Studienarbeit

Integration von SMES und Harvest in GETESS[1]

Lehrstuhl Datenbank- und Informationssysteme
Fachbereich Informatik
Universität Rostock

Ilvio Bruder

Betreuer
Prof. Dr. Andreas Heuer
Dr. Ing. Antje Düsterhöft

Rostock, den 15. Januar 2000

[1]GETESS – German Text Exploitation and Search System

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die vorliegende Studienarbeit[1] befaßt sich mit konzeptuellen Möglichkeiten zur Integration von SMES (ein natürlichsprachlicher Parser) und Harvest (eine Internetsuchmaschine) im Gathering–Prozeß (Prozeß des Sammelns von Informationen). Dazu wird in diesem Kapitel das Projekt GETESS vorgestellt und die Problematik dieser Arbeit innerhalb von GETESS eingeordnet. Weiterhin werden die Problemstellungen näher erläutert und Realisierungsmöglichkeiten aufgezeigt.
In den weiteren Kapitel werden die zu integrierenden Komponenten vorgestellt und konzeptionell Integrationsmöglichkeiten diskutiert. Ein Ausblick mit einer umfangreichen Problemanalyse wird diese Arbeit abschließen.

1.1 Das Projekt GETESS

GETESS (siehe auch [BPW+98] und [MPP+99]) ist ein dreijähriges Projekt des Bundesministerium für Bildung und Forschung (kurz BMBF). Ziel des Projektes ist die Entwicklung eines intelligenten Werkzeuges zur Bereitstellung und Recherche von Informationen im Internet. Darunter stellt man sich am einfachsten eine Internet–Suchmaschine mit erweiterter Funktionalität vor.

Es wird nachfolgend immer vom Begriff der "Suchmaschine" ausgegangen. Im herkömmlichen Sinne bezeichnet dies eine Internet–Suchmaschine, wie Altavista, Lycos oder HotBot. Gegenwärtig existieren vor allem Suchmaschinen, die sehr viele Informationen verarbeiten, diese aber sehr ungeeignet präsentieren und dem Nutzer kaum Unterstützung anbieten, sein Informationsbedürfnis zum Erfolg zu führen. Die Verbesserung dieses Problems ist ein wichtiger Grundgedanke des Projektes GETESS. Es werden neue intelligente Verfahren entwickelt, die dem Nutzer eine effektivere und vom System unterstützende Suche ermöglichen sollen.
Es haben sich folgende Anforderungen (aus [BPW+98]) an das zu realisierende System herauskristallisiert:

1. Es soll ein einfach zu bedienendes System sein.
2. Der Nutzer soll die Möglichkeit haben natürlichsprachliche Anfragen zu stellen.
3. Der Nutzer soll zur besseren Kommunikation in einer Ontologie navigieren können.
4. Die Präsentation der Suchergebnisse soll in geeigneter und nachvollziehbarer Form erfolgen.
5. Moderne und attraktive Multimediatechniken kommen zum Einsatz.

Projektpartner + Aufgaben
An diesem Projekt sind vier Projektpartner mit verschiedenen Forschungsschwerpunkten aus der Bundesrepublik beteiligt. Im folgenden werden neben den Partnern auch deren Aufgaben innerhalb des Projekts genannt. Die Zusammenhänge der Aufgaben werden weiter unten mit der Architekturskizze von GETESS gezeigt.

[1] Diese Studienarbeit wurde in der alten deutschen Rechtschreibung verfaßt.

Die **GECKO mbH Rostock** mit Geschäftsführer Siegfried Melzig ist eine Software– und Provider–Firma aus Rostock und Projekt–Koordinator dieses Projektes. Bei der GECKO mbH werden eine Dialogkomponente zur Interaktion mit dem Benutzer entwickelt und eine Common Knowledge Base[2] (CKB) erstellt bzw. realisiert.
Das Deutsche Forschungsinstitut für Künstliche Intelligenz (kurz **DFKI**) in **Saarbrücken** mit Prof. Hans Uszkoreit werden mit computerlinguistischen Techniken aus den Informationsquellen wichtige Daten zur Weiterverarbeitung extrahieren und eine multilinguale Verarbeitung natürlicher Sprache in das Projekt einbringen.
Die Fachgruppe Wissensmodellierung und wissensbasierte Systeme (kurz **AIFB Uni Karlsruhe**) unter Prof. Rudi Studer an der Universität Karlsruhe ist für den Entwurf und Realisierung einer Domänen–Ontologie verantwortlich und stellt den Zusammenhang zwischen linguistischen und ontologischen Aspekten her.
Als vierter Partner ist der Lehrstuhl Datenbank– und Informationssysteme an der Universität Rostock (**DBIS Uni Rostock**) unter Prof. Andreas Heuer für die Entwicklung eines Internet-Sammelagenten (Gatherer) für das System, die Speicherung der Daten in eine Datenbank und den Zugriff auf die Daten der verschiedenen Datenquellen verantwortlich.

GETESS-Architektur
In der Abbildung 1.1 wird die Komponentenarchitektur von GETESS gezeigt. Die Komponenten des Projektes sind in der Skizze verallgemeinert und in größere Abschnitte zusammengefaßt dargestellt. Aufgrund der Übersichtlichkeit sind keine Verbindungen der Komponenten in der Skizze enthalten. Die Beziehungen werden durch die Anordnung der Komponenten ausgedrückt.

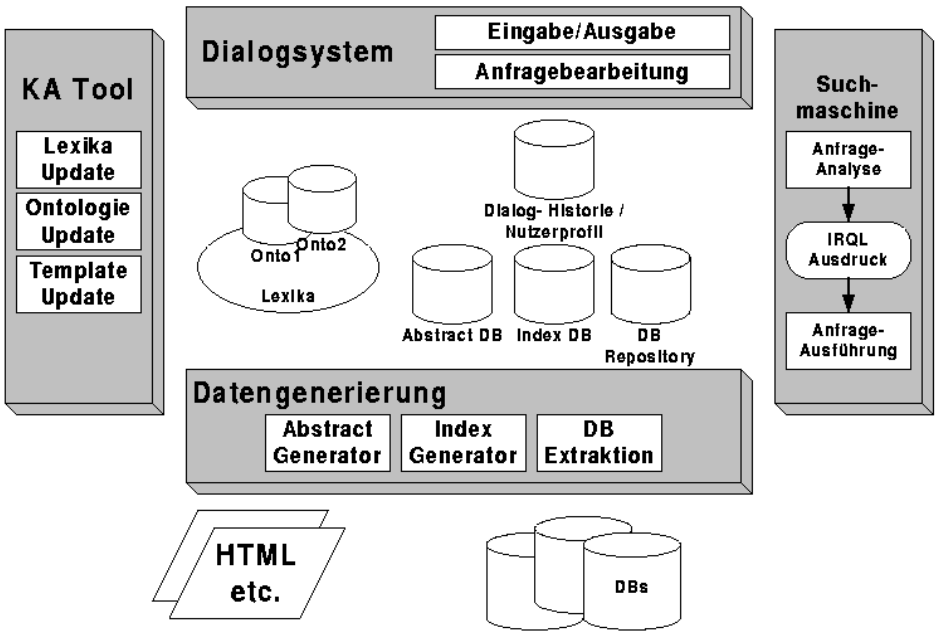

Abbildung 1.1: Architekturskizze der Komponenten von GETESS

Umrahmt durch die Hauptkomponenten werden in der Mitte der Architekturskizze die Daten und die Datenhaltung dargestellt. Zum einen sind hier die Lexika für die natürlichsprachlichen Erken-

[2]Common Knowledge Base — Wissensbank für Allgemeinwissen.

nungsmodule und die Ontologien für das domänenspezifische Wissen untergebracht. Und zum anderen liegen hier die eingesammelten und aufbereiteten Daten aus dem Internet als abstracts[3] (extrahierte domänenspezifische Daten eines Dokumentes in zusammengefaßter Form), als indizierte Daten oder als Datenbank–Repository (für externe Internetdatenbanken) vor. Außerdem werden hier weitere interne Daten, wie z.B. die Benutzerprofile, vorgehalten.

Auf der linken Seite sind die Knowledge Aquisition–Tools (kurz KA–Tools) zu finden. Diese Tools dienen zum Aufbau, Update und Löschen von Konzepten in den domänenspezifischen Datenquellen (Ontologien, Lexika).

Unterhalb der Datenhaltung sind die Komponenten zusammengefaßt, die die Informationen der Dokumente und anderer Datenquellen aus dem Internet extrahieren, aufbereiten und geeignet der Speicherung übergeben. Beim Aufbereiten der Daten liegt das Hauptaugenmerk auf der Abstract–Generierung, da hier mittels natürlichsprachlichen Parsern und Domänwissen nicht nur lexikalisch und syntaktisch, sondern auch semantisch die Datenquellen erfaßt werden können.

Auf der rechten Seite ist der Block "Suchmaschine" angeordnet. Mit "Suchmaschine" ist hier — im Gegensatz zu oben — der eigentliche Zugriff auf die extrahierten Daten gemeint. Dies beinhaltet die Anfrageanalyse, die Umwandlung in eine geeignete interne Anfragesprache (sie sollte alle Datenquellen und deren spezifischen Funktionen effizient ausnutzen können) und die Ausführung der Anfrage.

Oben angeordnet befindet sich das Dialogsystem. Darüber wird der Kontakt zum Nutzer (Anfragen–Steller)[4] der GETESS–Suchmaschine realisiert. Das Dialogsystem versucht natürlichsprachlich mit dem Nutzer zu kommunizieren. Dabei werden alle internen Datenbanken benutzt, um das Informationsbedürfnis des Nutzers zu stillen. Durch Dialoge kann der Nutzer zu seiner Information geführt werden, falls das erste Suchergebnis den Nutzer noch nicht zufrieden gestellt hat.

1.2 Einordnung der Aufgabe in GETESS

Um eine Einordnung der Aufgabe dieser Studienarbeit in GETESS und in die Teilaufgabe der Universität Rostock vorzunehmen, werden die Rostocker Aufgaben etwas genauer betrachtet, und anschließend wird eine Einordnung der Studienarbeitsproblematik vorgenommen.

Arbeitspakete der Universität Rostock
Ausgehend von der in 1.1 beschriebenen Komponentenarchitektur werden in diesem Abschnitt die Aufgaben der Universität Rostock dargestellt. Dazu sind in der Abbildung 1.2 die Rostocker Aufgabenpakete in die Mitte fokussiert worden. An den Rändern befinden sich die Arbeitspakete der anderen drei Projektpartner.

Es werden an der Universität Rostock im GETESS–Projekt im wesentlichen drei Teile bearbeitet. Das ist zum **ersten** die Konzeption und Realisierung eines geeigneten Sammelagenten (Gatherer). Dieser Sammelagent soll Informationsseiten im Internet sammeln und das enthaltende Wissen mit Hilfe der linguistischen und domänenspezifischen Analyse (in Abb. 1.2 der NL-Parser) herausfiltern. Diese Daten werden dann aufbereitet an die Speicherungskomponente übergeben.

Im **zweiten** Aufgabenteil, der Speicherung des Wissens, dieser Projektgruppe befaßt man sich mit dem optimalen Einbringen der Informationsmenge in die Datenbanken. Hier spielen Anforderungen, wie Effizienz, Sicherheit und Änderbarkeit eine große Rolle. Das Speicherschema wird im Einklang mit der Ontologie aufgebaut, damit die Abbildung zwischen Domänwissen und abgespeichertem Wissen erhalten bleibt.

[3] Abstract — in diesem Kontext ein Begriff aus der Terminologie der Computerlinguistik.
[4] Eine weitere Nutzergruppe stellen die Betreiber der Suchmaschine dar. Ihre Schnittstelle ist im wesentlichen über die KA-Tools definiert.

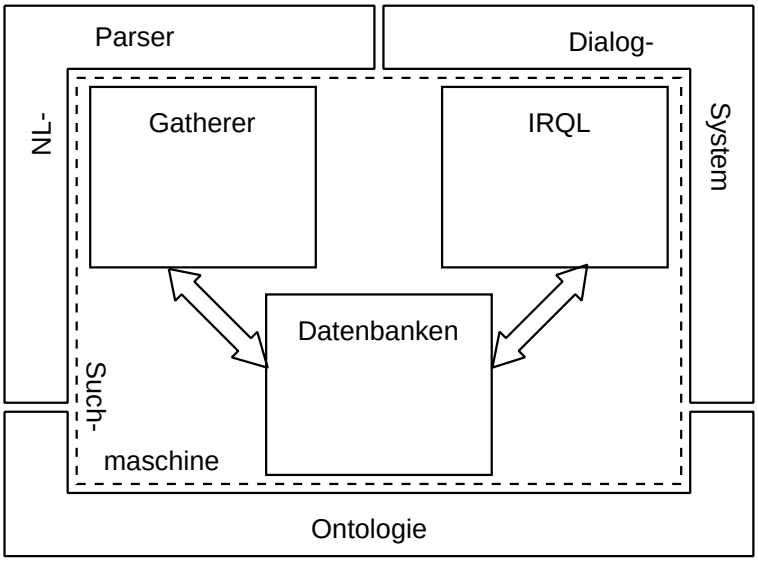

Abbildung 1.2: Darstellung von GETESS mit Fokus auf die Arbeitspakete der Universität Rostock

Der **dritte** Aufgabenteil ist zuständig für den Zugriff auf die Daten in den Datenbanken. Hier besteht das Problem, Information Retrieval–Suchtechniken und Datenbank–Anfragetechniken in einer Sprachdefinition zu vereinigen, um die Datenbankspeicherung mit den in 1.1 genannten Anforderungen zu unterstützen. Dazu entsteht momentan eine neuartige Anfragesprache namens Information Retrieval Query Language (kurz IRQL), die unter anderem die genannten Eigenschaften besitzen wird. Über diese Anfragesprache wird die Schnittstelle zum Dialogsystem geregelt.

Der Gatherer
Innerhalb der vorgestellten Aufgaben der Universität Rostock im vorigen Abschnitt ist die Studienarbeit im Aufgabenbereich des Sammelagenten (Gatherer) angesiedelt. Der Sammelagent ist die Komponente, die Daten aus den Quellen des Internets einsammelt. Dazu gehört auch, daß die gesammelten Daten aufbereitet werden. Vorgesehen ist, daß ein Abstract–Generierer abstracts aus den Dokumenten generiert (genaueres unter Kapitel 3). Zusätzlich sind die Dokumente über einen Index abrufbar, und externe Datenquellen, wie Internet–Datenbanken, können angefragt werden.

Ausgangspunkt für den GETESS–Gatherer stellt die SWING[5]-Suchmaschine (siehe auch [LDHM97] und [HMW99]) dar. SWING bietet als Suchmaschine neben der allgemeinen Internetsuche weitere interessante Konzepte, wie z.B. den Abonnementdienst (siehe [Por98]) oder das Einbinden externer Datenbanken. Die Grundlage für SWING ist Harvest, als einfaches, aber vollständiges Internetsuchsystem mit einer verteilten Architektur. Im folgenden wird in dieser Arbeit ausschließlich Harvest betrachtet, da weiterführende Konzepte, wie sie bei SWING entwickelt wurden, hier nicht Gegenstand der Betrachtungen sein werden.

Der Gatherer wird in dieser Arbeit mit Harvest als langjährig entwickelter Sammelagent und mit dem SMES-Client des DFKI Saarbrücken (für das natürlichsprachliche Parsing) implementiert und installiert.
Eine allgemeine Betrachtung der Studienarbeitsaufgabe findet sich im nächsten Abschnitt.

[5]Der Name kommt von **S**uchdienst für **W**WW-basierte **I**nformationssysteme der **n**ächsten **G**eneration.

1.3 Allgemeiner Betrieb eines Sammelagenten

In den vorangegangenen Abschnitten wurde GETESS genauer vorgestellt. Innerhalb dieses Projektes benötigt man einen Sammelagenten, der Informationsmaterial (zur Zeit ausschließlich HTML–Seiten) aus dem Internet sammelt und geeignet aufbereitet. Im folgenden sollen die grundlegenden Annahmen für den Betrieb eines Gatherers, die für die Realisierung relevant sind, aufgezeigt werden.

Der robot
Einen Agenten, der im Internet Informationsseiten sammelt, nennt man nach [BKKL99] "robot" oder "spider". Herkömmliche Suchmaschinen, wie Lycos, Altavista oder HotBot schicken einen oder mehrere Agenten los, um Internetseiten einzusammeln. Man gibt ihnen eine Liste mit URL's mit auf den Weg, deren Seiten (im folgenden sind allgemein HTML–Seiten gemeint, weitere Dokumententypen werden meist nicht betrachtet) sie einsammeln sollen. Weitere URL's erhalten sie durch Verweise innerhalb der HTML–Seiten. Damit diese Art des rekursiven Durchlaufens des Netzes von HTML–Seiten (als Teil des Internets) nicht ausartet (unendliches oder im Kreis laufendes Sammeln), werden bestimmte Restriktionen, wie z.B. eine vorgegebene Maximaltiefe der Suche oder die Beschränkung, URL–Verweise nur innerhalb eines Internet-Servers zu verfolgen, auf den robot angewendet. Restriktionen können auch von Serverseite aus und vom HTML–Seiten–Anbieter aufgestellt werden. Das setzt allerdings voraus, daß der robot die "Netiquette der robots"[6] beherrscht.

Der Indexierer
Mittels Information Retrieval–Techniken werden die gesammelten Daten zur Speicherung aufbereitet. Bei herkömmlichen, einfachen Suchmaschinen durchsucht eine spezielle Indexierer–Software die HTML–Seiten nach Schlüsselwörtern (keywords) und erstellt eine invertierte Liste (eine Abbildung von keywords auf Dokumente). Die Liste kann dann eventuell noch mit Informationen, wie Ranking, angereichert werden. Eine Such–Software versucht dann möglichst schnell, auf Anfragen des Nutzers die relevanten — also zur Anfrage passenden — Dokumente zu finden. Eine Einschätzung über die Qualität des Verhältnisses zwischen Anfrage und gefundenen Dokumenten erhält man mit den Eigenschaften "Recall" (Verhältnis zwischen gefundenen, relevanten Dokumenten und allen relevanten Dokumenten), "Precision" (Verhältnis zwischen gefundenen relevanten Dokumenten und allen gefundenen Dokumenten) und "Fallout" (gefundene irrelevante Dokumente zu allen irrelevanten Dokumenten).
Weitere Techniken, wie z.B. Relevance Feedback, des Information Retrieval sind unter [Fuh97] nachzulesen.

In dieser Studienarbeit wird nun untersucht, wie man, anstatt die gesammelten Informationen mit Information Retrieval–Techniken zu bearbeiten, durch natürlichsprachliche Mittel neben lexikalischen und syntaktischen auch semantische Informationen zur Wissensextraktion aus den Dokumenten erhalten kann. Dazu wird der SMES–Client (DFKI) in den Gathering–Prozeß eingebunden und jedes gesammelte Dokument zum SMES–Server geschickt. SMES liefert neben linguistischen Informationen ein abstract, in welchem semantisch das Dokument domänenbezogen beschrieben ist. Die domänenbezogene Analyse wird durch die Integration von SMES und der Ontologie aus Karlsruhe erfolgen.

Die Speicherung
Herkömmliche Suchmaschinen im Internet verwenden zur Speicherung ihrer Datensammlung viele verschiedene uneinheitliche Formate. Zu den vielversprechendsten zählen jene, die mit weiteren Indexen und "cleveren" Speichertechniken den Zugriff beschleunigen. Hier sind vor allem Indizierer auf Datenbankbasis interessant. Ausgiebige Darstellungen zu den Speicherungstechniken für Datenbanken findet man in [SH99].

