happy-1998-05-11-11-09.html     20     2      2      0      5
meta-happy-1998-05-16-10-05.html     17     2      3      0      6
metacrawler2, metacrawler_pages/pages2
meta-happy-1998-05-23-13-24.html     21     1      1      2      6
meta-happy-1998-05-26-17-09.html     21     1      1      3      5
meta-happy-1998-05-29-11-13.html     21     1      1      4      4
meta-happy-1998-06-02-11-57.html     21     1      1      3      5
meta-happy-1998-06-05-11-09.html     21     1      1      3      5
meta-happy-1998-06-08-11-14.html     21     1      1      2      5
meta-happy-1998-06-11-11-09.html     21     1      1      2      7
meta-happy-1998-06-14-11-15.html     21     1      1      2      6
meta-happy-1998-06-17-11-12.html     21     1      1      4      7
meta-happy-1998-06-20-08-58.html     21     1      1      4      4
meta-happy-1998-06-21-11-11.html     21     1      1      3      4
meta-happy-1998-06-24-11-11.html     21     3      1      4      7
meta-happy-1998-06-27-11-07.html     21     1      1      3      3
meta-happy-1998-06-30-11-17.html     21     1      1      3      5
meta-happy-1998-07-04-16-33.html     21     1      1      2      3
meta-happy-1998-07-07-11-17.html     21     1      1      3      9
metacrawler3, metacrawler_pages/pages3
meta-happy-1998-07-10-11-13.html     20     1      1      2      4
meta-happy-1998-07-13-11-09.html     19     2      3      2      2
meta-happy-1998-07-16-11-08.html     20     2      2      3      5
meta-happy-1998-07-18-11-23.html     14     2      3      3      4
meta-happy-1998-07-21-11-08.html     20     2      2      3      3
meta-happy-1998-07-24-11-13.html     19     2      3      3      5
meta-happy-1998-07-27-11-23.html     20     2      2      3      5
meta-happy-1998-07-30-11-07.html     14     2      3      3      5
meta-happy-1998-08-02-11-15.html     14     1      2      1      5
meta-happy-1998-08-05-11-07.html     14     1      2      2      5
meta-happy-1998-08-08-11-12.html     14     1      2      2      6
meta-happy-1998-08-11-11-09.html     14     2      3      2      5
meta-happy-1998-08-15-11-11.html     14     2      3      2      5
meta-happy-1998-08-18-11-09.html     14     1      2      3      5
meta-happy-1998-08-21-12-27.html     14     1      2      5      5
meta-happy-1998-08-24-11-14.html     14     1      2      5      5
meta-happy-1998-08-27-11-05.html     14     2      3      5      5
meta-happy-1998-08-30-11-35.html     14     1      2      6      6
meta-happy-1998-09-02-11-14.html     14     2      3      6      3
meta-happy-1998-09-05-11-15.html     14     1      2      5      6
meta-happy-1998-09-09-11-40.html     20     2      2      6      7
meta-happy-1998-09-12-11-31.html     20     2      2      8      7
meta-happy-1998-09-15-11-24.html     20     2      2      8      8
meta-happy-1998-09-18-11-13.html     20     2      2      4      5
meta-happy-1998-09-21-11-32.html     20     2      2      6      6
meta-happy-1998-09-24-11-12.html     20     2      2      6      6
meta-happy-1998-09-27-11-36.html     20     2      2      6      6
meta-happy-1998-09-30-11-29.html     20     1      1      6      6

Inkorrekt:
metacrawler2, metacrawler_pages/pages1
meta-happy-1998-04-12-11-08.html     0      0      0      0      0
meta-happy-1998-04-15-11-08.html     1      1      1      1      1
meta-happy-1998-04-18-11-08.html     0      0      0      0      0
meta-happy-1998-04-21-16-20.html     0      0      0      0      0
```

```
meta-happy-1998-04-23-11-06.html    0     0     0     0     0
meta-happy-1998-04-26-11-08.html    0     0     0     0     0
meta-happy-1998-04-29-11-13.html    0     0     0     0     0
meta-happy-1998-04-30-11-10.html    0     0     0     0     0
meta-happy-1998-05-02-11-26.html    0     0     0     0     0
meta-happy-1998-05-05-13-42.html    1     1     1     1     1
meta-happy-1998-05-08-11-11.html    1     1     1     1     1
meta-happy-1998-05-11-11-09.html    1     1     1     1     1
meta-happy-1998-05-16-10-05.html    1     1     1     1     1
metacrawler1, metacrawler_pages/pages2
meta-happy-1998-05-23-13-24.html    20    20    2     1     19
meta-happy-1998-05-26-17-09.html    20    20    2     2     19
meta-happy-1998-05-29-11-13.html    20    20    2     3     19
meta-happy-1998-06-02-11-57.html    20    20    2     2     20
meta-happy-1998-06-05-11-09.html    20    20    2     2     20
meta-happy-1998-06-08-11-14.html    20    20    2     1     19
meta-happy-1998-06-11-11-09.html    20    20    2     1     19
meta-happy-1998-06-14-11-15.html    20    20    2     1     19
meta-happy-1998-06-17-11-12.html    20    20    2     3     19
meta-happy-1998-06-20-08-58.html    20    20    2     3     19
meta-happy-1998-06-21-11-11.html    20    20    2     2     19