Die Speicherung der abstracts des SMES wird vorrangig in Datenbanken erfolgen. Dafür spricht vor allem die gute Ausnutzung der Struktur der abstracts (semistrukturierte Daten) bei der Speicherung in Datenbanken und die Möglichkeit mächtigere Anfragesprachen, als herkömmliche Information Retrieval–Zugriffssprachen, verwenden zu können.

[6] Das Verhalten der "robots" im Einklang zum Informationsanbieter.

1.4 Aufbau und Inhalt der Arbeit

Nach dieser Einleitung wird in der Studienarbeit zunächst Harvest im Kapitel 2 und SMES im Kapitel 3 vorgestellt. Diese beiden Softwaresysteme bilden die Grundlage dieser Studienarbeit und werden daher recht umfangreich vorgestellt. Bei Harvest wird vor allem der Gatherer interessant sein. SMES wird zusammen mit der Ontologie der Arbeitsgruppe Karlsruhe das natürlichsprachliche Parsing durchführen. Integrationsmöglichkeiten des Gathering und des Parsing sind in 4 dargestellt und bewertet. Eine Realisierungsmöglichkeit wurde im Verlaufe der Studienarbeit implementiert und in 4 genauer untersucht. Anschließend werden die Probleme der Realisierung bezüglich GETESS diskutiert. Im letzten Kapitel werden weiterführende Ideen dieser Studienarbeit erläutert und eine Zusammenfassung gegeben.

Kapitel 2

Harvest — Suchmaschine für das WWW

Harvest ist eine Internet–Suchmaschine mit Sammelagent, Datenaufbereitung, Speicherungskomponente, und Anfragekomponente (grob zusammengefaßt). Diese Suchmaschine wurde von der Internet Research Task Force Research Group on Resource Discovery (kurz IRTF-RD) entwickelt. Mitglieder dieser Forschungsgruppe kommen von der Universität von Colorado, der Universität von Arizona, der Transarc Corporation und vielen weiteren Institutionen. Die Implementierung ist im wesentlichen durch studentische Projekte entstanden. Der letzte Stand der Arbeiten ist von Anfang 1996. An der Universität von Edinburgh wurde diese Arbeit noch bis zur Version[1] 1.5.20 Ende 1996 weitergeführt. Seit diesem Zeitpunkt ist allerdings keine Fortführung der Arbeiten mehr vorgenommen worden.
Harvest wird als eine der wichtigsten Suchmaschinen in [BYRN99] bezeichnet, wie dieses Zitat aus [BYRN99] belegt:

> *"Currently, there are hundreds of Harvest applications on the Web (for example, the CIA, the NASA, the US National Academy of Science, and the US Government Printing Office), as this software is on the public domain. ..."*

Harvest ist freie Software unter der GNU-Lizenz[2].

In den nächsten Abschnitten wird Harvest mit dem letzten Stand vorgestellt. Es werden die Funktionsweise und der prinzipielle Aufbau der Harvest–Software dargestellt und die für diese Arbeit wesentlichen Teile hervorgehoben.

2.1 Allgemeine Funktionsweise von Harvest

Der Einsatz von Harvest erstreckt sich von kleinen Anwendungen im Intranet bis zum ausgereiften Einsatz als allgemeine Internet–Suchmaschine. Die Anforderungen an Hard– und Software sind verhältnismäßig gering. Im folgenden werden Grundlagen zum Verständnis und die Funktionsweise der Software dargelegt.

2.1.1 Grundlagen

Um den Einsatz der Software zu verstehen, werden Grundlagen aus den Bereichen WWW–Techniken, Agenten und allgemeine Softwaretechnik benötigt.

WWW-Techniken

Als Grundlage für Webanwendungen werden zuerst die Zugriffsprotokolle betrachtet. Harvest benötigt einmal das Hypertext–Transfer–Protokoll[3] (kurz **HTTP**) zum Zugriff auf Texte und Multimedia–

[1] Unter http://www.tardis.ed.ac.uk/~harvest/ zu finden.

[2] GNU kommt von General Public License und bezeichnet freie Software mit frei verfügbarem Quellcode (Open Source).

[3] Ursprünglich unterstützt Harvest auch FTP, Gopher und NNTP, im vorliegenden Szenario wird aber ausschließlich HTTP verwendet

Dokumente (Kompositionen aus mehreren verschiedenen Ausdrucksformen, wie Texte, Bilder, Audio) im Internet. Und zum anderen bietet Harvest eine verteilte Anwendung der Software mit Zugriffsmöglichkeit der Komponenten im Internet untereinander (in 2.1.4 genauer beschrieben). Die Verbindung wird über TCP–Ports mit einem internen Protokoll aufgebaut.
Nachzulesen sind diese allgemeinen Zugriffstechniken für das Internet z.B. in [San95].

Ein wichtiges Konzept im Zugriff auf entfernte Internetdienste ist die zu verwendende Adressierung. Eine Serveradresse im Internet wird allgemein als Uniform Resource Locator (**URL**) bezeichnet. Die URL hat folgende Form:

`Protokoll://Host:Port/Pfad/Ressource`.

Dies ist nur der zweite Teil der eigentlich zur Identifizierung von Internet–Ressourcen nötigen Uniform Resource Identifier (URI). Der erste Teil (Uniform Resource Name) wird momentan von niemandem benutzt, so daß im folgenden ausschließlich von URL's die Rede sein wird.

Zum Verständnis der einzusammelnden Dokumente und deren Informationsextraktion werden Grundlagen über die Hypertext Markup–Language (**HTML**) benötigt. Wie oben schon erwähnt, gehen wir im allgemeinen von HTML als Präsentation der Informationen im Internet aus. Die verschiedenen HTML–Versionen und ihre Spezifikationen können unter [Tol97] und auf vielen WWW–Seiten genauer betrachtet werden. Weitere, vor allem neuere Darstellungsformen und Erweiterungen von HTML–Dokumenten sind z.B. Stylesheets, VRML (Virtual Reality Modeling Language) oder XML (eXtensible Markup Language). Diese können mit Harvest derzeit nicht verarbeitet werden und werden deshalb in diesem Kapitel nicht weiter betrachtet.

Als letzte zu betrachtende WWW–Grundlage wird das Common Gateway Interface (**CGI**) hier kurz beschrieben. Harvest verwendet CGI, um Anfragen über HTTP–Requests beantworten zu können. CGI ist — wie der Name schon sagt — eine Schnittstellenbeschreibung. Es wird hier über spezielle Variablen festgelegt, wie ein solcher Request vom Client auszusehen hat und wie Daten vom Client zum Server übertragen werden. Unter Anwendung von CGI ist es möglich, abhängig vom Client auf Server–Seite z.B. dynamisch HTML–Dokumente zu erzeugen[4]. Eine genaue Beschreibung der CGI–Schnittstelle ist unter [Gun96] zu finden.

Agententechnologie
Unter einem Agenten im Informatikbereich ist terminologisch eine Software unter bestimmten Eigenschaften gemeint. Derzeit gibt es keine einheitliche und eindeutige Definition für einen Softwareagenten. Allgemein kann man einen Agenten nach [BZW98] so erklären: Softwareagenten operieren in einer nicht vorhersehbaren Umgebung, die sie über Sensoren wahrnehmen und durch Aktionen beeinflussen können.
Es werden hier nur kurz einige Eigenschaften, die allerdings für eine Agentensoftware erforderlich sind, genannt und kurz erklärt:
Agenten sind

- **reaktiv** — können auf Ereignisse in ihrer Umgebung reagieren,
- **deliberativ** — um Schlußfolgerungen aus Informationen ziehen zu können und
- **proaktiv** — damit sie selbständig ihre Ziele verfolgen.

Diese drei Eigenschaften sind im allgemeinen die wichtigsten unter einer größeren Anzahl von Charakteristiken für Agenten. Es gibt verschiedene Typen von Agenten, die weitere bestimmte Charakteristiken prägen. Es werden im wesentlichen drei Agentenhaupttypen unterschieden:

- **Persönliche Agenten** greifen ihrem Auftraggeber bei ihren aufwendigen Arbeiten unter die Arme.
- **Mobile Agenten** bewegen sich von Rechner zu Rechner, sammeln Daten und können Ressourcen der einzelnen Rechner ausnutzen.

[4] Es können mittels CGI beliebige Programme auf dem Server (unter Obhut der jeweiligen Serversoftware) gestartet werden.

- **Intelligente Agenten** verfügen über eine Wissensbasis und können planen, lernen und ihre Arbeitsweise optimieren.

Softwareagenten können meist nicht nur in eine dieser drei Kategorien eingeordnet werden. Agenten sind meist Kombinationen aus den genannten Kategorien.
Agenten werden, vor allem im Internet, in nächster Zeit an Bedeutung gewinnen. In [Cro97] und [MWJ96] sind weiterführende Arbeiten zu neueren Agententechniken (beispielsweise kooperative Agenten, Agentensysteme) bzgl. des Suchmaschineneinsatzes zu finden.

2.1.2 Installation und Konfiguration von Harvest

Zur Installation von Harvest sind notwendig:

- ein Webserver, z.B. Roxen oder Apache,
- die Harvest–Software[5],
- je ein freier TCP–Port für eine Harvestkomponente.

Nachdem man die Harvestsoftware von einem Webserver aus dem Internet runtergeladen hat, entpackt und kompiliert man die Software. Vor der Kompilierung ist es wichtig, den genauen Installationspfad festzulegen.

Bevor man nun Harvest startet, sollte ein Webserver installiert, konfiguriert und gestartet werden. Die Installation erfolgt je nach Webserver–Software. Bei der Konfiguration wird ein WWW–Pfad zur Harvest–Installation gesetzt und die WWW–CGI–Verzeichnisse durch den Harvest–CGI–Pfad erweitert.

Mittels "RunHarvest" installiert und konfiguriert man einen Gatherer und einen Broker. Neben allgemeinen Einstellungen der Harvest–Installation werden Angaben zum Gatherer und zum Broker verlangt. Zu den wichtigsten Einstellungen des Gatherer zählt die Angabe der zu durchsuchenden URL's. Hat man eine oder mehrere URL's angegeben, wird der Gatherer automatisch gestartet, und man erhält die erste Datenbasis. Am Ende der Konfiguration wird ein Gatherer–Dämon (Dämon nennt man eine vom System gestartete Software, die auf dem Rechner jederzeit einsatzbereit ist) und ein Broker–Dämon gestartet.
Es ist sinnvoll, von Zeit zu Zeit die Daten zu aktualisieren. Dazu wird der Gatherer einfach neu gestartet. Dies ist z.B. mit einem sogenannten cron–Job (Möglichkeit unter Unix regelmäßig zu definierten Zeiten Software zu starten) zu erreichen, der den Gatherer je nach crontab–Konfiguration regelmäßig aufruft. Der Gatherer wird dann die veralteten Daten erneuern oder löschen und neu hinzugekommene Seiten in die Sammlung aufnehmen.

Genauere und weitaus mehr Angaben zur Konfiguration sind in den Gatherer– und Broker–Konfigurationsdateien vorzunehmen. Sie liegen in den jeweiligen Verzeichnissen der Komponenten, also

`<HarvestHome>/gatherers/<GathererName>/<GathererName>.cf` und
`<HarvestHome>/brokers/<BrokerName>/admin/broker.conf`.

`<HarvestHome>` bezeichnet das Verzeichnis des installierten Harvest und `<GathererName>` und `<BrokerName>` sind jeweils die bei der Installation vergebenen Namen der Komponenten.
Konfigurierbar bei Gatherern sind:

- `Gatherer-Name` — Name des Gatherers,
- `Top-Directory` — Verzeichnis des Gatherers,
- `RootNotes` — Angabe der zu durchsuchenden Internet–Domänen;
  die RootNotes können weiter spezifiziert werden:
  - `URL=URL-Max[,URL-Filter]` — maximale Anzahl einzusammelnder Dokumente und Angabe eines möglichen Filter für die Dokumente,

[5] Die letzte Harvest–Version ist z.B. unter http://www.tardis.ed.ac.uk/~harvest/ zu finden.

  - `Host=Host-Max[,Host-Filter]` — maximal anzulaufende Hosts und die mögliche Angabe eines Filters für Hosts,
  - `Access=Typelist` — Access–Methode, z.B. HTTP, FTP, FILE,
  - `Delay=Seconds` — Anzahl Sekunden zwischen zwei Serverzugriffen,
  - `Depth=Number` — maximale Tiefe beim Verfolgen von Links,
  - `Enumeration=Enum-Programm` — mögliches externes Programm zur dynamischen RootNote–Parameter–Spezifizierung,

- `LeafNotes` – Angabe einzelner Dokumente.

Die Konfigurationsmöglichkeiten des Brokers sind in dieser Arbeit nur von nebensächlicher Bedeutung. Weitere Hinweise zur Installation und Konfiguration erhält man in [Gar96] und in [HSW96].
Angaben zur Installation und Konfiguration des GETESS–Harvest sind im Kapitel 4 zu finden.

2.1.3 Allgemeine Funktionsweise

In diesem Abschnitt wird nun kurz gezeigt, wie die Softwarekomponenten zusammenarbeiten. Dazu werden in der Abbildung 2.1 die Komponenten von Harvest und deren Beziehungen dargelegt. Die Komponenten sind zumeist aus einem Mix von C–Programmen und Programmskripten (Shell oder Perl) implementiert.

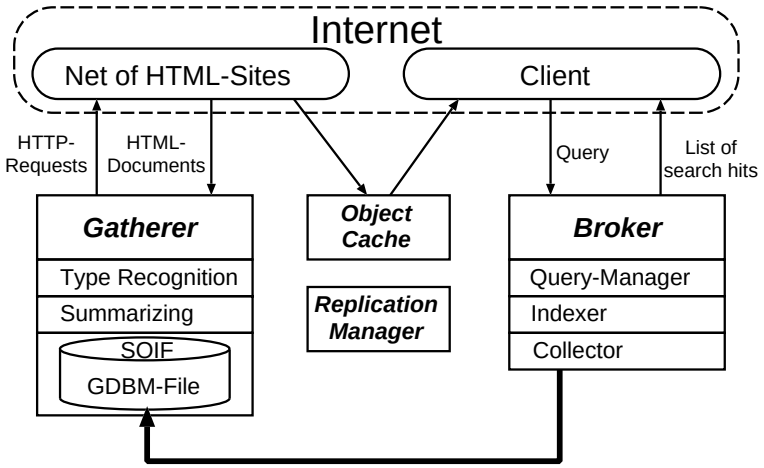

Abbildung 2.1: Darstellung der Komponenten mit Beziehungen und Funktionen von Harvest

Im wesentlichen gibt es bei Harvest vier Komponenten (in Abbildung 2.1 die durch "Italic"–Schreibweise hervorgehobenen).
Der **Gatherer** ist dafür verantwortlich, als Sammelagent die HTML–Seiten einzusammeln. Dazu werden per HTTP–Request nach und nach alle gegebenen URL's und die durch Links gewonnenen URL's angesprochen. Der jeweilige Server des Informationsanbieters gibt daraufhin die HTML–Seite oder eine Fehlermeldung zurück. Zu jedem gesammelten Dokument wird der Typ ermittelt (Type Recognition); wie bereits gesagt, ist der Typ, der hier betrachtet wird, HTML; alle weiteren Dokumenttypen werden nicht behandelt. Danach wird die Seite mit einem Summarizer–Programm (parst ein Dokument und liefert eine SOIF–Struktur zurück) bearbeitet und als SOIF–Datensatz in eine lokale GDBM–Datei geschrieben.

Der **Broker** kann nun mit seinem Collector auf die SOIF–Daten des Gatherers zugreifen. Der Broker indexiert die SOIF–Daten und stellt mit dem Query–Manager eine Zugriffsmöglichkeit bereit. Die Indexierer–Software (siehe Abschnitt 1.3) wird extern eingebunden und kann einfach ausgetauscht

werden (vorinstalliert und voreingestellt ist die GLIMPSE–Software).
Der Client, z.B. ein Nutzer mit Webbrowser, kann über eine Anfrageseite seine Suchanfrage an das System richten. Der Broker sucht in seinen Datenbeständen die gesuchten Informationen heraus und präsentiert sie in einer Trefferliste wieder als HTML–Seite.

Zwei weitere Komponenten sind der **Object Cache**, der als WWW–Cache eingesetzt wird und der **Replication Manager**, zur Replikation des Brokers. Diese beiden Komponenten werden allerdings im folgenden nicht weiter betrachtet, da sie in dem Szenario, das der Studienarbeit zugrunde liegt, keine Bedeutung haben werden. In weiterführenden Betrachtungen dürfen die Konzepte der beiden Komponenten allerdings nicht außer Acht gelassen werden.

Weitere Angaben zum Gatherer und Broker findet man im nächsten Abschnitt, wo die Architektur genauer inspiziert wird.

2.1.4 Verteilungskonzept bei Harvest

Ein wichtiges Konzept in bezug zur Suchmaschine GETESS ist die Möglichkeit der Verteilung der Aufgaben in Harvest. Vorteil für GETESS ist, das ein dezentrales Sammeln und Anfragen der Informationen möglich wird, so daß Rechnerkapazitäten verteilt genutzt werden können. Allerdings ist damit auch eine höhere Netzbelastung verbunden, die aber durch geschickte Verteilung auf die Komponenten wieder ausgeglichen wird. Man kann durch gute Verteilungskonzepte viel größere Datenmengen verwalten, ohne die Skalierbarkeit zu verlieren. Realisierungskonzepte einer verteilten Suchmaschine wurden in [Tit98] untersucht.

Mit Verteilung ist bei Harvest die verteilte Anwendung mehrerer Gatherer und Broker auf mehreren Rechnern gemeint. Die Verteilung ist dezentral, und die einzige Abhängigkeit, die zwischen einem Brokerknoten und einem Gathererknoten besteht, ist, daß ein Gatherer mindestens eine Brokerverbindung benötigt, um seine Daten anbieten zu können, und ein Broker mindestens einen Gatherer erreichen muß, um Daten indexieren zu können.

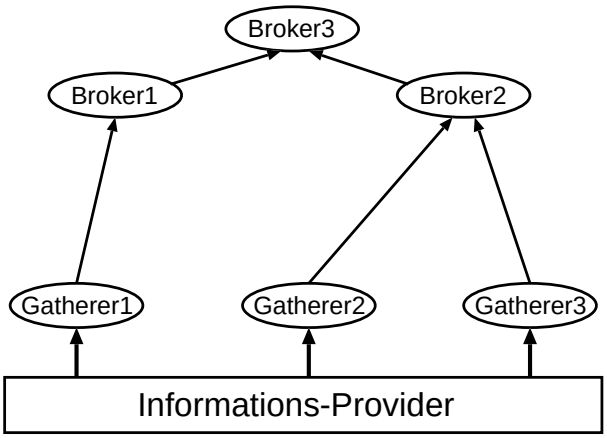

Abbildung 2.2: Verteilte Anwendung mit drei Gatherer und drei Broker

In der Abbildung 2.2 ist eine mögliche Verteilung dargestellt.
Die Basis bilden die Informationsanbieter. Lokale (d.h. lokal beim Informationsanbieter) oder entfernte Gatherer, die auf verschiedenen Rechnern stationiert sein können, sammeln die Dokumente der Informations–Provider ein und stellen eine Schnittstelle zu den Informationen bereit. Auf diese Schnittstelle greifen Broker (auch diese können auf lokalen oder entfernten Rechner installiert sein) zu, um die Daten zu indexieren und eine Nutzerschnittstelle für die Suche in den Daten anzubieten. Ein Broker kann auch auf einen anderen Broker zugreifen (wie in Abbildung 2.2 zu sehen). Dabei

benutzt der zugreifende Broker die Nutzerschnittstelle des anderen Broker.
SWING verwendet eine solche Verteilung, um die Informationen der Internet–Provider in Mecklenburg/Vorpommern[6] zu indexieren.

2.2 Interner Aufbau von Harvest

In diesem Abschnitt werden Interna, Arbeitsweise und das Zusammenspiel der Komponenten von Harvest vorgestellt. Es wird vor allem der Aufbau des Gatherers betrachtet, da die Integration von Harvest und SMES im GETESS–Szenario im wesentlichen durch die Integration des Gatherers mit SMES vorgenommen wird.

2.2.1 Datenformate und Speicherung

Um den Aufbau von Harvest genau beschreiben zu können, sind folgende, grundlegende Daten von Harvest interessant.

GDBM
GDBM ist der GNU Database Manager (eine C–Bibliothek). Dieser verwaltet Daten in Dateien, und zwar in sogenannten GDBM–Files. Man kann eine GDBM–Datei öffnen, etwas hineinspeichern, etwas löschen und die Datei wieder schließen. Der Zugriff erfolgt über Schlüssel (Key). Das heißt, ein Datensatz im Datenbank–File besteht aus einem Key und einem Eintrag pro Key (Wert). Um in der Datenbank zu navigieren, benutzt man Funktionen auf den Keys (z.B. “firstkey” oder “nextkey”).
Harvest verwaltet seine Daten in GDBM–Dateien.

SOIF–Datenformat
In [HSW96] wird das SOIF (Summary Object Interchange Format) als Speicherungsdarstellung der Daten bei Harvest beschrieben. Gesammelte Dokumente werden geparst und die Informationen in Attributwertpaaren gespeichert. Aus einem Dokument wird eine Liste dieser Attributwertpaare erzeugt und im SOIF–Format gespeichert. Die zu erzeugenden Attribute sind vom Dokumenttyp abhängig (z.B. werden bei HTML andere Attribute als bei ASCII–Texten generiert). In der Tabelle 2.1 ist die formale Beschreibung zu sehen.

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

Tabelle 2.1: Formale Beschreibung der SOIF–Grammatik (BNF–Darstellung)

Das Objekt, also das geparste Dokument, besteht aus einer Typenbezeichnung, der URL und der Attributliste. Die URL fungiert hier als Identifikator des Objektes. Die Attribute sind durch einen Attributidentifikator (Attributname), einer Angabe zur Größe des Attributwertes und dem Attributwert selbst gekennzeichnet.
Neben den Attributen aus den gesammelten Dokumenten werden noch weitere Attribute, die für den Gathererlauf wichtig sind, erzeugt.

[6] Dieses Projekt heißt Landesinformationssystem MV–Info und baut auf die Techniken von SWING auf (unter http://www.m-v.de zu finden).

Metadaten als SOIF–Attribute
Um eine Verwaltung der gesammelten Daten gewährleisten zu können, ist es notwendig, weitere Informationen über die Dokumente, sogenannte Metainformationen, zu sammeln und zur Verfügung zu stellen. Bei Harvest werden diese Informationen als SOIF–Attribute zwischen den eigentlichen Dokumentdaten gespeichert. Es werden hier einige wichtigen Metadaten eines gesammelten Dokumentes in Harvest vorgestellt.

- **URL** — Die URL ist im allgemeinen nicht im Dokument selbst gespeichert und wird beim Einsammeln gewonnen (URL's sind gegeben oder kommen durch Verfolgen von Linkstrukturen im HTML–Code hinzu). Die URL dient als Schlüssel bzw. Identifikator des Dokumentes (siehe GDBM bzw. SOIF).
- **Type** — Der Typ wird beim Sammeln der Dokumente bestimmt (Type Recognition) und wird in diesem SOIF–Attribut gespeichert.
- **File–Size** — Dies ist die Größe des Dokumentes in Bytes. File–Size wird als SOIF–Attribut gespeichert.
- **MD5** — MD5 ist eine Prüfsumme, die mit UNIX–Systemmittel einfach berechenbar ist und die für jedes Dokument angelegt wird, um ein aufwendiges Vergleichen beim Update der Daten zu vermeiden. MD5 ist als SOIF–Attribut zu finden.
- **Update–Time** — In diesem SOIF–Attribut ist der Zeitpunkt des letzten Updates, also der letzte Abgleich zwischen Original und SOIF–Datensatz der Datensammlung, gespeichert.
- **Refresh–Rate** — Refresh–Rate ist die Anzahl von Sekunden, nach denen frühestens das Dokument in der Datensammlung mit dem Original wieder abgeglichen werden kann. Ist auch als SOIF–Attribut verfügbar.
- **Time–to–Live** — Dieses SOIF–Attribut beinhaltet die Lebenszeit (in Sekunden) eines Datensatzes in der Datensammlung nach dem letzten Update.

Es gibt neben den eben genannten noch weitere Metadaten, die beim Broker wichtig sind. Der Broker benötigt z.B. Angaben wie Gatherer–Name, Gatherer–Host und Gatherer–Port, um bei einer verteilten Anwendung mit mehreren Gatherern zu wissen, woher die Daten seines Datenbestandes kamen.

2.2.2 Der Gatherer

In Abbildung 2.3 wird der interne Aufbau des Gatherers an einer Skizze veranschaulicht und die einzelnen Teilkomponenten des Gatherers erläutert.

Das Szenario
Ein Szenario für die Abbildung 2.3 kann man sich folgendermaßen vorstellen:
Ausgangspunkt ist, daß der Gatherer eine oder mehrere URL's zum Durchsuchen bekommen hat und einen Suchdurchlauf startet. Der Gatherer geht davon aus, daß eine Datei "PRODUCTION.gdbm" existiert. In dieser Datei sind die gesammelten Daten des vorigen Gathering–Laufs im GDBM–Format abgelegt. Existiert eine solche Datei nicht, sind keine Daten vorhanden, und die einzige Aufgabe des Gatherers besteht darin, neue Daten einzusammeln und zu speichern.
Gestartet wird der Gatherer durch den Aufruf des "RunGatherer"–Programmes.
Nach dem Start werden nun die vier Hauptteile des Gatherers (in der Abbildung 2.3 durch die vier "Italic"–dargestellten Objekte beschrieben) nacheinander abgearbeitet.

Die Hauptbestandteile
Als erstes wird der Programmteil "**PrepUrls**" durchlaufen. Hier wird das Einsammeln der Dokumente vorbereitet, indem die gegebenen URL's geprüft werden, ob sie neu oder schon in der GDBM–Datei vorkommen. Bei den in der Datensammlung vorkommenden Daten wird das SOIF–Attribut Refresh–Time geprüft. Wenn diese Zeit seit dem letzten Update abgelaufen ist, wird das Dokument in den

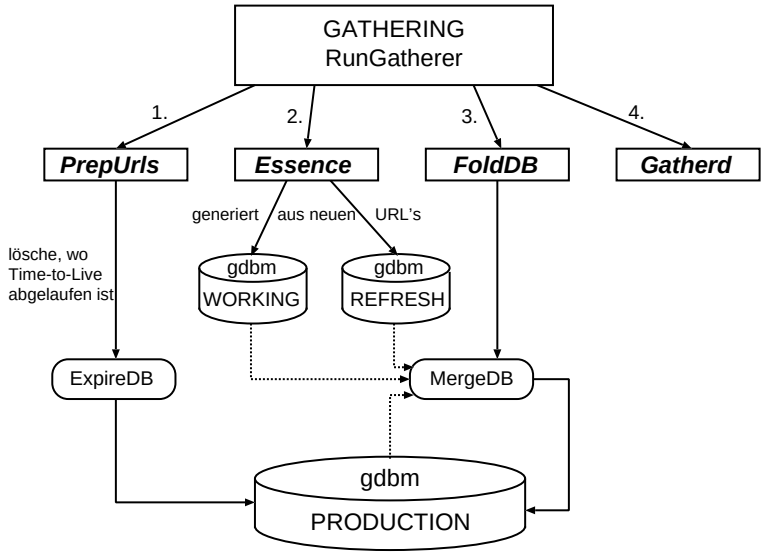

Abbildung 2.3: Skizze der wesentlichen Teilkomponenten im Gatherer

weiteren Programmteilen neu gelesen. Ein weiterer Schritt ist in diesem Programmteil das Entfernen von Informationen aus der "PRODUCTION.gdbm", deren SOIF–Attribut Time–To–Live seit dem letzten Update abgelaufen ist. Dies geschieht durch Aufruf der Funktion "ExpireDB". Davon betroffen sind die Dokumente, deren Update seit längerem nicht mehr vollzogen werden konnte, da vermutlich das Dokument im Internet nicht mehr verfügbar ist.

Der zweite Programmteil ist das **Essence**–System. Im Essence–Programmteil werden die Dokumente eingesammelt, der Typ bestimmt, der jeweilige Summarizer (ein Programm zum Parsen eines Dokumentes eines bestimmten Typs) gestartet und die Daten in der "WORKING.gdbm" bzw. in der "REFRESH.gdbm" zwischengespeichert. Dieses Essence–System wird im nächsten Abschnitt noch genauer beschrieben.

Der nächste Programmabschnitt "**FoldDB**" ist dafür verantwortlich, die Datenbasis in Form der "PRODUCTION.gdbm" neu aufzubauen. Dazu wird aus der alten "PRODUCTION.gdbm" (die bestehenden Daten), aus der "WORKING.gdbm" (die neu gesammelten) und aus der "REFRESH.gdbm" (die upgedateten) eine neue "PRODUCTION.gdbm" durch Verschmelzen (Funktion "MergeDB") gewonnen.

Als letztes wird der "**Gatherd**" (Sammel–Dämon) gestartet. Der Gather–Dämon ist ständig verfügbar und stellt die Daten für die Broker zum Zugriff bereit.

2.2.3 Das Essence–System

Das Essence–System ist für die Informationsextraktion zuständig und wird in [HS95] ausführlich beschrieben. Es ist nötig diesen internen Teil von Harvest zu betrachten, da an dieser Stelle das Parsen der Dokumente und die Erstellung der SOIF–Attribute durchgeführt wird. Man wird sehen, daß diese Informationen für die SMES–Integration von wesentlicher Bedeutung sind. Im folgenden wird gezeigt, wie das Essence–System intern abläuft.

Zunächst ist es sinnvoll, zwischen neu zu sammelnden und aus der Datenbasis zu erneuernde Dokumente zu unterscheiden. Bei den zu erneuernden ist ein erneutes Erstellen der SOIF–Attribute (Summarizing) nur dann erforderlich, wenn sich das Dokument geändert hat. Dazu wird das SOIF–Attribut MD5, wie oben beschrieben, verwendet. Man braucht nur die gespeicherte SOIF–MD5 mit

einer neu generierten MD5–Prüfsumme der aktuellen Version des Dokumentes vergleichen. Wenn sich das Dokument nicht verändert hat, muß nur das SOIF–Attribut "Update-Time" neu gesetzt werden.

Bei neuen Dokumenten und zu erneuernden, veränderten Dokumenten wird nun durch den **Type–Recognition**–Schritt der Typ bestimmt. Je nach Typ wird dann eine eventuelle Vorverarbeitung[7] nötig und der passende Summarizer ausgewählt. Bevor die Dokumente zum Summarizer geschickt werden, werden einige Metadaten (siehe oben) der Dokumente bestimmt und damit Dokumente herausselektiert, die nicht unbedingt durch einen Summarizer geparst werden müssen (z.B. aufgrund ihres Typs).

Für das **Summarizing** existiert für jeden, zu parsenden Typ ein externes Programm[8]. Mittels dieser Summarizer–Programme werden die gesammelten Dokumente geparst. Dabei wird eine Umsetzung von Wörtern und strukturellen Informationen — wenn vorhanden — zu SOIF–Attributwertpaaren vorgenommen. Neben den global definierten SOIF–Attributen bilden die dokumentspezifischen Attribute die andere Klasse von SOIF–Attributen. Die dokumentspezifischen SOIF–Attribute werden aus herausgefilterten Worten und strukturellen Informationen gebildet. Dazu gibt es intern eine Art Stopwortliste, die allerdings nicht konfigurierbar ist. Die hier generierten Datensätze werden zusammen mit den Metadaten später beim Broker indexiert und können angefragt werden.

Nach dem Summarizing gibt es noch ein sogenanntes **Postsummarizing**. In dieser Phase wird der komplette SOIF–Datensatz, also Metadaten und dokumentinterne Daten zusammengestellt. An dieser Stelle ist es möglich extern den Datensatz noch zu beeinflussen. Dazu benötigt man ein sogenanntes Rule File, in dem bedingte Instruktionen auf SOIF–Datensätze definiert werden (diese Technik wird bei SWING zur Einbindung externer Datenbanken verwendet).

Am Ende dieser Prozesse hat Harvest für jedes Dokument einen SOIF–Datensatz erstellt und bietet die Datensätze über den Gather-Dämon nach außen hin an.

[7] Vorverarbeitung — wie z.B. Entpacken gepackter Dokumente oder Zerlegen von sogenannten Multi–Files.
[8] Die Programmiersprache für den Summarizer ist beliebig, aber die In/Out–Schnittstelle ist definiert.

Kapitel 3

SMES/Ontologie — NL–Parsing in GETESS

Am DFKI Saarbrücken sind im Verlaufe mehrerer Jahre einige umfangreiche Tools zur Informationsextraktion aus natürlichsprachlichen Texten konzipiert worden. SMES, wie in [NM98] beschrieben, ist ein Kernmodul zum natürlichsprachlichen Textparsing, das in diesem Zusammenhang realisiert wurde. SMES dient im GETESS–System als Grundlage zur flachen Textverarbeitung.
Um die geparsten Texte domänenspezifisch aufbereiten zu können, wird in Karlsruhe eine Ontologie (siehe [BPW+98] und [MPP+99]) erstellt. Die Ergebnisse des natürlichsprachlichen Parsens mit Domänwissen werden den Gathering–Prozess wesentlich beeinflussen.

Zunächst werden einfache linguistische Grundlagen zum Verständnis der SMES–Software dargestellt. Anschließend werden Aufbau und Funktionsweise des NL–Parsings und die Integration von SMES und der Ontologie betrachtet. Zum Ende dieses Kapitels wird das Parsing am Beispiel getestet.

3.1 Grundlagen zur Dokumentanalyse in GETESS

Um die folgenden Betrachtungen zu motivieren und verständlich zu machen, werden einige Grundlagen zur Sprachanalyse in der Computerlinguistik kurz aufgezeigt.

3.1.1 Terminologie und Konzepte des natürlichsprachlichen Parsen

Es werden hier wichtige Begriffe der Computerlinguistik und wichtige Konzepte des natürlichsprachlichen Parsens genannt und erklärt.

Begriffe
Als erstes wird nochmal der Begriff des abstracts — etwas genauer — definiert:
Ein **abstract** ist eine Menge von instanziierten Templates bzw. domänenspezifischer Konzeptbeschreibungen. Das heißt für GETESS, aus dem gegebenen Dokument wird eine Zusammenstellung von domänenspezifischen Informationen des jeweiligen Inhalts des Dokumentes generiert. Template beschreibt in diesem Zusammenhang also ein Konzept, welches, wenn es im Dokument vorkommt, mit den Werten aus dem Inhalt des Dokumentes instanziiert, in die Menge der abstract–Beschreibungen aufgenommen wird.
Des weiteren werden diese abstracts später in ihrer internen Form betrachtet. Nachfolgend werden einige Begriffe der Computerlinguistik entsprechend [Bus90] kurz erläutert, die für das Verständnis der weiteren Arbeit notwendig sind.

- Morphologie — Oberbegriff für Flexion und Wortbildung in der Sprachwissenschaft.
- Grammatik — systematische Beschreibung der morphologischen und syntaktischen Regularitäten einer natürlichen Sprache.
- Lexikon — beschreibt den Wortschatz einer Sprache soweit kodifiziert, als seine Formen und Bedeutungen nicht aus den Regularitäten des Sprachsystems ableitbar sind.

- Syntaxbaum — formale Beschreibung der syntaktischen Analyse einer natürlichsprachlichen Äußerung.
- Phrase — aus dem Englischen übernommene Bezeichnung für syntaktisch zusammengehörige Wortgruppen.
- Nominalphrase (NP) — Phrase, die ein Nomen oder Pronomen als Kern enthält und in unterschiedlicher Weise erweitert sein kann.
- Präpositionalphrase (PP) — Phrase mit unterschiedlicher kategorialer Ausprägung, die vor allem die syntaktischen Funktionen des Adverbials, des Attributs oder des Objekts erfüllt.
- Verbalphrase (Verbkomplex) — Phrase, die als unmittelbare Konstituente des Satzes fungiert und obligatorisch ein Verb enthält.

Die Zusammenhänge dieser Begriffe werden bei den weiteren Betrachtungen deutlich.

Konzepte
Bei natürlichsprachlichen Systemen unterscheidet [GW93] die theoretische und die Ingenieur–Perspektive. SMES ist ein System, das aus der theoretischen Sicht entwickelt wurde. Das heißt, das System beinhaltet vordergründig Fragestellungen zur Linguistik, weniger die pragmatische Sicht auf konkrete Anwendungen. Die Anwendersicht, wird bei jedem Einsatz von SMES in einem Anwendungsszenario extern bestimmt und realisiert.

Eine allgemeine Architektur der natürlichsprachlichen Textverarbeitung beinhaltet im allgemeinen eine syntaktische und eine semantische Analyse.

In der **syntaktischen Analyse** werden allgemein Parser, Grammatiken und Lexika benötigt. Der Input in Form eines natürlichsprachigen Textes wird mittels Parser unter Nutzung der Grammatiken und der Lexika in Syntaxbäume aufgelöst. Bei der lexikalischen Analyse, als ersten Teil der Syntaxerkennung, werden mittels Lexika und unter Berücksichtigung der morphologischen Regeln die lexikalischen Einheiten separiert und Eigenschaften bestimmt. Die lexikalischen Einheiten mit Eigenschaften werden nun durch die Regeln der Grammatik in ein oder mehrere Syntaxbäume transferiert. Der Syntaxbaum präsentiert eine logische Abstraktion aus den einzelnen Lexemen zu logisch zusammengehörigen Einheiten (z.B. “Ich bin gegangen.” → ich, sei, geh → Subjekt, Verbalphrase → Satz).

Die **semantische Analyse** reichert die syntaktisch gewonnen Daten mit Bedeutungsinformationen der Daten an. Die semantische Analyse ist meist eng mit der syntaktischen Analyse verbunden. Prinzipiell besteht die Möglichkeit Syntax und Semantik getrennt, im direkten Zusammenhang oder als Einheit zu analysieren.
Beim natürlichsprachlichen Parsen wird von den Bedeutungen der Lexeme einer Phrase auf die Bedeutung der Phrase geschlossen und von den Phrasen auf die jeweiligen Sätze. Dies nennt man die Ermittlung der Satzsemantik. Dieses Prinzip läßt sich rekursiv auf Textabschnitte, Dokumente, Dokumentgruppen erweitern. Dabei wächst allerdings der Aufwand schnell und bei jedem Rekursionsschritt wird das Ergebnis im allgemeinen ungenauer. Außerdem können wichtige Bedeutungsinhalte zwischen Einheiten verschiedener Ebenen bestehen, so daß eine Analyse über den vollen Umfang aussichtslos erscheint. Einschränkungen macht man beim zu suchenden Wissen und bei den Rekursionsebenen.

3.1.2 Ontologie

Es ist sinnvoll, durch die Eingrenzung des Bedeutungsraumes einen geringeren Aufwand und höhere Effektivität der natürlichsprachliche Analyse zu gewährleisten. Eine Möglichkeit einen eingegrenzten Bedeutungsraum zu beschreiben, ist die Erstellung einer Ontologie.

In GETESS wird eine Ontologie als Modell eines Ausschnitts der allgemeinen Begriffswelt betrachtet. Die zu einer ausgewählten Domäne gehörenden Dinge werden mit ihren Begriffen und den Beziehungen untereinander dargestellt. Weiterhin ist es möglich Regeln bzw. Methoden auf den Begriffen zu definieren.