meta-happy-1998-06-24-11-11.html    20    20    2     3     19
meta-happy-1998-06-27-11-07.html    20    20    2     2     19
meta-happy-1998-06-30-11-17.html    20    20    2     2     19
meta-happy-1998-07-04-16-33.html    20    20    2     1     19
meta-happy-1998-07-07-11-17.html    20    20    2     2     19
metacrawler3, metacrawler_pages/pages2
meta-happy-1998-05-23-13-24.html    20    20    2     1     19
meta-happy-1998-05-26-17-09.html    20    20    2     2     19
meta-happy-1998-05-29-11-13.html    20    20    2     3     19
meta-happy-1998-06-02-11-57.html    20    20    2     2     20
meta-happy-1998-06-05-11-09.html    20    20    2     2     20
meta-happy-1998-06-08-11-14.html    20    20    2     1     19
meta-happy-1998-06-11-11-09.html    20    20    2     1     19
meta-happy-1998-06-14-11-15.html    20    20    2     1     19
meta-happy-1998-06-17-11-12.html    20    20    2     3     19
meta-happy-1998-06-20-08-58.html    20    20    2     3     19
meta-happy-1998-06-21-11-11.html    20    20    2     2     19
meta-happy-1998-06-24-11-11.html    20    20    2     3     19
meta-happy-1998-06-27-11-07.html    20    20    2     2     19
meta-happy-1998-06-30-11-17.html    20    20    2     2     19
meta-happy-1998-07-04-16-33.html    20    20    2     1     19
meta-happy-1998-07-07-11-17.html    20    20    2     2     19
metacrawler2, metacrawler_pages/pages3
meta-happy-1998-07-10-11-13.html    0     0     0     0     0
meta-happy-1998-07-13-11-09.html    1     1     1     1     1
meta-happy-1998-07-16-11-08.html    1     1     1     1     1
meta-happy-1998-07-18-11-23.html    1     1     1     1     1
meta-happy-1998-07-21-11-08.html    1     1     1     1     1
meta-happy-1998-07-24-11-13.html    1     1     1     1     1
meta-happy-1998-07-27-11-23.html    1     1     1     1     1
meta-happy-1998-07-30-11-07.html    1     1     1     1     1
meta-happy-1998-08-02-11-15.html    1     1     1     1     1
meta-happy-1998-08-05-11-07.html    1     1     1     1     1
meta-happy-1998-08-08-11-12.html    1     1     1     1     1
meta-happy-1998-08-11-11-09.html    1     1     1     1     1
meta-happy-1998-08-15-11-11.html    1     1     1     1     1
meta-happy-1998-08-18-11-09.html    1     1     1     1     1
meta-happy-1998-08-21-12-27.html    1     1     1     1     1
meta-happy-1998-08-24-11-14.html    1     1     1     1     1
meta-happy-1998-08-27-11-05.html    1     1     1     1     1
meta-happy-1998-08-30-11-35.html    1     1     1     1     1
meta-happy-1998-09-02-11-14.html    1     1     1     1     1
meta-happy-1998-09-05-11-15.html    1     1     1     1     1
meta-happy-1998-09-09-11-40.html    1     1     1     1     1
meta-happy-1998-09-12-11-31.html    1     1     1     1     1
meta-happy-1998-09-15-11-24.html    0     0     0     0     0
meta-happy-1998-09-18-11-13.html    1     1     1     1     1
meta-happy-1998-09-21-11-32.html    1     1     1     1     1
meta-happy-1998-09-24-11-12.html    1     1     1     1     1
```

meta-happy-1998-09-27-11-36.html	0	0	0	0	0
meta-happy-1998-09-30-11-29.html	0	0	0	0	0

Anhang B

Die wichtigsten Fremdwörter in der Arbeit

Black-box testing: Funktionelles Testen; nur die Ein- und Ausgabe des zu testenden Systems werden in Betracht genommen, und nicht seine innere Struktur

Feature: Eigenschaft, auch oft als positive Eigenschaft oder charakteristische Eigenschaft

Frame: Rahmen; ein unsichtbares GUI-Element, das andere Widgets gruppiert

Layout: Anordnung, Auslegung; hier: die planare Auslegung (und Formatierungen) von HTML-Seiten

Mediator: Vermittler; hier: eine Komponente zur Integration heterogener Informationsquellen (siehe Abschnitt 1.2.3)

Rapid prototyping: Schnelle Implementierung eines Prototyps, mit dem man dann schon diverse Tests durchführen kann

RegExp, Regular expression: Regulärer Ausdruck

Site: Stelle; hier: Menge von Web-Seiten, die gemeinsam eine HTML-Schnittstelle (z. B. einer Firma, einer Institution, oder einer Datenbank) bilden

Soft: Weich; heuristische Methoden, deren Korrektheit nicht beweisbar ist, werden oft "Soft methods" genannt; eine Bedingung, die nicht ganz genau, sondern nur ungefähr erfüllt werden muss, wird auch so genannt

Tag: Ein Element einer HTML-Seite, das nicht unmittelbar angezeigt wird, sondern Informationen für den Browser bereitstellt; z. B. Formatierungsinformationen für den Text

Ticker: Börsentelegraf

Timeout: Zeitüberschreitung, Zeitabschaltung, Abfallzeit

White-box testing: Strukturtest; Strukturtesten basiert auf genauer Kenntnis des Softwareentwicklung und deswegen wird es hauptsächlich vom Programmierer, auf dem unit level benutzt, um zu prüfen, ob die Implementierung mit der Spezifikation übereinstimmt [10].