Die Ontologie dient zur Darstellung des Auschnitts der allgemeinen Begriffswelt. Dies schließt ein Navigieren durch die Ontologie ein. Beim Zugriff auf die Ontologie bei Anfragen, wie z.B. einen Slot zwischen Hotel und Zimmer finden, wird durch eine Inferenzmaschine die benötigte Information aus den Ontologiedaten generiert. Weitere diesbezügliche Zusammenhänge sind in [FDES97] zu finden. Eine Visualisierung kann z.B. mit dem Hyperbolic Ontology View realisiert werden, damit dem Benutzer Ontologiekonzepte verdeutlicht werden können (siehe auch die Visualisierung der Ontologie in [BPW+98]).

Eine solche Ontologie ist geeignet als semantisches Referenzmodell im GETESS–System die natürlichsprachliche Analyse zu unterstützen.

3.2 Aufbau und Funktionsweise des NL–Parsing in GETESS

Zunächst wird das NL–Parsing in GETESS genau eingeordnet, dann wird an einer Architekturskizze die momentane Umsetzung des Parsens gezeigt und abschließend die erfolgten Arbeiten beschrieben.

3.2.1 Einordnung und Anpassung in GETESS

Der Einsatz von SMES in GETESS betrifft das Parsen der Nutzeranfragen im Dialogmodul und das Parsen der gesammelten Texte beim Gathering–Prozeß. Die semantische Analyse wird mit Hilfe der Integration von SMES und der Ontologie durchgeführt. Die wesentliche Schnittstelle beim Parsen besteht zur Ontologie, um — wie erwähnt — die Bedeutungen des natürlichsprachlichen Textes auf die definierten Konzepte der Domäne abzubilden.
Das natürlichsprachliche Parsen mit SMES/Ontologie wird vom Gathering–Prozeß bzw. vom Dialogsystem initiiert.

GETESS–relevante Eigenschaften von SMES
In [BPW+98] werden fünf Eigenschaften (die teilweise einander bedingen) genannt, die für GETESS relevant sind.

- Robuste und effiziente Sprachverarbeitung von deutschen Texten:
  Die *robuste und effiziente Verarbeitung* ist ein Ergebnis der sehr flachen Sprachverarbeitung und Informationsextraktion. Es wird innerhalb von GETESS in diesem Bezug eine große Herausforderung an die Geschwindigkeit bestehen. Die Problematik besteht dabei nicht nur bei der Extraktion von Informationen aus dem Internet, sondern auch beim natürlichsprachlichen Dialog mit dem Nutzer.

- Umfangreiche linguistische Wissensquellen:
  Die bei SMES zugrundeliegenden *Wissensquellen* (Lexika und Grammatiken) sind sehr *umfangreich* und können auf relativ einfache Weise erweitert werden.

- Erweiterbarkeit der linguistischen Wissensquellen durch deklarative Formalismen:
  Dies ist vor allem für die Integration mit der Ontologie von Bedeutung. Die *Erweiterbarkeit* ist deshalb wichtig, da einerseits bei einer domänenspezifischen Semantikanalyse mit der Ontologie und andererseits bei der *Adaption auf weitere Sprachen* die Wissensquellen erweiterbar und änderbar sein müssen.

- Hohes Maß an Modularität:
  Voraussetzung für eine einfache Erweiterung ist, daß das System *hoch modular aufgebaut* ist.

- Adaptierbarkeit auf andere Sprachen:
  GETESS wird Deutsch und Englisch unterstützen.

Anpassung von SMES/Ontologie an die GETESS–Anforderungen
Das Informationsextraktionsmodul SMES gibt ohne GETESS–Aufsatz nach Eingabe eines Textes einen Syntaxbaum in Form einer LISP–Struktur (LISP ist eine Programmiersprache; die Software ist intern mit LISP realisiert) aus. Damit SMES den Wortschatz der im GETESS–Projekt zugrundeliegenden Domäne Tourismus verstehen kann, wurde zum **ersten Meilenstein** (vgl. [BPW+98]) das

System mit einem Korpus aus www.all-in-all.de trainiert. Dabei wurden die Wissensquellen erweitert, angepaßt und evaluiert. Die lexikalische Abdeckung konnte auf 89% der gelesenen Worte gesteigert werden.
In Karlsruhe wurde ausgehend vom Ontobroker–Projekt ([FDES97] und [BPW+98]) ein Prototyp einer Ontologie erstellt und Schnittstellen zu anderen GETESS–Komponenten abgeglichen.

Zum **zweiten Meilenstein** (vgl. [MPP+99]) konnte bei SMES eine lexikalische Abdeckung von 100% erreicht werden. Dabei wurden die Wissensquellen teilweise per Hand erweitert.
Die Ontologie wurde in Karlsruhe um weitere Konzepte erweitert und der Zugriff auf die Ontologie via Inferenzmaschine ermöglicht.
Weiterhin wurde die Ontologie in die semantische Analyse integriert. Dabei ist ein sogenanntes Domänenlexikon entstanden. In dem die domänenspezifischen Lexeme auf ein Konzept und einen möglichen Slot in die Ontologie abgebildet werden. Wörter mit mehreren Konzepten erhalten auch mehrere Einträge. Dieses Domänenlexikon liegt vorläufig als ASCII–Version vor. Mit dieser Abbildung von Lexikoneinträgen auf Konzepte der Ontologie ist die Integration einer domänenspezifischen Semantikanalyse in die flache Textverarbeitung von SMES möglich. Dazu werden die Grammatiken um eine domänenspezifische Schnittstelle erweitert.

Weiterhin ist noch zu nennen, daß die Arbeitsgruppen Karlsruhe und Saarbrücken gemeinsam Tools zur Aquisition und Pflege der domänenspezifischen Module entwerfen.
Außerdem wurde mit der Spezifikation linguistischer Wissensquellen zur Behandlung englischsprachiger Texte begonnen.

3.2.2 Die Architektur von SMES/Ontologie

In der Abbildung 3.1 ist die Architektur des NL–Parsings in GETESS skizziert.

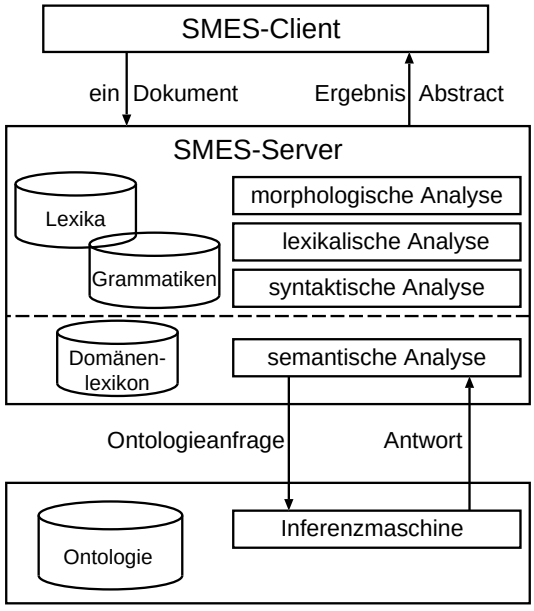

Abbildung 3.1: Architekturskizze der zum NL–Parsing beteiligten Komponenten

Der SMES–Client ist ausschließlich für den Zugriff auf den Server zuständig. Es wird genau ein Dokument vom SMES–Client an den SMES–Server geschickt (das gleichzeitige Parsen mehrerer Dokumente ist momentan nicht angedacht). Im SMES–Server werden unter Zuhilfenahme der Lexika und Grammatiken die morphologische Analyse, dann die lexikalische Analyse und die syntaktische

Analyse durchgeführt. Die Ergebnisse werden dann in der semantischen Analyse mittels der Ontologie domänenspezifisch aufgelöst. Es wird dabei zum einen das Domänenlexikon benutzt und zum anderen werden durch Ontologie–Anfragen an eine Inferenzmaschine semantische Beziehungen aufgeschlüsselt. Der Vorgang der abstract–Generierung wird nun genauer beschrieben.

3.2.3 Die abstract–Generierung

Nach der morphologischen und lexikalischen Analyse bei SMES erfolgt die syntaktische Analyse durch einen Chunk Parser. Ein Chunk Parser ist nach [MPP+99] ein Parser mit mehreren flachen Grammatiken, die jeweils nur bestimmte Phrasen bearbeiten können. Die Grammatiken, als endliche Automaten definiert, werden in einen Erkennungsteil und einen Ausgabeteil getrennt. So kann der Ausgabeteil durch einfache Erweiterungen um die Angabe domänenspezifischer Informationen ergänzt werden.

Ausgangspunkt für die abstract–Generierung ist das Ergebnis der domänenspezifischen Termextraktion. Um ein abstract zu bilden, werden die einzelnen Terme in Beziehung gesetzt. Dazu ist eine Anfrage an die Ontologie nötig. Das heißt, daß neben statischen Domänwissen innerhalb des Domänenlexikons beim Parsen der Lexeme auch dynamisches Domänwissen nötig ist (s. Abb. 3.1).
Welche Terme in Beziehung gesetzt werden, entscheiden linguistische Regeln und Heuristiken. Folgende drei Heuristiken werden momentan genutzt:

- Kombination aller Terme eines Satzes untereinander,
- Kombination eines Terms aus dem Titel mit allen im Text stehenden,
- Kombination von aufeinander folgenden NP– und PP–Termen miteinander.

Bei der Ontologie–Anfrage werden Paare von Termen linguistisch und heuristisch ausgewählt und an die Inferenzmaschine der Ontologie gesendet. Von dort bekommt man dann umgehend die Antwort, ob es einen Slot zwischen den Termen gibt. Die so gewonnenen Zusammenhänge werden zu einem abstract zusammengefaßt. Die Kombination der Terme und ihren Slots untereinander ist keine triviale Aufgabe.

Als Ausgabeformat steht XML zur Verfügung. Dabei wurde die interne LISP–Darstellung in das projektinterne Austauschformat XML überführt. Die Schnittstellenvorschrift wird über DTD's (Document Type Definition für XML) geregelt. Die XML–Ausgabe ist deklarativ und flexibel realisiert.

3.3 Der SMES–Client

SMES ist als Client/Server–System realisiert. Der SMES–Server ist am DFKI Saarbrücken installiert und wird dort im Rahmen des GETESS–Projektes weiterentwickelt. Der Server beinhaltet die gesamte natürlichsprachliche Textverarbeitung mit allen Lexikas und Grammatiken und dem Chunk Parser. Die Ontologie und deren Inferenzmaschine gehören zum Arbeitspaket des Projektpartners an der Uni Karlsruhe und sind dort auch installiert.

Der Server ist unter

```
limit.dfki.uni-sb.de
```

auf einem bestimmten TCP–Port ansprechbar. Mittels dem SMES–Client kann man auf den Server zugreifen. Dazu wird die Software "parac", der SMES–Client, in der momentan vorliegenden Version 0.9 installiert. Man benötigt für die Software Perl (eine Skriptprogrammiersprache) ab der Version 5.004_04 und PerlTk ab der Version 8.00_012. Der Client besteht aus einem Perlskript und mehreren Perlmodulen (unter anderem die Bibliotheken "ParaClient" und "ParaIO"). Es gibt neben dem Kommandozeilenmodus, auch die Möglichkeit per grafischer Oberfläche[1] mit dem Client zu arbeiten. Für die Einbindung des Client in den Gathering–Prozeß wird ausschließlich der Kommandozeilenmodus benötigt.
Bevor man den Client benutzen kann, müssen zwei Umgebungsvariablen gesetzt sein. Zum einen ist das "PARAC_HOME", das Verzeichnis des Programms, und zum anderen "PERL5LIB", wo neben

[1] Aus diesem Grunde benötigt das Programm PerlTk.

anderen Perl–Bibliotheken die Parac–Libraries per Pfad anzugeben sind.

Aufruf des Client
Gestartet wird das Programm "parac" mit einigen Parametern. Aufruf:

```
parac -m CALL [OPTIONS]
```

Dabei bedeutet "-m" der Kommandozeilenmodus, und "CALL" bezeichnet einmal das Eingabeformat, und zum anderen können Ebenen des Textparsing ausgewählt werden (z.B. nur morphologische Analyse). Beispielsweise wird mit der folgenden "CALL"–Angabe die Verarbeitung von Strings mit allen Parsing–Ebenen ausgewählt:

```
'smes::fst-from-string ((:fst . :all-frags))'
```

Die Optionen (`[OPTIONS]`) sind folgendermaßen zu verstehen:

- `-h <host>` — Servername.
- `-i <file>` — Datei mit dem zu parsenden Text; wenn nicht angegeben, wird die Standardeingabe gelesen.
- `-p <port>` — TCP–Port des Servers.
- `-t <type>` — Auswahl des Ergebnistyps, z.B. 'text/plain' oder 'text/xml'.
- `-v <level>` — Angabe des debug level von '0' für keine Informationen bis '10' für maximale Ausgabe.

Client–Internes
Wenn man den SMES–Client erfolgreich gestartet hat, wird, nachdem die Parameter gelesen wurden, der zu parsende Text bei Angabe eines Textfiles eingelesen oder von der Standardeingabe erwartet. Danach wird eine Verbindung zum Server hergestellt. Wenn die Verbindung zustande gekommen ist, wird der zu parsende Text als Anfrage an den Server geschickt. Das Ergebnis wird dann auf der Standardausgabe ausgegeben. Abschließend wird die Verbindung wieder geschlossen. Die Verbindungsfunktionen findet man im "ParaIO"–Modul. Alle allgemeinen Funktionen sind in "ParaClient" untergebracht.

Der modulare und objektorientierte Aufbau der Software ist für mögliche Modifikationen gut geeignet.

3.4 Beispiel einer SMES–Anfrage

In der Abbildung 3.2 ist ein Dokument der "www.all-in-all.de"–Domäne dargestellt. Es ist sozusagen das Original, dargestellt in einem Browser–Fenster, aus dem Informationen bezüglich der Tourismusdomäne im folgenden geparst werden.

Die Seite aus Abbildung 3.2 wurde als HTML–Code in eine lokale Datei abgelegt und mittels folgendem Aufruf über den SMES–Client an den Server geschickt:

```
parac -m 'smes::pairs-from-html ((:fst . :all-frags))' -i 1127.htm >1127.xml
```

Der komplette HTML–Code der Seite ist im Anhang A zu finden. Und im Anhang B ist ein Ausschnitt des Ergebnisses des Servers als XML–Dokument zu sehen.

Es werden nun Ausschnitte der HTML–Seite und die zugehörigen Ergebnisse betrachtet. Die Ergebnisse sind gekürzt und in eine Tabelle zusammengefaßt, da die momentanen Ausgaben von SMES sehr umfangreich sind. Dies kommt hauptsächlich daher, da Konzepte aufgrund verschiedener zutreffender Erkennungsmerkmale mehrfach aufgenommen werden.

Es wurden folgende Ergebnisrelationen zwischen Termen gefunden:

In der Tabelle (Tab. 3.1) werden nur die ersten 20 Relationen betrachtet.
Die Relation bezeichnet den gefundenen Slot. Die zwei Attribute sind die beteiligten, in Beziehung

Nr.	Relation	Attribut 1	Term 1	Attribut 2	Term 2	Heu.
1	liegt_in_Stadt	Hotel	hotel	Stadt	Rostock	p, sh, nph
2	-	Hotel	hotel	Region	Mecklenburg	sh
3	-	Stadt	Rostock	Region	Mecklenburg	sh
4	hat_Adresse	Hotel (2)	hotel	Adresse	fax	sh
5	-	Ferienwohnung	appartement	Telefon	telefon	sh
6	-	Ferienwohnung	appartement	Fernseher	tv	sh
7	-	Ferienwohnung	appartement	Radio	radio	sh
8	-	Ferienwohnung	appartement	Foen	Foen	sh
9	-	Telefon	telefon	Fernseher	tv	sh
10	-	Telefon	telefon	Radio	radio	sh
11	-	Telefon	telefon	Foen	foen	sh
12	-	Fernseher	tv	Radio	radio	sh
13	-	Fernseher	tv	Foen	foen	sh
14	-	Radio	radio	Foen	foen	sh
15	-	Unterkunft	fruehstueck	Einbettzimmer	einzelzimmer	sh
16	-	Unterkunft	fruehstueck	Geld	dm	sh
17	-	Unterkunft	fruehstueck	Zweibettzimmer	doppelzimmer	sh
18	-	Unterkunft	fruehstueck	Geld (2)	dm	sh
19	-	Unterkunft	fruehstueck	Mehrbettzimmer	dreibettzimmer	sh
20	-	Unterkunft	fruehstueck	Geld (3)	dm	sh

Tabelle 3.1: SMES–Ergebnis auf Anfrage der HTML–Seite aus Abbildung 3.2 (Heu = Heuristiken/Erkennungsmerkmale, p = PREP, sh = SENTENCE-HEURISTIC, nph = NP-PP-HEURISTIC)

stehenden Begriffe der Tourismusdomäne. Die zugehörigen Terme sind die gefundenen Lexeme aus dem Originaldokument. In der letzten Spalte stehen die Erkennungsmerkmale, teilweise Heuristiken und teilweise sprachliche Beziehungen sind hier zu finden.

Der SMES–Parser befindet sich innerhalb der GETESS–Anforderungen in einer sehr frühen Phase, so daß die im folgenden genannten Probleme ausschließlich dieser frühen Version zuzuschreiben sind.

Die ersten vier Relationen in der Tabelle 3.1 sind innerhalb der HTML–Seite aus der Überschrift zwischen den beiden Bilder generiert. Es wurden zwei Slots "liegt_in_Stadt" und "hat_Adresse" gefunden. Diese beiden Slots sind korrekt, aber insgesamt die einzigen im HTML–Code gefundenen. Man kann sich allerdings weitaus mehr Slots aus dem HTML–Code aufbauen. Als problematisch ist weiterhin anzusehen, daß die Terme, die eigentlichen Attributwerte, meist nicht die erwarteten Werte zu den Attributen sind. Beispielsweise erwartet man als Wert beim Attribut "Hotel" den Hotelnamen. Beim Attribut "Stadt" allerdings ist "Rostock" und bei "Region" "Mecklenburg" korrekt gefunden worden (die Ortserkennung funktioniert offenbar recht gut).
Bei den Zeilen 5 bis 14 sieht man weitere Besonderheiten. Im Original gehören diese Zeilen zur Beschreibung der Zimmer. Es wurde das Appartement und dessen Beziehung zu Telefon, Fernseher, Radio und Fön erkannt. Hier könnte noch die genaue Beziehung, z.B. "hat", angegeben werden. In den dann folgenden Beziehungen wurden Telefon, Fernseher, Radio und Fön jeder mit jedem in Beziehung gesetzt. Dies ist nicht sinnvoll, da sie ihre Beziehung ausschließlich durch die Beziehung zum Appartement erhalten und eine solche "Überkreuz"–Beziehung sich eher verwirrend auswirken kann. Die letzten Zeilen von 15 an beschreiben die angegebenen Preisklassen. Dabei ist die Zuordnung von "Preise: (inclusive Frühstück)" aus dem Original auf Attribut "Unterkunft" mit Term "Frühstück" nicht ganz korrekt. Hier fehlt die Beziehung zwischen "Unterkunft" und "Frühstück", bzw. die Preise in Abhängigkeit zum Frühstück. Die Normierung von "Einzelzimmer" auf Einbettzimmer (Ontologiekonzept) ist gelungen. Die Numerierung der Instanzen in den einzelnen Preiskategorien "Geld", "Geld (1)", "Geld (2)" ist richtig — jedoch ist z.B. weiter oben das Hotel als "Hotel" und "Hotel (2)" zweimal vertreten, obwohl es nur ein Exemplar davon gibt. Außerdem funktioniert offensichtlich die

Abbildung 3.2: HTML–Seite 1127.htm aus der “www.all-in-all.de”–Domäne

Ziffernerkennung noch nicht, da die “Geld”–Terme momentan nur die Angabe “dm” enthalten.

Bei den Erkennungsmerkmalen ist die Sentence-Heuristic die zumeist vorherrschende. Einige Relationen sind durch mehrere Erkennungsmerkmale erkannt worden und könnten zusammengefaßt werden.
Momentan fehlen auch weitere Zusammenfassungen auf höheren Ebenen, so daß man aus dieser flachen Darstellungsweise von Paaren von Konzepten einer Relation zu tiefer geschachtelten abstracts kommt.
Wie gesagt, ist diese Version vom GETESS–SMES eine sehr frühe und wird noch wesentlich verbessert und verändert werden.

Kapitel 4

Realisierung im Gathering-Prozeß

Harvest mit Hauptaugenmerk Gatherer und das NL–Parsing mit Fokus SMES–Client wurden in den vorherigen Kapiteln vorgestellt. Diese beiden Softwarepakete stellen nun die Voraussetzungen für die Integration eines natürlichsprachlichen Parsers und eines Sammelagenten dar.
In diesem Kapitel werden Möglichkeiten der Realisierung der Integration geschildert. Die Realisierungsmöglichkeiten werden anhand von Kriterien untersucht und bewertet. Eine der Realisierungsformen wird dann aufgrund bestimmter Charakteristiken genauer betrachtet und prototypisch umgesetzt. Die Ergebnisse von Tests der prototypischen Umsetzung werden in diesem Kapitel abschließend untersucht. Im nächsten Kapitel finden sich allgemeine Schlußfolgerungen und ein Ausblick dieser Arbeit bzgl. GETESS.

4.1 Bewertung der Realisierungsmöglichkeiten

Im wesentlichen besteht die Realisierungsaufgabe darin, innerhalb des Harvest–Gatherings den SMES–Client zu starten und die gesammelten HTML–Dokumente zu übergeben. Die Frage ist, an welcher Stelle ist es sinnvoll bzw. technisch möglich den Aufruf des SMES–Clients einzubinden.

4.1.1 Mögliche Realisierungen

In Abbildung 2.3 in 2.2.2 wurde der Gatherer skizziert und beschrieben. Der Gatherer aus der Harvest–Software ist die Komponente, die Dokumente einsammelt, verarbeitet und speichert. Nur in der Gathering–Komponente sind alle zu bearbeitenden Dokumente vollständig vorhanden (in den anderen Komponenten liegen die Dokumente in bearbeiteter Form vor). Da der NL–Parser vollständige HTML–Dokumente benötigt, sind Möglichkeiten der Realisierung im Gathering–Prozeß zu suchen.
Es wurden im Gathering–Prozeß vier Hauptkomponenten dargestellt. Es wird nun gezeigt, welche der vier Komponenten für das Einbinden von SMES in Frage kommen:

"PrepUrls" ist ungeeignet, denn hier werden ausschließlich Vorbereitungen zum Einsammeln getroffen. Für den Aufruf von SMES benötigt man gesammelte Daten. In "PrepUrls" finden sich allerdings einige wichtige Funktionen, die für ein Update der Daten unentbehrlich sind. Dazu gehört vor allem die Funktion "ExpireDB", die alte Daten aus der GDBM–Datei löscht (näheres dazu in 2.2.2).

Das **Essence**–System wird die wichtigste Komponente für die Integration in dieser Arbeit darstellen. Hier werden die Daten eingesammelt und ein SOIF–Datensatz des Dokuments erstellt. Das Dokument liegt für diese Zeit als Datei im "temp"–Verzeichnis des Gatherers. Ein Summarizer greift dann auf die Daten zu. Im nächsten Abschnitt wird das Essence–System für das Einbinden des SMES–Client genauer betrachtet.

In der **"FoldDB"**–Komponente wird eine neue "PRODUCTION.gdbm" aufgebaut. Daraus folgt, daß nur noch SOIF–Datensätze betrachtet werden und keine Originaldateien mehr zur Verfügung stehen. Man könnte einen erweiterten SOIF–Datensatz implementieren, in dem das komplette Originaldokument als Attributwertpaar enthalten ist. Diesen könnte man dann im "FoldDB"–Programmteil

dem SMES–Client übergeben. Dies wäre dieselbe Vorgehensweise wie beim Essence–System, nur eine Stufe später. Zusätzlich käme ein mindestens doppelt so hohes Aufkommen an SOIF–Daten auf Harvest zu. Aus diesen Gründen und da es implementationstechnisch schwierig erscheint SMES in diesem Programmteil zu integrieren, ist es nicht sinnvoll die Realisierung an dieser Stelle vorzunehmen.

Der **"Gatherd"** als Gatherer–Dämon ist die Schnittstelle zu den Brokern, die auf die eingesammelten SOIF–Daten zugreifen können. Hier stellt man sich die Frage, warum nutzt man diese Schnittstelle nicht, um auf die Daten für SMES zuzugreifen. Dazu müßte man, wie beim Programmteil "FoldDB" beschrieben, das SOIF–Format so ändern, daß als Ausgabe zusätzlich das gesamte Dokument als SOIF–Wert übergeben wird. Dann könnte man eine Software realisieren, die mittels Zugriff auf den "Gatherd", wie ein Harvest–Broker, die SOIF–Datensätze ausliest und an den SMES–Client schickt.

Im Grunde sind damit zwei mögliche Realisierungen (Einbindung in das Essence–Systems und Nutzung des Gather–Dämon) herausgefiltert. Eine weitere Möglichkeit ergibt sich durch eine Neuimplementation einer Gatherer–Umgebung. Mehr dazu weiter unten.

4.1.2 Anforderungen an die Realisierungen

Um eine Bewertung der Realisierungsmöglichkeiten vorzunehmen, benötigt man Anforderungen und daraus resultierende Kriterien an die Realisierung. Die Anforderungen und Kriterien ergeben sich aus dem Projektkontext von GETESS. Die nachfolgenden Kriterien sind in allgemeinen Anforderungen, Anforderungen bzgl. des Gathering und Anforderungen bzgl. des Parsings klassifiziert. Die allgemeinen Anforderungen ergeben sich aus technischen und konzeptuellen Charakteristiken der Integration. Die beiden weiteren Klassifizierungen ergeben sich aus den jeweiligen gegebenen Softwarepaketen Harvest und SMES/Ontologie.

- Allgemeine Anforderungen:

  Einfachheit/Aufwand
  Für eine schnell zu realisierende Lösung ist die Einfachheit der Realisierung sehr wichtig. Größere Implementierungsaufgaben sind meist mit viel Zeit und einer großen Fehleranfälligkeit verbunden. Der Aufwand ist eng mit Einfachheit verbunden. Der Aufwand sollte demnach nicht zu groß werden, da Zeit und Arbeit begrenzt zur Verfügung stehen.

  Flexibilität/Modularität/Erweiterbarkeit
  In dieser Arbeit hat man es mit einem frühen Stand der GETESS–Arbeiten zu tun. Es ist höchstwahrscheinlich, daß sich an den beteiligten Komponenten und ihren Schnittstellen noch einiges ändern bzw. erweitern wird. Für schnelles und sicheres Ändern und Erweitern sind Flexibilität und Modularität sehr wichtige Kriterien.

  Möglichkeit der Parallelisierung von Gathering und Parsing
  Parallele Abarbeitung kann einen Geschwindigkeitszuwachs implizieren. Fakt ist, daß erst ein Dokument eingesammelt werden muß, bevor es geparst werden kann.

  Unabhängigkeit zwischen Gathering und Parsing
  Die Unabhängigkeit könnte den Gathering–Prozeß zeitlich und örtlich vom Parsing trennen. Vor allem in verteilten Szenarien ist dies eine sehr nützliche Eigenschaft.

  Datenbankeinsatz
  Dieser Punkt beschreibt die Umstände und Schwierigkeiten beim Einsatz einer Datenbank. Dazu gehören Übergabe der Datensätze, Vorhandensein und Übergabe von Metadaten zu den Datensätzen und die Qualität dieser möglichen Schnittstelle. Außerdem ist interessant, wie eine Speicherung der abstracts ohne Datenbank erfolgen kann und ob man bestehende Speichertechniken verwenden kann.

  Integration
  Hiermit wird der Grad der erreichten Integration von SMES und Harvest bewertet. Die Integration ist abhängig von der Flexibilität, von der Unabhängigkeit, von den Einschränkungen beider Systeme gegeneinander und der Möglichkeit der Parallelität.

- Anforderungen vom Gathering her:

  Einfluß auf den Harvest–Gathering–Prozeß
  Dieser Punkt beschreibt mögliche Einschränkungen gegenüber einem ursprünglichen Gatherer–Lauf. Ein ursprünglicher Gatherer–Lauf hat den Vorteil, daß ältere Harvest–Systeme ihre Funktionalität behalten und als Referenz für eine Bewertung des GETESS–Systems dienen können. Inwiefern eine Veränderung positiv oder negativ zu betrachten ist, wird von der Situation abhängig sein.

  Parallelisierung im Gathering–Prozeß
  Hiermit ist paralleles Einsammeln der Seiten gemeint.

  Umgang mit Metadaten
  Einsatz der Metadaten nach dem Parsing–Prozeß ist für eine darauffolgende Speicherung der Parsing–Ergebnisse sehr wichtig.

- Anforderungen vom Parsing her:

  Vollständigkeit der Dokumente
  Ein aussagekräftiges abstract eines Dokumentes setzt ein vollständiges Original voraus. Abstriche an dieser Stelle sind nicht wünschenswert.

  Parallelisierung im Parsing–Prozeß
  Gleichzeitiges Parsen mehrerer Dokumente kann den Prozeß des Gathern/Parsen beschleunigen. Es ist möglich den SMES–Client mehrmals zu starten. Es wird jedoch nur ein Dokument zu einem Zeitpunkt vom SMES–System bearbeitet. Eine parallele Verarbeitung zur Leistungssteigerung beim NL–Parsing ist erstrebenswert.

In dieser Studienarbeit geht es vor allem um eine einfache, prototypische Umsetzung, bei der die Untersuchung der vorliegenden zwei Programme im Vordergrund stand. Deshalb sind in diesem Zusammenhang die Kriterien für eine sinnvolle Realisierung Einfachheit/Aufwand, Flexibilität/Modularität/Erweiterbarkeit und Integration ausschlaggebend.

4.1.3 Vorstellung und Bewertung der Realisierungsvorschläge

In diesem Abschnitt werden zunächst konzeptionelle Aspekte und dann die Bewertung anhand der beschriebenen Kriterien für jede der oben herausgefilterte Realisierungsmöglichkeit diskutiert. Die abschließende Tabelle vergleicht die betrachtenden Realisierungsmöglichkeiten.

1. Möglichkeit: Realisierung im Essence–System
Abbildung 4.1 zeigt, wie prinzipiell diese Realisierung funktioniert.

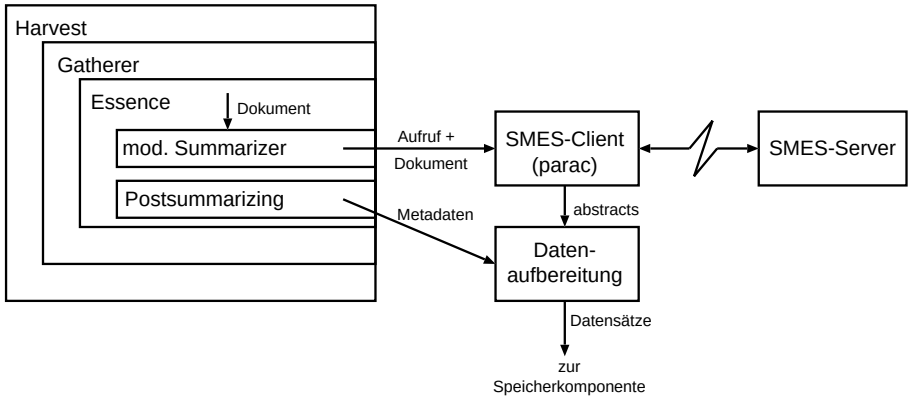

Abbildung 4.1: Skizze der Realisierungsmöglichkeit im Essence–System

Ausgehend von Harvest und dessen Gatherer–Komponente werden im Essence–System des Gatherers Modifikationen am Summarizer und beim Postsummarizing vorgenommen. Im Summarizer werden der SMES–Client gestartet und das zum Summarizing übergebene Dokument an den Client gereicht. Der SMES–Client übernimmt die Kommunikation mit dem SMES–Server und gibt ein geparstes Dokument zurück. Dieses Dokument wird nun zwischengespeichert. Beim Postsummarizing (es liegen nun sämtliche Metadaten der Dokumente vor) werden die Metadaten aller geparsten Dokumente ausgelesen und mit den jeweiligen NL–geparsten Datensätzen integriert.

- **Einfachheit/Aufwand**
  Die Realisierung im Essence–System wird die einfachste Realisierung darstellen. Der Aufwand ist gering, Erfahrungen beim Umgang mit Summarizern und dem Postsummarizing bestehen aus dem SWING–Projekt.

- **Flexibilität/Modularität/Erweiterbarkeit**
  Die Realisierung erfolgt innerhalb eines Summarizers, wobei die Schnittstelle aufgrund der einfachen Realisierung flexibel und erweiterbar bleiben wird. Eine Beschränkung der Erweiterbarkeit liegt im begrenzten Umfeld der Summarizer im Essence–System. Veränderungen über das gesetzte Maß der Summarizer und des SOIF–Formates (siehe Abschnitt 2.2) sind nur mit größerem Aufwand zu vollziehen. Die Modularität ist innerhalb der Beschränkungen durch die Harvest–Software gegeben.

- **Parallelisierung von Gathering und Parsing**
  Der Gathering–Vorgang und der Parsing–Vorgang sind nicht parallelisierbar, da der Summarizer unmittelbar nach dem Sammeln eines Dokumentes aufgerufen wird und das Dokument danach nicht mehr verfügbar ist.

- **Unabhängigkeit**
  Mit der Unabhängigkeit verhält es sich ähnlich der Parallelisierung, da durch den internen Aufruf der Summarizer der Parserlauf örtlich und zeitlich gebunden ist. Man könnte durch zusätzliche Implementierung eine aufgesetzte Unabhängigkeit erreichen, dann wäre man vom Aufwand her allerdings schon bei der zweiten Realisierung ("Gatherd").

- **Datenbankeinsatz**
  Die zu speichernden Datensätze liefert der SMES–Client. Die Metadaten werden nachträglich mit den Datensätzen integriert. Das Postsummarizing liefert die Metadaten, ist aber von Harvest für diese konkrete Aufgabe nicht konzipiert worden. Die Integration von Daten und ihren Metadaten kann ausschließlich extern mit ein dafür zu entwickelndes Tool durchgeführt werden.
  Eine Speicherung der abstracts ohne Datenbankeinsatz, z.B. in die GDBM–Speicherung von Harvest als Teil des SOIF–Datensatzes, ist schwierig. Welche Möglichkeiten es konzeptuell gibt, wird im nächsten Kapitel beschrieben. Prinzipiell ist es möglich durch das Postsummarizing den SOIF–Datensatz zu modifizieren. Man kann aber auch schon im eigentlichen Summarizing–Prozeß einen modifizierten Datensatz erzeugen.

- **Integration**
  Die Integration ist nur bedingt erfüllt, da Abstriche vor allem bei der Unabhängigkeit und der Parallelisierung vorhanden sind.

- **Einfluß auf Gathering–Prozeß**
  Ein gewisser Einfluß dieser Realisierungsform auf den Gatherer–Prozeß besteht darin, daß ein normaler Gathering–Lauf und ein Gathering–Lauf mit NL–Anbindung nicht getrennt werden können.

- **Parallelisierung des Gathering**
  Eine Parallelisierung des Dokumentsammelns ist bedingt durch mehrere verteilte Harvest–Installationen möglich, innerhalb eines Gatherings allerdings nicht.

- **Umgang mit Metadaten**
  Metadaten werden im Verlaufe der Arbeit des Essence–System in Harvest erzeugt. Metadaten können beim Summarizing nicht an die Speicherung übergeben werden. Hierzu benötigt man weitere Schnittstellen zum Gatherer, wie das Postsummarizing.

- **Vollständigkeit der Dokumente**
  Die Vollständigkeit ist gewährleistet, da direkt nach dem Einsammeln der Dokumente mit dem Summarizer auf die eingesammelte Datei im "temp"–Verzeichnis des Gatherers zugegriffen wird.

Diese Realisierung wurde innerhalb der Studienarbeit implementiert und getestet und wird im nächsten Abschnitt genau betrachtet.

2. Möglichkeit: Realisierung beim Gather–Dämon
Diese Realisierungsmöglichkeit stellt eine höhere Herausforderung bei der Implementierung gegenüber der ersten dar. In der Abbildung 4.2 ist die Architektur skizziert.

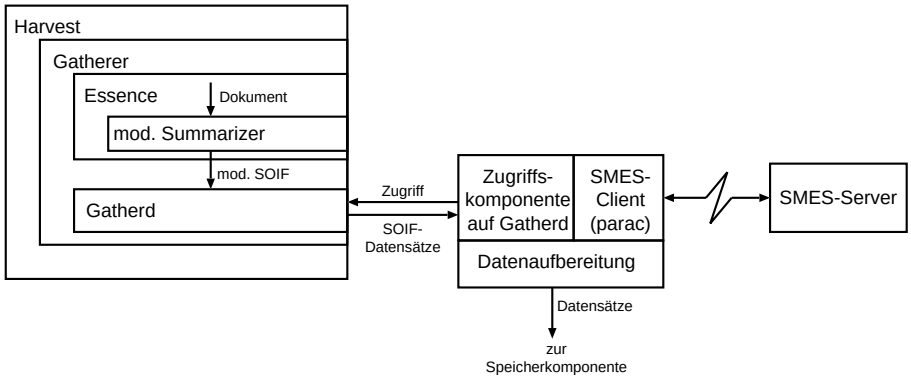

Abbildung 4.2: Skizze der Realisierungsmöglichkeit mittels Gather–Dämon

Bei Harvest wird im Gatherer der HTML–Summarizer des Essence–Systems so modifiziert, daß der resultierende SOIF–Datensatz ein zusätzliches SOIF–Attribut ausgibt. Dieses zusätzliche Attribut enthält das komplette unveränderte Dokument. Der Gatherer behandelt diese erweiterten SOIF–Attribute genauso, als wenn diese Erweiterung nicht vorhanden wäre. Der "Gatherd" des Gatherers stellt die erweiterten SOIF–Datensätze über den Gatherer–TCP–Port zur Verfügung. Eine zu entwickelnde Zugriffskomponente greift auf die Daten des "Gatherd" zu, startet den SMES–Client und übergibt das komplette Dokument aus dem zusätzlichen SOIF–Attribut an den SMES–Client. Die Ergebnisse des SMES–Client werden mit den anderen SOIF–Daten (Metadaten) aufbereitet und der Speicherkomponente zugeführt.

- **Einfachheit/Aufwand**
  Die Implementierung wird sich als schwieriger und der Aufwand sich etwas größer herausstellen als bei der ersten Realisierungsmöglichkeit.

- **Flexibilität/Modularität/Erweiterbarkeit**
  Die Flexibilität der Schnittstellen und die Modularität sind genauso gegeben wie bei der Realisierung im Essence–System. Aufgrund dessen, daß weite Teile der Implementierung außerhalb der Harvest–Umgebung liegen, ist die Erweiterbarkeit größer als bei der Essence–Realisierung. Einschränkungen bestehen nur beim Erweitern des SOIF–Datensatzes.