Widget: GUI-Element, z. B. Menü, Liste, Druckknopf etc.

Wrapper: Hülle; in der Softwaretechnik: eine Komponente, die die Funktionalität einer bereits existierenden Komponente um neue Funktionalität erweitert; hier: jene Komponente der Mediatorarchitektur, die die Anbindung von Informationsquellen an den Mediator ermöglicht (siehe Abschnitt 1.2.2)

Anhang C

Die wichtigsten Abkürzungen in der Arbeit

BDD	Binary Decision Diagram
CDM	Common Data Model
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DNS	Domain Name Service
DSS	Decision Support System
EDI	Electronic Data Interchange
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IIS	Information Integration System
IP	Internet Protocol
KAMEL	KArlsruhe MEdiator Language
KOMET	Karlsruhe Open MEdiator Technology
MDA	Model Driven Architecture
OMG	Object Management Group
PEDGUI	Printed Embedded Data Graphical User Interface
PERL	Practical Extraction and Report Language
QoS	Quality of Service
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
TCL	Tool Command Language
TMR	Triple Modular Redundancy
UML	Unified Modeling Language
URL	Uniform Resource Locator
WSDL	Web Services Description Language
WWW	World Wide Web
XML	eXtensible Markup Language

Literaturverzeichnis

[1] K. Arnold, J. Gosling, und D. Holmes: *The Java programming language*. The Java Series. Addison-Wesley, dritte Auflage, 2000.

[2] I. Bach: *Formális nyelvek*. TypoTeX, Budapest, 2001.

[3] T. Berners-Lee und D. Connolly: RFC 1866: Hypertext Markup Language 2.0. `http://www.ics.uci.edu/pub/ietf/html/rfc1866.txt`, November 1995.

[4] J. Calmet, S. Jekutsch, und J. Schü: A generic query-translation framework for a mediator architecture. In *13th International Conference on Data Engineering, Birmingham*, 1997.

[5] J. Calmet, S. Jekutsch, *et al.*: KOMET – A system for the integration of heterogeneous information sources. In *10th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, 1997.

[6] J. Calmet und P. Kullmann: Meta web search with KOMET. In *International Joint Conference on Artificial Intelligence, Stockholm*, 1999.

[7] R. C. Carrasco und J. Oncina: Learning stochastic regular grammars by means of a state merging method. In *Proceedings of the 2nd International Colloquium ICGI-94*, Seiten 106–118, 1994.

[8] W. W. Cohen: Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of ACM SIGMOD-98*, 1998.

[9] W. W. Cohen: Recognizing structure in web pages using similarity queries. In *AAAI-99 (Orlando)*, 1999.

[10] A. Fetke, C. Schroeder, und P. Kruzynski: Cleanroom. Referat im Seminar „Vorgehensmodelle für die Entwicklung sicherheitskritischer Software-Systeme“ im WS 96/97 an der TU-Berlin, `http://www.user.cs.tu-berlin.de/~przemek/cleanroom.html`.

[11] T. Goan, N. Benson, und O. Etzioni: A grammar inference algorithm for the world wide web. In *Proc. of the 1996 AAAI Spring Symposium on Machine Learning in Information Access (MLIA), Stanford, CA*. AAAI Press, 1996.

[12] D. L. Hecht: Printed Embedded Data Graphical User Interfaces. *IEEE Computer*, 34(3):Seiten 47–55, March 2001.