- **Unabhängigkeit**
  Die große Stärke dieser Realisierung besteht in der Unabhängigkeit der beiden Systeme und die damit verbundene Möglichkeit eine "Pseudo"–Parallelität bei der Verarbeitung zu erreichen.

Unabhängigkeit deshalb, da der "Gatherd" immer verfügbar ist und die zu implementierende Zugriffskomponente jederzeit und von jedem Ort auf die Daten des "Gatherd" zugreifen kann.

- **Parallelisierung**
  Eine gewisse Parallelität läßt sich durch die Unabhängigkeit konstruieren, indem man nach einem erfolgten Gathering–Lauf das Parsing durchführt und während des Parsens der "alten" Daten, der Gatherer schon die nächsten Daten einsammeln kann.

- **Datenbankeinsatz**
  Die Speicherkomponente kann direkt durch den Block Zugriffskomponente/SMES–Client/Datenaufbereitung bedient werden.
  Eine Speicherung innerhalb von Harvest als SOIF–Daten stellt sich aufgrund des externen Aufbereitens als nicht geeignet dar.

- **Einfluß auf Gathering–Prozeß**
  Der Einfluß auf das Gathering besteht darin, daß die SOIF–Datensätze geändert werden, und damit zu rechnen ist, daß sich der Speicheraufwand aufgrund der zusätzlichen kompletten Dokumentdaten für das natürlichsprachliche Parsen verdoppelt.

- **Parallelisierung des Gatherers**
  Die Einschränkungen bei der Parallelisierung des Gathering verhalten sich genau wie bei der ersten Realisierung (Essence–System).

- **Umgang mit Metadaten**
  Weitaus günstiger als in der Essence–Realisierung sieht es beim Umgang mit den Metadaten aus. Hier werden die Metadaten, die Harvest zur Verfügung stellt, beim "Gatherd" vorgehalten. Dann ist es an der Zugriffskomponente, diese Metadaten dem Speichersystem zur Verfügung zu stellen.

- **Vollständigkeit der Dokumente**
  Bei der Vollständigkeit der Dokumente sind die Interna von Harvest entscheidend. Gibt es Einschränkungen bei den SOIF–Daten, wie beispielsweise bei Größe des SOIF–Datensatzes oder bei bestimmten Zeichen, wird eine Vollständigkeit der Daten nur begrenzt gewährleistet.

- **Integration**
  Aufgrund der Unsicherheit bei der Harvest–internen SOIF–Verarbeitung ist die Integration bedingt gelungen.

Diese Realisierungsmöglichkeit wird in dieser Arbeit nicht weiter betrachtet, aber für die Anforderungen an GETESS bezüglich eines verteilten Einsatzes (s. 2.1.4) ist diese Art der Integration (aufgrund von Unabhängigkeit und Parallelität) sehr vielversprechend.

3. Möglichkeit: Neuimplementation

Diese Realisierung ist als allgemeine Konzeption zu sehen. Es können neu implementierte Komponenten oder teilweise bestehende Systemkomponenten integriert werden. Bei einer Neuimplementation gibt es viele konzeptuelle Freiheiten. Im wesentlichen benötigt man einen Sammelagent, der die Fähigkeiten eines WWW–robots hat, eine Zugriffskomponente zum NL–Parsing und eine Schnittstelle für die Speicherung.
Eine Neuimplementation des Gathering–Prozesses kann bei den genannten Kriterien nur teilweise bewertet werden. Einige Kriterien hängen stark von der jeweiligen realen Implementation ab. Es werden deshalb nur die Möglichkeiten einer solchen Neuimplementierung aufgezeigt.

- **Einfachheit/Aufwand**
  Zum ersten ist der Aufwand natürlich groß und von einer einfachen Integration des SMES–Client und einer Gatherer–Umgebung kann nicht ausgegangen werden.

- **Flexibilität/Modularität/Erweiterbarkeit**
  Diese Kriterien hängen stark von der jeweiligen Implementation ab. Die Flexibilität und Erweiterbarkeit sollten bei einer modularen Bauweise in der Regel gegeben sein.

- **Parallelisierung von Gatherer und Parser**
  Die Parallelisierung ist bei einem selbst implementierten System möglich (wenn man dies bei der Programmierung so vorsieht). Beschränkungen, wie durch Harvest, sind nicht vorhanden.
- **Unabhängigkeit, paralleles Gathering, Umgang mit Metadaten, Vollständigkeit**
  Wie bei der Parallelisierung verhält es sich mit diesen Kriterien. Sie sind stark implementationsabhängig.
- **Datenbankeinsatz**
  Bei einer Neuimplementation können jegliche Speichertechniken in das System integriert werden. Jede zusätzliche Komponente bedeutet aber einen erhöhten Konzeptions– und Implementationsaufwand.
- **Integration**
  Für eine sehr gute Integration hat die Neuimplementation aufgrund ihrer wenigen Einschränkungen die besten Voraussetzungen.

Die implementationsabhängigen Kriterien werden in der abschließenden Bewertungstabelle (Tabelle 4.1) mit dem "mittelmäßig"–Zeichen dotiert. Bei einem idealen System sind für die Integrationsaufgabe die eben genannten Felder als gut zu bewerten. Dabei ist es durchaus möglich bei geeigneter Konzipierung und Implementierung ein ideales System zu erhalten. Bei einem solchen Fokus und ohne Einschränkungen, z.B. von Harvest–Seite her, ist eine gute Integration erreichbar. Aufgrund des hohen Aufwands wird momentan von dieser Realisierungsmöglichkeit abgesehen und in dieser Arbeit nicht weiter betrachtet.

Zusammenfassende Bewertungstabelle

Kriterium	im Essence–System	beim "Gatherd"	Neuimplementation
Einfachheit/Aufwand	$\oplus$	$\odot$	$\ominus$
Flexibilität/Modularität/ Erweiterbarkeit	$\odot$	$\odot$	$\oplus$
Parallelisierung Gatherer, Parser	$\otimes$	$\odot$	$\odot$
Unabhängigkeit	$\ominus$	$\oplus$	$\odot$
Datenbankeinsatz	$\odot$	$\odot$	$\oplus$
Integration	$\ominus$	$\odot$	$\oplus$
Einfluß auf Gathering–Prozeß	$\odot$	$\odot$	$\oplus$
Parallelisierung Gatherer	$\odot$	$\odot$	$\odot$
Umgang mit Metadaten	$\ominus$	$\odot$	$\odot$
Vollständigkeit der Dokumente	$\oplus$	$\odot$	$\odot$

Tabelle 4.1: Bewertung der vorgestellten Realisierungsmöglichkeiten ($\oplus$ – gut; $\odot$ – mittelmäßig oder unbestimmt (s. Text); $\ominus$ – schlecht; $\otimes$ – nicht möglich oder nicht vorhanden)

In der vorgestellten Bewertungstabelle (Tabelle 4.1) wurden beim "Gatherd" und der Neuimplementation teilweise nur vage Aussagen gemacht, da einerseits diese Realisierungen in der vorliegenden Arbeit nicht evaluiert worden sind und es andererseits auf die jeweilige Implementierung ankommt.

4.2 Realisierung innerhalb des Essence–Systems

Unter den betrachteten Realisierungsmöglichkeiten ist die Integration von SMES und Harvest beim Summarizing innerhalb des Essence–Systems des Gatherers die einfachste und schnellste Methode. Al-

lerdings bietet diese Möglichkeit weniger Flexibilität und Optimierungsmöglichkeiten (Unabhängigkeit und Parallelisierung) als andere Realisierungen. Diese Realisierung stellt für den GETESS–Prototyp eine vom Aufwand/Nutzen sehr günstige Gathering–Parsing–Integration dar. Deshalb wurde im Laufe diese Studienarbeit genau diese Realisierung genauer analysiert und prototypisch implementiert.

Installation und Konfiguration
Für die Realisierung dieser Integrationsmöglichkeit wurden am Fachbereich Informatik der Universität Rostock ein Webserver (Roxen 1.3), Harvest (Version 1.5.20), dazu gehören ein Gatherer und ein Broker, und der aktuelle SMES–Client (Version 0.9) installiert und konfiguriert. Näheres dazu ist in den Kapiteln 2 und 3 zu finden. Anzumerken ist, daß Harvest auf dem zur Verfügung stehenden System nicht kompiliert werden konnte. Deshalb wurde Harvest auf einem System mit älterer Betriebssystem–Software kompiliert und auf dem neuen System dann installiert. Diese Vorgehensweise muß zwangsläufig nicht immer funktionieren.

Der HTML–Summarizer
In Kapitel 2.2 ist das Summarizing ein Schritt innerhalb des Gatherers, bei dem mittels kleinerer Parsing–Programme (geparst werden einige Metadaten und zur Indexierung sinnvolle Worte), Summarizer genannt, ein SOIF–Datensatz erstellt wird. Zu diesem Zeitpunkt ist im "temp"–Verzeichnis eine vollständige Kopie des Originaldokumentes gespeichert. Hauptaufgabe ist es nun, den SMES–Client zu starten und diese Kopie zu übergeben. Da bei GETESS momentan nur HTML–Dokumente betrachtet werden, wird an dieser Stelle nur der HTML–Summarizer betrachtet.

Der bei Harvest bevorzugte HTML–Summarizer ist ein SGML–Parser (SGML ist eine Metasprache zur Dokumentbeschreibung, HTML ist eine Instanz von SGML), der unter Angabe des Dokumenttyps das Dokument strukturell in SOIF–Bestandteile zerlegt. Aufgerufen wird beim Gathering dann eine Datei, deren Namen sich aus Dokumenttyp und dem Suffix "sum" zusammensetzt. Die Datei "HTML.sum" ist also das zuständige Summarizing–Programm. Diesem Programm wird der Name mit Pfad der im "temp"–Verzeichnis liegenden Dokumentkopie übergeben. Das Programm ist ein Shell–Script (Skriptsprache der jeweiligen verwendeten Shell unter UNIX) und ruft den SGML–Summarizer auf und übergibt den Dokumenttyp (HTML) und den Namen plus Pfad der Dokumentkopie. Die Erstellung des SOIF–Datensatzes übernimmt dann komplett der aufgerufene SGML–Summarizer. Nach dessen Beendigung wird auch der HTML–Summarizer beendet.

Einbau des SMES–Client–Aufrufes
Der HTML–Summarizer wurde nun so präpariert, daß er den SMES–Client aufruft und das Dokument übergibt. Dazu wurde vor dem SGML–Summarizer–Aufruf, folgender Aufruf des SMES–Client eingefügt:

```
parac -m 'smes::fst-from-string ((:fst . :all-frags))' -t text/xml -i $* >
<outfile>
```

Der Aufruf kann mit den Erklärungen aus Kapitel 3.3 nachvollzogen werden. Die Angaben zu Port, Host und debug level sind im SMES–Client–Programm vorgenommen worden. Die Angabe `$*` beim Dateiparameter (`-i`) ist der dem Summarizer übergebene Name plus Pfad der Dokumentkopie. Durch "`> <outfile>`" wird das Parsing–Ergebnis in eine Datei mit dem angegebenen Namen geschrieben. Damit dieser Name für jedes Dokument eindeutig ist, wurde mittels des Unix–Programmes "date" der Name mit dem Erstellungszeitraum kombiniert. Das ermöglicht einerseits Eindeutigkeit und anderseits bemerkenswerte Analysen zum Zeitverbrauch einzelner Dokument–Gathering–Parsing–Vorgänge. Ein Testlauf und deren Ergebnisse sind im nächsten Abschnitt zu finden.

Zugriff auf die Metadaten
Metadaten sind für die Speicherung sehr wichtig. Zum einen will eine Internetsuchmaschine zu einer Anfrage Internetseiten präsentieren. Dazu benötigt man die URL's der Seiten. Zum anderen möchte man in der Speicherkomponente aktuelle Daten haben. Dies erreicht man durch regelmäßiges Update oder Delete der älteren Seiten. Dafür braucht man Metadaten, wie Refresh–Time, Update–Time, Live–Time etc. Diese Metadaten werden von Harvest für die interne Speicherung schon verwendet. Die Aufgabe besteht nun darin, diese Metadaten auch der externen Speicherung in Datenbanken zur

Verfügung zu stellen.

Das Problem, welches bei der Realisierung im Essence–System auftritt, ist, daß bisher keinerlei Metadaten zu den eigentlichen Daten hinzugefügt werden konnten. Beim Summarizing werden nur die Dokumente selbst geliefert. Man kennt eigentlich zu diesem Zeitpunkt nicht einmal zu welcher Internetadresse das Dokument gehörte. Es wird aber intern eine Environment–Variable "SUMMARIZER_URL" gesetzt, die die aktuell zu bearbeitende URL enthält. Damit ist es zumindest zum Zeitpunkt des Summarizings möglich, die Zuordnung vom geparsten Dokument zur URL durchzuführen. Weitere Metadaten können per Postsummarizing dazugewonnen werden. Das Postsummarizing ist für das nachträgliche Verändern der SOIF–Attribute zuständig. An dieser Stelle kann man einen SOIF–Datensatz nach angegebenen Kriterien auswählen und dessen Attribute lesen und schreiben. Für das besagte Problem mit den Metadaten benötigt man nur die Lesefähigkeit. Die gelesenen Informationen können dann an die geparsten Daten angehängt oder anderweitig der Speicherkomponente zugänglich gemacht werden.
Am sinnvollsten ist es, ein XML–Dokument mit Metadaten plus geparsten Daten zu übergeben. Die geparsten Daten werden vom SMES–System schon als XML–Dokument, wie in Anhang B zu sehen, geliefert (die URL des Originaldokument ist nicht enthalten). Am einfachsten wäre es, das XML–Dokument um die Metadaten zu erweitern; beispielsweise so:

```
<?xml version='1.0' encoding='ISO-8859-1' standalone='yes'?>
<url>http://www.all-in-all.de/1127.htm</url>
<update-time>27. April 2000</update-time>
<time-to-live>31 days</time-to-live>
<refresh-time>7 days</refresh-time>
...
<list>
...
SMES–Ergebnis siehe Anhang B.
...
</list>
```

Zur Beschreibung dieser Schnittstelle zwischen dem Gathering und der Speicherkomponente wäre es sinnvoll, eine DTD (Document Type Definition für XML–Dokumente) zu definieren. Mehr zu XML– und DTD–Definitionen ist unter [BM98] zu finden

Möglichkeit der abstract–Speicherung in SOIF
Angenommen, man hat keine Datenbank zur Speicherung der abstracts zur Verfügung. In diesem Falle möchte man die hohe Strukturierung der domänenspezifischen Daten der abstracts nicht vermissen. Neben der simplen Speicherung in Dateien (die abstracts liegen zwischenzeitlich als XML–Dateien im "temp"–Verzeichnis) besteht die Möglichkeit die Harvest–interne GDBM–Speicherung zu nutzen. Technisch gesehen ist dies beim Summarizing oder beim Postsummarizing durch relativ einfache Modifikationen möglich.
Konzeptuell besteht die Fragestellung, wie man das SOIF–Format zur Speicherung nutzen kann. SOIF als flach strukturierte Speichervorschrift in Harvest ist in 2.2 dargestellt. Die einfachste Möglichkeit besteht darin, das abstract als ein SOIF–Attribut eines Dokumentes, dessen SOIF–Wert die kompletten abstract–Daten des Dokumentes beinhalten, zu integrieren. Dies würde bedeuten, daß einige Informationen als XML–Daten innerhalb des SOIF–Formates und einige Informationen (Metadaten und durch Summarizer geparste Daten) als SOIF–Daten vorliegen. Wenn man das SOIF–Format auf tiefe Strukturen, wie in XML, erweitert, besteht in der Regel das Problem, daß der Harvest–Broker diese Strukturen nicht auswerten kann. Das bedeutet, daß es prinzipiell einfach ist, die abstract–Daten in Harvest–internen Speicherstrukturen unterzubringen, aber es viel schwieriger ist, diese neuen Strukturen auszunutzen (z.B. durch den Harvest–Broker).

4.3 Ergebnisse und Schlußfolgerungen

Es werden nun die Ergebnisse von Tests dieser konkret vorgestellten Realisierungsmöglichkeit dargestellt und Schlußfolgerungen gezogen.

Ergebnis eines Gathering–Laufes
Zum Test dieses Grundsystems wurde der Tourismus–Server "www.all-in-all.de" herangezogen. In Absprache mit der Arbeitsgruppe am DFKI Saarbrücken wurde ein kompletter Gatherer–Lauf mit den Seiten des besagten Tourismus–Anbieter durchgeführt. Bei vorangegangenen Tests wurde herausgefunden, das ungefähr 4000 Dokumente durchsucht werden und durchschnittlich 30 Sekunden pro Dokument zu veranschlagen sind. Daraus folgt rechnerisch ein Gatherer–Lauf von rund 33 $^1/_2$ Stunden.
Der tatsächliche Lauf dauerte 49 Stunden und es wurden 4316 Dokumente gesammelt. Im Anhang C sind Ausschnitte des Logs des Gatherers von diesem Prozeß beschrieben. Die größte Zeitbelastung ist durch die Internet–Verbindungen zustande gekommen. Besonders die Mittagsstunden[1] ergaben inakzeptable Zugriffszeiten im deutschen Internet–Bereich.
Momentan verwendet diese vorläufige GETESS–Realisierung drei Verbindungen ins Internet: zum ersten der Gatherer beim Einsammeln des Dokumentes, dann zum zweiten der Aufruf des SMES–Client und zum dritten die Kontaktierung der Ontologie durch den natürlichsprachlichen Parser. Diese drei Verbindungen summieren sich zu ungünstigen Tageszeiten bis zu 5 $^1/_2$ Minuten (durchschnittlich 40 Sekunden) für ein Dokument. Aus diesem Grunde ist im vorigen Abschnitt bei der Bewertung viel Wert auf die Möglichkeit der Parallelisierung gelegt worden, da bei größeren Providern die Gesamtzeiten des Gatherings dementsprechend schlechter werden (www.all-in-all.de kann man als mittelgroß bezeichnen). Hier spielt auch der Verteilungsaspekt hinein, der aber in gewisser Weise von der Parallelität abhängt. Zur möglichen Verteilung von Harvest ist auch ein mehrfaches gleichzeitiges Arbeiten mit dem natürlichsprachlichem Parser vonnöten.

Bei diesem Test wurden noch keine Metadaten in das Parser–Ergebnis eingebunden. Dieser Vorgang — so wie oben beschrieben — benötigt eine Zeit, die gegenüber den zeitintensiven Vorgängen beim Gathering und Parsing vernachlässigbar ist.
Ganz im Gegenteil dazu könnte das in diesem Test nicht durchgeführte Speichern in der Speicherkomponente von GETESS durchaus zeitaufwendig sein. Bei der geplanten Umsetzung der XML–Dokumente in die Datenbankstruktur werden Parser eine gewisse Programmablaufzeit, die abhängig von der XML–Struktur und –Größe ist, benötigen.

Abschließende Bemerkungen
Der Test ist sehr zuversichtlich abgelaufen. Bis auf einen Aussetzer der Software (Ursache konnte nicht bestimmt werden) verlief der Test wie geplant. Die zeitliche Komponente ist zwar sehr wichtig, jedoch wurde der Test unter den schlechtesten Bedingungen durchgeführt. Dazu gehört, daß der Test zu sehr schlechten Verbindungszeiten lief, daß die verwendete Software für den GETESS–Einsatz noch lange nicht ausgereift war und daß keinerlei Optimierungen, wie Parallelisierung, durchgeführt wurden.

4.4 Probleme

Es werden nun Probleme von Harvest, SMES und der Integration im GETESS–System beschrieben.

Probleme bei Harvest
Die letzte Harvest–Version ist mit fast vier Jahren Alter keine neue Software mehr. Das spiegelt sich — wie in 4.2 beschrieben — bei der **Kompilierung auf neueren Systemen** wieder. Dazu kommt, daß neuere, aktuelle Daten und Formate nicht unterstützt werden. Es gibt zwar sehr viele Erweiterungsmöglichkeiten (siehe Summarizer), für alle Eventualitäten werden solche Ansätze auf Dauer sicher nicht anwendbar sein. Dazu gehört auch das intern verwendete SOIF–Format, welches nur in

[1] Die Zugriffszeiten variieren auch zwischen den Wochentagen, wobei Wochenendtage günstigere Zugriffszeiten verzeichnen.

der angegebenen **flachen Struktur** von Harvest genutzt wird. Eine Erweiterung des SOIF–Formates auf stärker strukturierte Dokumente würde zumindest einer Neuimplementation von Indexierer und Broker gleichkommen.
Eine dieser stärker strukturierten Dokumentformate ist **XML**, das vom WWW–Consortium[2] (W3C) zur Ablösung bzw. Erweiterung von HTML als Standard gesetzt wurde. XML und deren zusätzlichen Beschreibungskomponenten (DTD, XSL siehe dazu [BM98]) benötigen neue Summarizer und neue SOIF–Strukturen.
Aber auch beim momentan verwendeten HTML gibt es viele neue Formate und Zusätze, neue Ausdrucksformen und Stile, die Harvest nicht beherrscht. Dazu gehören **Frames**, **Bilder**, **grafischer Text**, **Map–Dateien**, **Javascript** und **Java**, um nur einige zu nennen.
In diesem Zusammenhang ist ein weiteres Problem von Harvest, daß **technisch zusammengehörige Dokumente**, wie eine HTML–Seite mit Style–Sheet oder HTML–Seite mit Bild und unterliegender Map–Datei, nicht zusammen geparst oder in logischer Verbindung gebracht werden können.
Dasselbe Problem gilt für **Dokumente mit logischen bzw. inhaltlichen Zusammenhang**.
Harvest bietet **keine Sprachunabhängigkeit**. In GETESS ist dies aber ein wichtiges Konzept. In den Information Retrieval–Fähigkeiten von Harvest fehlen Konzepte, wie eine konfigurierbare **Stopwortliste** oder eine **Gewichtung einzelner Dokumentstrukturen**.
In den vorherigen Kapiteln wurde immer von Parallelisierung von Sammel– und Parsing–Vorgängen gesprochen, die einzige Möglichkeit eine gewisse Parallelität von Harvest zu erreichen, ist eine verteilte Installation der Software. Dies bedeutet ein Extra an Hardware und eine höhere Netzbelastung.

Man sieht, daß Harvest Probleme in technischer Hinsicht, wie auch Probleme bei inhaltlichen Aspekten aufweist. Für eine konzeptuelle Lösung der GETESS–Anforderungen und deren prototypischen Umsetzung ist die Gatherer–Software des Harvest–Paketes eine gute Grundlage und mit einigen Veränderungen und Optimierungen durchaus für einen realen Einsatz verwendbar.

Probleme beim SMES–System
Die Probleme des SMES–Systems wurden weitestgehend mit dem Beispiel in 3.4 erläutert. Bezüglich des Gathering wäre ein mehrfaches Ansprechen des Servers wünschenswert, da man so echte **parallele Verarbeitung** erreichen kann.

Allgemeine Probleme der Integration
Mögliche Realisierungsformen wurden in Kapitel 4 erläutert. Wobei in dieser Studienarbeit eine dieser Realisierungen implementiert und genauer betrachtet wurde. Dabei ist die **Zeitproblematik** als ein großes Problem erkannt und in 4.3 beschrieben worden.
An dieser Stelle wurden keine Angaben zur **Qualität der Ergebnisse** gemacht. Das Problem liegt daran, daß die Suchmaschine GETESS noch nicht komplett, sondern nur in Teilen getestet werden kann. Um zu sehen, was bei einer Anfrage an Ergebnissen geliefert wird, sollten alle Komponenten zusammenarbeiten können. Dann wären Angaben, wie Recall und Precision eine wichtige Qualitätsbeurteilung. Um trotzdem Aussagen zur Qualität machen zu können, werden die möglichen Ergebnisse durch theoretische Annahmen erzielt.

Als interessant erweist sich der Vergleich zwischen den von Harvest gesammelten Stichworten und den stark strukturierten Parser–Ergebnisse von GETESS. Dabei sind die Parser–Ergebnisse domänenbezogen. Das bedeutet, alle Konzepte bzw. Stichworte, die nicht im Domänenlexikon enthalten sind, werden auch nicht von der Suchmaschine betrachtet. Es ist anzunehmen, daß der Index aufgrund der einfacheren und nicht domänenspezifischen Analyse einen größeren Umfang haben wird als die Parser–Ergebnisse. Dafür wird bei bei den GETESS–Ergebnissen einerseits die Qualität durch die semantische Analyse höher zu bewerten sein und anderseits die Anfragemächtigkeit höher ausfallen als beim Harvestbroker.

[2] Das WWW–Consortium befindet über die im WWW gesetzten Standards.

Kapitel 5

Zusammenfassung und Ausblick

Nach einer kurzen Zusammenfassung der vorliegenden Arbeit, wird aufbauend auf die im vorigen Kapitel genannten Ergebnisse und Probleme ein Ausblick auf weiterführende Arbeiten gegeben.

Zusammenfassung
Ausgehend von Harvest, einer Internetsuchmaschinenarchitektur, und SMES, einem natürlichsprachlichen Parser, wurden Integrationsmöglichkeiten dieser beiden Tools konzipiert. Dabei wurden Harvest und SMES in den Kapiteln 2 und 3 sehr genau vorgestellt. Genau genommen, benötigt man den Gatherer von Harvest und den "parac"–Client vom SMES–System. Harvest ist aus dem Einsatz im SWING–Projekt bekannt und SMES wurde in 3.4 in der vorliegenden Version getestet. Aus den Eigenschaften der beiden genannten Werkzeuge und den Anforderungen von GETESS (Kapitel 1) bildeten sich die in Kapitel 4 vorgeschlagenen Realisierungsmöglichkeiten heraus. Die innerhalb dieser Studienarbeit umgesetzte Variante ist eine einfache, aber effektive Lösung, die in Kapitel 4 genauer betrachtet wurde. Dabei wurde der Aufruf des Parsers und die Übergabe der gesammelten Dokumente prototypisch im Gatherer–System integriert. Einen Test kann man in 4.3 begutachten. Probleme bei den Tools und bei der Integration sind in 4.4, ein Ausblick auf weitere Fragestellungen ist in diesem Kapitel beschrieben.

Ausblick
Eine Herausforderung für die Programmierung besteht in den weiteren genannten Realisierungsmöglichkeiten. Dabei ist die Zugriffskomponente für den "Gatherd", die zweite, vorgestellte Realisierungsmöglichkeit, sehr vielversprechend. Gerade für die verteilte Installation der GETESS–Umgebung ist diese Realisierung sinnvoll.

Eine bis jetzt nicht betrachtete Problematik und damit ein interessantes Forschungsthema ist die fehlende Semantikanalyse bei dokumentübergreifenden, logischen Verbindungen.
Im Internet gibt es alle nur erdenklichen Formen von Wissensrepräsentationen. Dabei ist es durchaus so, daß Informationen, die vom Kontext her zusammen gehören (oder in einer zugehörigen Ontologie zu einem Slot gehören), auf mehrere Internet–Seiten verteilt wurden. Diese Zusammenhänge sind in den, in dieser Arbeit beschriebenen, Szenarien nicht vorgesehen.
Es gibt mehrere sinnvolle Ansätze solche Zusammenhänge zu erkennen. Ein Ansatz wäre, z.B. alle Dokumente eines Informationsanbieters zusammen zum Parser zu schicken und die Erkennung der Zusammenhänge dem SMES–System zu überlassen. Eine andere Variante ist das Kombinieren der resultierenden abstracts aus den Dokumenten eines Informationsanbieters. Dazu wäre ein Tool zu konzipieren und implementieren, welches zwischen dem Parsing und der Speicherung die beschriebene Kombination durchführen kann. Beide Varianten benötigen Informationen zum Erkennen des Kontextes von einer Menge von Daten (z.B. eines Providers). Möglichkeiten einer solchen Erkennung wären z.B. durch Auswertung der Struktur des Internet–Bereiches, durch Heuristiken der Informationsdarstellung in den Internet–Bereichen oder durch das Erkennen und Auswerten von Meta–Informationen (z.B. HTML–Elemente).
Da davon auszugehen ist, daß eine verteilte Informationshaltung von den Anbietern betrieben wird und auch mittels technischer Hilfsmittel unterstützt wird (s. 4.4), ist die Betrachtung der Kombination

oder auch Anreicherung einzelner abstracts im GETESS–Szenario ein wichtiges Thema.

Ein in 4.3 schon angedeutetes Problem ist die Synchronisation zwischen der Speicherung in Datenbanken und der Speicherung beim Gatherer (bei Harvest die gdbm–Dateien).
Der Gatherer nutzt die Metadaten seiner gespeicherten Informationen für das Update, Delete oder Refresh (Update ist das Überschreiben geänderter Dokumente, Refresh ist das allgemeine Prüfen auf Veränderungen des Dokumentes mit anschließendem Verlängern der Refresh–Time). Veränderungen in den Datensätzen des Gatherers ziehen im allgemeinen auch Änderungen in den Datenbanken nach sich.
Die Umsetzung der Metadaten und daraus resultierenden Operationen wurde in dieser Arbeit nur oberflächlich auf die Möglichkeiten hin untersucht. Für ein GETESS–Szenario mit beiden Speicherformen ist eine Synchronisation der Metadaten und der Anwendung der Speicheroperationen ein unumgänglicher Schritt.

Eine in dieser Arbeit nicht betrachtete Fragestellung ist die Sprachklassifizierung. Dies ist zum einen für das Parsing wichtig, und zum anderen sind viele Informationen im Internet in mehreren Sprachen abgelegt, wobei nur eine Sprachversion ausreicht, um die Information erfassen zu können. Problematisch sind dabei Dokumente, die Informationen in mehreren Sprachen in sich tragen.

Eine wachsende Zahl an Informationen liegt im Internet nunmehr dynamisch in Datenbanken vor. Diese Dokumente werden nur auf Anfrage generiert. Die in SWING benutzte Lösung zur Integration externer Datenbanken ist nicht auf eine abstract–Generierung abbildbar. In den hier vorgestellten Realisierungsmöglichkeiten mit Harvest bleibt die SWING–Lösung als Alternative zur abstract–Generierung erhalten.

Bei realem Einsatz der GETESS–Software ist es sinnvoll verschiedene Speichertechniken anzubieten (teils aus Anforderungen der Anwender, teils aus wirtschaftlichen Gründen). Eine kurz betrachtete Möglichkeit in 4.2 sieht die Speicherung in Harvest–internen SOIF–Daten vor. Inwiefern es sinnvoll ist, von dieser Möglichkeit Gebrauch zu machen oder andere Alternativen anbieten zu können, sind in dieser Arbeit offen gebliebene Fragestellungen.

Anhang A

HTML–Seite aus “www.all-in-all.de”

Diese HTML–Seite ist stellvertretend für ein im allgemeinen so dargestelltes Dokument in der Tourismusbranche. Das Aussehen dieser Seite stellt die Abbildung 3.2 in 3.4 dar. Diese Seite wurde automatisch aus einem Verwaltungstool des Providers heraus erstellt.
HTML–Seiten sind in einen Header und einen Body unterteilt. Dazu gibt es diverse Tags (eingeklammerte Befehlsworte), mit denen man die Ordnung und das Layout der Seite bestimmt.
Im folgenden ist die HTML–Seite mit einigen erklärenden Zwischenworten beschrieben. Für eine allgemeine Erklärung des HTML–Codes sei auf [Tol97] verwiesen.

```
<html>
```

Beginn Header.

```
<head>
```

Einige Metadaten.

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="author" content="MANET Marketing GmbH, Mecklenburg- Vorpommern und die
Ostsee">
<meta name="description" content="Das Atrium Hotel Krüger in Rostock - Mecklenburg
erleben - Alles in einem! - Informationen sinnvoll vernetzt.MANET">
<meta name="expires" content="07 Sep. 2000 12:00:00 GMT">
<meta name="GENERATOR" content="Microsoft FrontPage 3.0">
<meta name="keywords" content="Rostock, Mecklenburg, Tourismus, Tagung, Hotels,
Zimmer, Reservierung, Hansestadt, Ostsee, Küste, Atrium Hotel Krüger">
<meta name="lang" content="de, ch, at">
<meta name="revisit-after" content="31 days">
<meta name="robots" content="all">
<meta name="timestamp" content="990907 163644">
```

Der Titel des Dokuments.

```
<title>Das Atrium Hotel Krüger in Rostock - Urlaub und Reisen in Mecklenburg
</title>
```

Ein JavaScript–Programm mit Unterscheidung zwischen verschiedenen Browsern.

```
<script language="JavaScript"><!--
<!--
function PopUp()
{

if (navigator.appName == 'Netscape')
{
var url='bilderbogen/b1127.htm';
var win='left=290,top=150,toolbar=0,directories=0,menubar=0,scrollbars=0,
resizable=0,width=369,height=249';
open(url,'banner', win);
}
else //internet explorer oder andere
{var url='bilderbogen/b1127.htm';
var win='left=290,top=150,toolbar=0,directories=0,menubar=0,scrollbars=0,
resizable=0,width=350,height=234';
open(url,'banner', win);
}
}
// --></script>
```

Ende des Headers.

```
</head>
```

Beginn des Bodies mit einigen Grundeinstellungen zur Farbe etc.

```
<body background="grund.gif" bgcolor="#F7F7ED" text="#000000" link="#008000"
vlink="#008000" alink="#FF0000">
```

Dieser Block ist als Tabelle dargestellt und wird zur Darstellung von Icons in einer bestimmten Anordnung benutzt.

```
<div align="center"><center>
<table border="0" cellpadding="2" width="620">
<tr>
<td><p align="left"><a href="index.htm" target="_top"><img src="start.gif"
alt="Tourismus-Information - Startseite" align="left" border="0" hspace="0"
WIDTH="148" HEIGHT="24"></a></td>
<td><p align="right">[<a href="suche.htm">Suche</a>] [<a href="english/1127.htm">
English</a>]</td>
</tr>
</table>
</center></div>
```

Dieser Block bezeichnet dieselbe Funktion wie der vorherige.

```
<div align="center"><center>
<table border="0" cellpadding="2" width="640">
```

```
<tr>
<td><font size="4"><b><a href="http://www.all-in-all.com/fax/fax.cgi?form=fax2&amp
;firmid=F2ATRIUMROSTOCK"><img src="fax.gif" alt="Kontaktaufnahme, Buchungsanfrage,
Atrium Hotel Krüger" border="0" WIDTH="90" HEIGHT="30"></a><a href="tb030.htm">
<img src="esstrink.gif" alt="Essen &amp; Trinken, Hansestadt Rostock, Mecklenburg"
border="0" WIDTH="90" HEIGHT="30"></a><a href="9122.htm"><img src="wassersp.gif"
alt="Wassersport, Hansestadt Rostock, Mecklenburg" border="0" WIDTH="90" HEIGHT=
"30"></a><a href="tb042.htm"><img src="sportfit.gif" alt="Sport &amp; Fitneß,
Hansestadt Rostock, Mecklenburg" border="0" WIDTH="90" HEIGHT="30"></a><a href=
"tb048.htm"><img src="kulturkl.gif" alt="Kultur, Hansestadt Rostock, Mecklenburg"
border="0" WIDTH="90" HEIGHT="30"></a><a href="9192.htm"><img src="sehenskl.gif"
alt="Sehenswertes, Hansestadt Rostock, Mecklenburg" border="0" WIDTH="90" HEIGHT=
"30"></a><a href="9138.htm"><img src="touren.gif" alt="Angebote, Hansestadt
Rostock, Mecklenburg" border="0" WIDTH="90" HEIGHT="30"></a></b></font></td>
</tr>
</table>
</center></div>
```

Dieser Block ist wieder als Tabelle dargestellt. Hier findet man teilweise Bilder und die Hauptinformationen der Seite, mittig dargestellt.

```
<div align="center"><center>
<table border="0" cellpadding="2" cellspacing="7" width="620">
<tr>
<td align="center"><font size="4" face="Arial"><img src="bild/b1127_logo.gif" alt=
"Logo Atrium Hotel Krüger" align="left" hspace="10" vspace="5" WIDTH="120" HEIGHT=
"80"></font><font size="4"><br>
</font></td>
<td valign="top"><p align="center"><b><i><font size="5" color="#000080">Atrium
Hotel Krüger </font><font size="1" color="#000080"><br>
</font></i></b><font size="1"><br>
</font><font color="#000040" size="3">D-18069 Rostock-Sievershagen <br>
Ostsee-Park-Str. 2 <br>
Tel:(0381)800 23 43  Fax:(0381)800 23 42 </font></td>
<td align="center"><b><a href="javascript:PopUp()"><font size="4" face="Arial">
<img src="bild/b1127_1k.gif" alt="Atrium Hotel Krüger" align="left" border="0"
hspace="10" vspace="5" WIDTH="120" HEIGHT="80"></font></a><a href="g43g.htm"><font
size="4"><br>
</font></a></b></td>
</tr>
</table>
</center></div>
```

Ein weiterer Block, als Tabelle dargestellt, enthält die eigentlichen Informationen dieses Dokumentes.

```
<div align="center"><center>
<table border="0" cellpadding="2" cellspacing="7" width="620">
<tr>
<td><hr>
</td>
</tr>
```

```
<tr>
<td valign="top"><b><font size="3" color="#000080">Lage: </font><font color=
"#000040" size="3"><br>
</b>Das von der Familie Krüger geführte GARNI-HOTEL im Westen der</font><font
size="3"> </font><a href="1022.htm"><font color="#008000" size="3"><strong>
Hansestadt Rostock</strong></font></a><font size="3"> </font><font color="#000040"
size="3">präsentiert sich als ein modernes Geschäfts- und Messehotel. <br>
Touristen oder Geschäftsreisende schätzen die angenehme und freundliche Atmosphäre.
</font><b><font size="3"><br>
<font color="#000080">Hausbeschreibung: </font></font><font color="#000040" size=
"3"><br>
</b>Die 59 Komfortzimmer und Appartements sind mit Bad oder Dusche/WC, Telefon,
TV, <b>Radio, </b>Minibar und Fön ausgestattet. </font><font size="3"><br>
<b><font color="#000080">Preise:</font> </font><font color="#000040" size="3">
(inclusive Frühstück) <br>
</b>Einzelzimmer: von 95,00 - 125,00 DM  Doppelzimmer: 150,00 DM, <br>
Dreibettzimmer: 185,00 DM  Appartements: von 135,00 - 190,00 DM </font></td>
</tr>
<tr>
<td valign="top"><font color="#000080" size="3"><b>Bei uns willkommen: </b></font>
<font color="#000040" size="3">American-Express, Visa-Card, Euro-Card, Diners-Club
</font><font size="3"> </font></td>
</tr>
<tr>
<td valign="top"><font size="3"><b><font color="#000080">Besonderheiten:</font>
<br></b></font>
<font color="#000040" size="3">Tagungen und Seminare können bis zu einer Kapazität
von 40 Personen durchgeführt werden. Das Haus ist idealer Ausgangspunkt, um den
Ostseestrand (5 km) und die reizvolle Umgebung bequem zu erreichen. </font></td>
</tr>
</table>
</center></div>
```

Zum Schluß sind hier noch Informationen zum Provider und zum Copyright.