[13] K. Hernaut: Zukunftssicherung durch Wissensmanagement. Einführungsvortrag in der 13. Bayerischen Industrial-Engineering-Fachtagung und dem 4. Bayerischen SUPER-Symposium, Juni 1999.

[14] C. Hsu und M. Dung: Generating finite-state transducers for semi-structured data extraction from the web. *Journal of Information Systems*, 23(8):Seiten 521–538, 1998.

[15] IAKS Calmet: KAMEL and KOMET: a project in knowledge integration. `http://iaks-www.ira.uka.de/iaks-calmet/research/kamel-komet`.

[16] Intelligent Information Integration, Inofficial home page: `http://www.tzi.org/grp/i3`.

[17] S. Jekutsch: *Entwurf und Implementierung eines generischen Anfrageübersetzers zur Integration heterogener Informationsquellen*. Diplomarbeit, Universität Karlsruhe, Fakultät für Informatik, Institut für Algorithmen und Kognitive Systeme, Juni 1996.

[18] L. P. Kaelbling, M. L. Littmann, und A. W. Moore: Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:Seiten 237–285, May 1996.

[19] G. J. Klir, U. S. Clair, und B. Yuan: *Fuzzy set theory – foundations and applications*. Prentice Hall, 1997.

[20] C. A. Knoblock, K. Lerman, *et al.*: Accurately and reliably extracting data from the web: a machine learning approach. *Data Engineering Bulletin*.

[21] N. Kushmerick: Regression testing for wrapper maintenance. In *AAAI-99 (Orlando)*, Seiten 74–79, 1999.

[22] N. Kushmerick: Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence Journal*, 118(1-2):Seiten 15–68, 2000. (Special issue on Intelligent Internet Systems).

[23] N. Kushmerick: Wrapper verification. *World Wide Web Journal*, 3(2):Seiten 79–94, 2000. (Special issue on Web Data Management).

[24] S. Lawrence, D. M. Pennock, *et al.*: Persistence of web references in scientific research. *IEEE Computer*, 34(2):Seiten 26–31, February 2001.

[25] K. Lerman und S. Minton: Learning the common structure of data. In *AAAI-2000 (Austin)*, 2000.

[26] I. Muslea: Extraction patterns for information extraction tasks: a survey. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[27] I. Muslea, S. Minton, und C. Knoblock: Co-testing: Selective sampling with redundant views. In *Proceedings of the 17th National Conference on Artificial Intelligence AAAI-2000*, Seiten 609–614, 2000.

[28] I. Muslea, S. Minton, und C. Knoblock: Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 2001.

[29] J. Nielsen: Kill the 53-day meme. Alertbox `http://www.useit.com/alertbox/9509.html`, September 1995.

[30] D. J. Power: What is a Decision Support System? *The On-Line Executive Journal for Data-Intensive Decision Support*, 1(3), October 1997.

[31] B. Randell, J.-C. Laprie, *et al.* (Herausgeber): *Predictably dependable computing systems*. Springer-Verlag, Berlin, 1995.

[32] G. Salton (Herausgeber): *Automatic text processing: the transformation, analysis and retrieval of information by computer*. Addison Wesley, Reading, Massachusetts, 1989.

[33] Search Engine Watch: `http://www.searchenginewatch.com`.

[34] S. Soderland: Learning extraction rules for semi-structured and free text. *Machine Learning*, 34:Seiten 233–272, 1999.

[35] C. Stephanidis und M. Sfyrakis: Current trends in man-machine interfaces. In *ECSC-EC-EAEC, Brussels*, 1995.

[36] V. S. Subrahmanian, S. Adalı, *et al.*: HERMES: A heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1995.

[37] S. Trcek: *Entwurf und Implementierung eines Datenmodells zur Integration heterogener Informationsquellen*. Diplomarbeit, Universität Karlsruhe, Fakultät für Informatik, Institut für Algorithmen und Kognitive Systeme, Juni 1996.

[38] U. Ulm: Praktikum für Studenten der Medizin und Zahnmedizin: Klinische Chemie, Hämatologie, WS 1999/2000. `http://www.uni-ulm.de/klinik/klinchem/Praktikumsheft.pdf`.

[39] L. Wall, T. Christiansen, und J. Orwant: *Programming Perl*. O'Reilly, dritte Auflage, July 2000.

[40] G. Wiederhold: Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):Seiten 38–49, March 1992.

[41] G. Wiederhold: Value-added mediation in large-scale information systems. In *Proceedings of the IFIP-DS6 Conference*, 1995.

[42] G. Wiederhold und M. Genesereth: The conceptual basis for mediation services. *IEEE Expert*, 12(5):Seiten 38–47, September-October 1997.