```
<p align="center"><img src="copy.gif" WIDTH="555" HEIGHT="20"><br>
<i><font size="2">Copyright (c) <!--webbot bot="Timestamp" s-type="REGENERATED"
s-format="%B %Y" startspan -->September 1999<!--webbot bot="Timestamp" endspan
i-checksum="29531" --> </font><a href="0702.htm">MANET</a><font size="2">
Marketing GmbH, Schwerin; Fax: +49-(0)385-3993-411</i> </font></p>
```

Ende des Bodies und des Dokumentes.

```
</body>
</html>
```

Anhang B

SMES–Ergebnis der Beispiel–HTML–Seite

Hier ist ein Ausschnitt des Ergebnisses von SMES nach Parsing–Anfrage mit der HTML–Seite aus 3.4 zu finden. Es sind nur die 20 Relationen aus der Tabelle 3.1 dargestellt. Aufgrund der Größe des kompletten XML–Dokumentes (75 Relationen) wurde die restlichen Relationen nicht aufgeführt. Zur besseren Orientierung sind die Relationen nach den Nummern in der Tabelle 3.1 durchnumeriert. Zum Verständnis der XML–Tags sei auf [BM98] verwiesen.

```
<?xml version='1.0' encoding='ISO-8859-1' standalone='yes'?>
<list>
```

1. Rel.

```
<tuple name='getess-output' rel='liegt_in_Stadt'>
<fragment> <type>Hotel</type> <inst>Hotel_1</inst> <name>hotel</name> </fragment>
<fragment> <type>Stadt</type> <inst>Stadt_1</inst> <name>Rostock</name> </fragment>
<constraint type='PREP'>in</constraint> </tuple>
```

Zur 1. Rel., nur anderes Erkennungsmerkmal.

```
<tuple name='getess-output' rel='liegt_in_Stadt'>
<fragment> <type>Hotel</type> <inst>Hotel_1</inst> <name>hotel</name>
</fragment>
<fragment> <type>Stadt</type> <inst>Stadt_1</inst> <name>Rostock</name> </fragment>
<constraint type='HEURISTIC'>NP-PP-HEURISTIC</constraint>
</tuple>
```

Ebenfalls 1. Rel., anderes Erkennungsmerkmal.

```
<tuple name='getess-output' rel='liegt_in_Stadt'>
<fragment> <type>Hotel</type> <inst>Hotel_1</inst> <name>hotel</name> </fragment>
<fragment> <type>Stadt</type> <inst>Stadt_1</inst> <name>Rostock</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

2. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Hotel</type> <inst>Hotel_1</inst> <name>hotel</name> </fragment>
<fragment> <type>Region</type> <inst>Region_1</inst> <name>Mecklenburg</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

3. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Stadt</type> <inst>Stadt_1</inst> <name>Rostock</name> </fragment>
<fragment> <type>Region</type> <inst>Region_1</inst> <name>Mecklenburg</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

4. Rel.

```
<tuple name='getess-output' rel='hat_Adresse'>
<fragment> <type>Hotel</type> <inst>Hotel_2</inst> <name>hotel</name> </fragment>
<fragment> <type>Adresse</type> <inst>Adresse_1</inst> <name>fax</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

5. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Ferienwohnung</type> <inst>Ferienwohnung_1</inst> <name>appartement</name>
</fragment>
<fragment> <type>Telefon</type> <inst>Telefon_1</inst> <name>telefon</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

6. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Ferienwohnung</type> <inst>Ferienwohnung_1</inst> <name>appartement</name>
</fragment>
<fragment> <type>Fernseher</type> <inst>Fernseher_1</inst> <name>tv</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

7. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Ferienwohnung</type> <inst>Ferienwohnung_1</inst> <name>appartement</name>
</fragment>
<fragment> <type>Radio</type> <inst>Radio_1</inst> <name>radio</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

8. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Ferienwohnung</type> <inst>Ferienwohnung_1</inst> <name>appartement</name>
</fragment>
<fragment> <type>Foen</type> <inst>Foen_1</inst> <name>foen</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

9. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Telefon</type> <inst>Telefon_1</inst> <name>telefon</name> </fragment>
<fragment> <type>Fernseher</type> <inst>Fernseher_1</inst> <name>tv</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
```

```
</tuple>
```

10. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Telefon</type> <inst>Telefon_1</inst> <name>telefon</name> </fragment>
<fragment> <type>Radio</type> <inst>Radio_1</inst> <name>radio</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

11. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Telefon</type> <inst>Telefon_1</inst> <name>telefon</name> </fragment>
<fragment> <type>Foen</type> <inst>Foen_1</inst> <name>foen</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

12. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Fernseher</type> <inst>Fernseher_1</inst> <name>tv</name> </fragment>
<fragment> <type>Radio</type> <inst>Radio_1</inst> <name>radio</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

13. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Fernseher</type> <inst>Fernseher_1</inst> <name>tv</name> </fragment>
<fragment> <type>Foen</type> <inst>Foen_1</inst> <name>foen</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

14. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Radio</type> <inst>Radio_1</inst> <name>radio</name> </fragment>
<fragment> <type>Foen</type> <inst>Foen_1</inst> <name>foen</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

15. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Unterkunft</type> <inst>Unterkunft_1</inst> <name>fruehstueck</name>
</fragment>
<fragment> <type>Einbettzimmer</type> <inst>Einbettzimmer_1</inst> <name>einzelzimmer</name>
</fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

16. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Unterkunft</type> <inst>Unterkunft_1</inst> <name>fruehstueck</name>
</fragment>
<fragment> <type>Geld</type> <inst>Geld_1</inst> <name>dm</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
```

```
</tuple>
```

17. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Unterkunft</type> <inst>Unterkunft_1</inst> <name>fruehstueck</name>
</fragment>
<fragment> <type>Zweibettzimmer</type> <inst>Zweibettzimmer_1</inst>
<name>doppelzimmer</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

18. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Unterkunft</type> <inst>Unterkunft_1</inst> <name>fruehstueck</name>
</fragment>
<fragment> <type>Geld</type> <inst>Geld_2</inst> <name>dm</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

19. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Unterkunft</type> <inst>Unterkunft_1</inst> <name>fruehstueck</name>
</fragment>
<fragment> <type>Mehrbettzimmer</type> <inst>Mehrbettzimmer_1</inst>
<name>dreibettzimmer</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
```

20. Rel.

```
<tuple name='getess-output'>
<fragment> <type>Unterkunft</type> <inst>Unterkunft_1</inst> <name>fruehstueck</name>
</fragment>
<fragment> <type>Geld</type> <inst>Geld_3</inst> <name>dm</name> </fragment>
<constraint type='HEURISTIC'>SENTENCE-HEURISTIC</constraint>
</tuple>
</list>
```

Anhang C

Ausschnitt aus dem Gatherer–Log

Es werden hier einige Ausschnitte aus dem Logfile (insgesamt ist das Logfile knapp 400 KiloByte groß) des Gathering–Laufes, der in Kapitel 4 beschrieben wurde, gezeigt. Es sind die Ausschriften, die das Essence–System beim Sammeln der Dokumente und beim Summarizing der Informationen gemacht hat.
In der ersten Spalte ist die verantwortliche Komponente dargestellt, gefolgt von Datum und Uhrzeit. Dann kommt die durchgeführte Operation.

Im ersten Ausschnitt wird der Gatherer gestartet. Es werden dazu einige Angaben, wie Name, Gatherer–Rechner, Gatherer–Version, gemacht. Dann wird das Sammeln über den "enum"–Schritt gestartet. Die darauffolgende Warnung ist das Ergebnis des erstmaligen Anlaufens der Software, so daß bestehende Daten nicht in diesem Gathering–Prozeß berücksichtigt werden. Dies sagen auch die danach kommenden Ausschriften über fehlende Datenfiles aus. Diese Warnung wird — wie gesagt — ausschließlich beim ersten Gatherer–Lauf ausgegeben. Es werden nun sämtliche in der Konfiguration angegebene URL's durchsucht und je nach Konfiguration die in den URL's gefundenen Links weiterverfolgt. Bei den gesammelten Dokumenten wird am Ende des Logeintrags der Dokumenttyp eingetragen.
Begonnen wird die Suche mit www.all-in-all.de. Einige der in all-in-all.de gefundenen Links konnten nicht aufgelöst werden, wie an den Ausschriften zu erkennen. Wie man sieht, liegt es an Javascript–Programmen innerhalb des Dokumentes, die Harvest nicht verstehen kann.

```
 essence: 19991203 21:00:03: Running Gatherer...
 essence: 19991203 21:00:03: Gatherer-Name:     Getess-Harvest
 essence: 19991203 21:00:03: Gatherer-Host:     zeus
 essence: 19991203 21:00:03: Gatherer-Version:  1.5.19
    enum: 19991203 21:00:03: Starting RootNode enumeration.
 essence: 19991203 21:00:03: WARNING: Incremental Gatherering will NOT be supported on this
                             run.
 essence: 19991203 21:00:03:    unable to locate these database(s) needed for incremental
                                gatherering:
 essence: 19991203 21:00:03:    /users/db18/getess/harvest/gatherers/getess/data/
                                PRODUCTION.gdbm
 essence: 19991203 21:00:03:    /users/db18/getess/harvest/gatherers/getess/data/INDEX.gdbm
 essence: 19991203 21:00:03:    /users/db18/getess/harvest/gatherers/getess/data/MD5.gdbm
httpenum: 19991203 21:00:03: Processing URL 'http://www.all-in-all.de/':
 essence: 19991203 21:00:07: http://www.all-in-all.de/  HTML
httpenum: 19991203 21:00:11: url_parse: Unknown URL scheme: javascript
httpenum: 19991203 21:00:11: url_parse: Unknown URL scheme: javascript
httpenum: 19991203 21:00:11: url_parse: Unknown URL scheme: javascript
httpenum: 19991203 21:00:16: url_parse: Unknown URL scheme: javascript
httpenum: 19991203 21:00:16: url_parse: Unknown URL scheme: javascript
httpenum: 19991203 21:00:25: url_parse: Unknown URL scheme: javascript
httpenum: 19991203 21:00:25: url_parse: Unknown URL scheme: javascript
```

```
httpenum: 19991203 21:00:25: url_parse: Unknown URL scheme: javascript
httpenum: 19991203 21:00:33: url_parse: Unknown URL scheme: javascript
httpenum: 19991203 21:00:33: url_parse: Unknown URL scheme: javascript
 essence: 19991203 21:00:50: http://www.all-in-all.de/0502.htm  HTML
httpenum: 19991203 21:01:19: url_parse: Unknown URL scheme: javascript
```

Gerade die ersten gesammelten Seiten beinhalteten viele Javascript–Elemente. Das liegt daran, daß die Eingangsseiten eines Informationsanbieters meist besonders geschmackvoll gestaltet sind und damit viele neue, effektvolle Gestaltungsmittel verwendet werden. Die etwas später gesammelten und damit in tieferen Sammelebenen gefundenen Dokumente haben weniger solche Effekte bzw. besitzen wiederkehrende Elemente, die nur einmal gesammelt werden.
In den nun folgenden Ausschnitt sind auch die in 4.3 beschriebenen Sammelzeiten gut nachvollziehbar.

```
 essence: 19991204 11:24:38: http://www.all-in-all.de/9172.htm  HTML
 essence: 19991204 11:25:25: http://www.all-in-all.de/9028.htm  HTML
 essence: 19991204 11:27:32: http://www.all-in-all.de/9032.htm  HTML
 essence: 19991204 11:28:12: http://www.all-in-all.de/9040.htm  HTML
 essence: 19991204 11:28:55: http://www.all-in-all.de/9068.htm  HTML
 essence: 19991204 11:29:50: http://www.all-in-all.de/9023.htm  HTML
 essence: 19991204 11:30:29: http://www.all-in-all.de/9078.htm  HTML
 essence: 19991204 11:31:24: http://www.all-in-all.de/9249.htm  HTML
 essence: 19991204 11:32:08: http://www.all-in-all.de/9051.htm  HTML
 essence: 19991204 11:32:49: http://www.all-in-all.de/9169.htm  HTML
 essence: 19991204 11:33:41: http://www.all-in-all.de/9049.htm  HTML
 essence: 19991204 11:35:50: http://www.all-in-all.de/9011.htm  HTML
 essence: 19991204 11:36:37: http://www.all-in-all.de/9173.htm  HTML
 essence: 19991204 11:37:57: http://www.all-in-all.de/9022.htm  HTML
 essence: 19991204 11:38:35: http://www.all-in-all.de/9399.htm  HTML
 essence: 19991204 11:39:15: http://www.all-in-all.de/9413.htm  HTML
 essence: 19991204 11:41:02: http://www.all-in-all.de/9177.htm  HTML
 essence: 19991204 11:41:44: http://www.all-in-all.de/9466.htm  HTML
 essence: 19991204 11:42:24: http://www.all-in-all.de/9071.htm  HTML
 essence: 19991204 11:44:54: http://www.all-in-all.de/9515.htm  HTML
 essence: 19991204 11:46:28: http://www.all-in-all.de/9094.htm  HTML
 essence: 19991204 11:47:18: http://www.all-in-all.de/9123.htm  HTML
 essence: 19991204 11:48:07: http://www.all-in-all.de/9034.htm  HTML
 essence: 19991204 11:48:53: http://www.all-in-all.de/9041.htm  HTML
 essence: 19991204 11:49:41: http://www.all-in-all.de/9176.htm  HTML
 essence: 19991204 11:50:27: http://www.all-in-all.de/9078_1.htm        HTML
 essence: 19991204 11:51:24: http://www.all-in-all.de/9468.htm  HTML
 essence: 19991204 11:52:09: http://www.all-in-all.de/9300.htm  HTML
```

Der letzte Ausschnitt zeigt den erfolgreichen Abschluß des Gathering–Vorganges.

```
 essence: 19991207 14:44:26: http://www.all-in-all.de/english/hotel-crowne-plaza/
                             weekend1.html  HTML
 essence: 19991207 14:45:12: http://www.all-in-all.de/english/hotel-crowne-plaza/
                             weekend2.html  HTML
 essence: 19991207 14:45:54: http://www.all-in-all.de/english/hotel-crowne-plaza/
                             weekend3.html  HTML
 essence: 19991207 14:46:36: http://www.all-in-all.de/english/hotel-crowne-plaza/
                             weekend4.html  HTML
 essence: 19991207 14:47:22: Terminated normally.
```

Literaturverzeichnis

[BKKL99] Bager, Jo, Axel Kossel, Stefan Karzauninkat und Sven Lennartz: *Suchmaschinen — Orientierungslose Infosammler; Preissuchen; Zielfahndung; Ich bin wichtig!* c't — Magazin für Computertechnik, 23:158–186, 1999.

[BM98] Behme, Henning und Stefan Mintert: *Extensible Markup Language (XML) — in der Praxis.* Addison Wesley Longman Verlag GmbH, Bonn, 1998.

[BPW+98] Bruder, Ilvio, Jan Pretzel, Burkhard Wrenger, Bernd Prager, Günter Neumann, Christian Braun, Hans Uszkoreit, Hans-Peter Schnurr, Steffen Staab, Rudi Studer, Antje Düsterhöft, Meike Klettke und Andreas Heuer: *BMBF–Projekt GETESS: GErman Text Exploration and Search System — Arbeitsbericht zum Meilenstein 1.* Meilensteinbericht 1, GECKO mbH Rostock, DFKI Saarbrücken, Universität Karlsruhe und Universität Rostock, Dezember 1998.

[Bus90] Bussmann, Hadumod: *Lexikon der Sprachwissenschaft.* Alfred Kröner Verlag, zweite, neubearb. Auflage, 1990.

[BYRN99] Baeza-Yates, Ricardo und Berthier Ribeiro-Neto: *Modern Information Retrieval.* ACM Press. Addison Wesley, New York, 1999.

[BZW98] Brenner, Walter, Rüdiger Zarnekow und Hartmut Wittig: *Intelligente Softwareagenten.* Springer Verlag, Berlin, 1998.

[Cro97] Croft, W. Bruce (Herausgeber): *Transactions on Information Systems*, Band 15 der Reihe *ACM*, New York, 1997. ACM, Inc.

[FDES97] Fensel, Dieter, Stefan Decker, Michael Erdmann und Rudi Studer: *Ontobroker: Transforming the WWW into a Knowledge Base.* Technical Report, University of Karlsruhe, Institute AIFB, Karlsruhe, Oktober 1997.

[Fuh97] Fuhr, Norbert: *Information Retrieval, Skriptum zur Vorlesung.* Vorlesungsskript, Universität Dortmund, Dortmund, Mai 1997. Kap. 1-3 Einführung und Konzepte, Kap. 9 IR and Databases.

[Gar96] Garlipp, Kai: *Hinweise zur Installation von Harvest 1.4.* Universität Rostock, Fachbereich Informatik, Lehrstuhl Datenbank– und Informationssysteme, November 1996.

[Gun96] Gundavaram, Shishir: *CGI Programming on the World Wide Web.* O'Reilly & Associates, Inc., Cambridge, 1996.

[GW93] Goerz, G. und W. Wahlster: *Sprachverarbeitung: Einleitung und Überblick.* In: Goerz, G. (Herausgeber): *Einführung in die künstliche Intelligenz.* Addison Wesley, Bonn, 1993.

[HMW99] Heuer, Andreas, Holger Meyer und Gunnar Weber: *SWING: Die Suchmaschine des Landesinformationssystems MV-Info.* Technischer Bericht, Universität Rostock, Fachbereich Informatik, Rostock, Juni 1999.

[HS95] Hardy, Darren R. und Michael F. Schwartz: *Customized Information Extraction as a Basis for Resource Discovery.* Technical Report, Department of Computer Science, University of Colorado, Boulder, February 1995.

[HSW96] Hardy, Darren R., Michael F. Schwartz und Duane Wessels: *Harvest Users Manual.* University of Colorado at Boulder, Department of Computer Science, January 1996.

[LDHM97] Langer, Uwe, Antje Düsterhöft, Andres Heuer und Holger Meyer: *SWING: Ein Anfrage– und Suchdienst im Internet.* Rostocker Informatik–Bericht, Fachbereich Informatik, Universität Rostock, Rostock, 1997.

[MPP+99] Melzig, Siegfried, Bernd Prager, Jan Pretzel, Burkhard Wrenger, Günter Neumann, Christian Braun, Hans Uszkoreit, Alexander Mädche, Hans-Peter Schnurr, Steffen Staab, Rudi Studer, Antje Düsterhöft, Meike Klettke, Denny Priebe und Andreas Heuer: *BMBF–Projekt GETESS: GErman Text Exploration and Search System — Arbeitsbericht zum Meilenstein 2.* Meilensteinbericht 2, GECKO mbH Rostock, DFKI Saarbrücken, Universität Karlsruhe und Universität Rostock, Juli 1999.

[MWJ96] Müller, Jörg P., Michael J. Wooldridge und Nicholas R. Jennings (Herausgeber): *Intelligent Agents III - Agent Theories, Architectures, and Languages*, Nummer 1193 in *Lecture Notes in Artificial Intelligence*, August 1996.

[NM98] Neumann, Günter und Giampaolo Mazzini: *Domain–adaptive Information Extraction.* Draft Version, DFKI Saarbrücken, Saarbrücken, Germany, August 1998.

[Por98] Porst, Beate: *Konzeption und Implementierung eines Abonnementdienstes für die SWING-Suchmaschine.* Studienarbeit, Universität Rostock, Fachbereich Informatik, 1998.

[San95] Santifaller, Michael: *TCP/IP und ONC/NFS in Theorie und Praxis — Internetworking mit UNIX.* Addison Wesley Publishing Company, Bonn, dritte Auflage, 1995.

[SH99] Saake, Gunter und Andreas Heuer: *Datenbanken: Implementierungstechniken.* MITP–Verlag GmbH, Bonn, 1999.

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

[Tol97] Tolksdorf, Robert: *Die Sprache des Web: HTML 4.* dpunkt Verlag, Heidelberg, dritte Auflage, 1997.