Studiengang:	Informatik
Prüfer:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer:	Dipl.-Inf. Matthias Großmann
begonnen am:	01. Oktober 2002
beendet am:	31. März 2003
CR-Klassifikation:	H.2.8, H.3.1, H.3.3

Diplomarbeit Nr. 2049

Ortsbezug für Web-Inhalte

Mihály Jakob

Institut für Parallele und Verteilte Systeme (IPVS)
Abteilung Anwendersoftware
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Kurzfassung

Das Ziel der Forschergruppe NEXUS ist die Entwicklung einer offenen und globalen Infrastruktur für ortsbezogene Anwendungen. Das Weltmodell, das diesen Anwendungen zu Grunde liegt, das *Augmented World Model*, ist eine, durch virtuelle Objekte erweiterte, Abbildung der realen Welt. Virtuelle Objekte, wie zum Beispiel Webseiten des *World Wide Webs*, speichern Informationen, die dem mobilen Anwender jederzeit zur Verfügung stehen sollen. Anwendungen sollen diese Informationen, abhängig von der geographischen Position des Anwenders, darstellen.

Das Ziel dieser Diplomarbeit, im Rahmen des Projekts NEXUS, ist die Untersuchung und die Implementierung von Verfahren, die die Sammlung von Webseiten mit Ortsinformationen ermöglichen. Nach der Darstellung von Ortsinformationsarten und Webseitenkomponenten werden Erfolg versprechende Verfahren ausgewählt und in der Form der Analysekomponente eines Webroboters implementiert. Die Präsentation von Testergebnissen und die Bewertung der implementierten Lösung schließt die Arbeit ab.

Danksagung

An dieser Stelle möchte ich meinem Betreuer, Herrn Dipl.-Inf. Matthias Großmann, danken. Er stand mir bei der Erstellung dieser Diplomarbeit stets hilfreich zur Seite.

Inhaltsverzeichnis

KAPITEL 1 Einführung 1

1.1 Motivation 3
1.2 Aufgabenstellung 3
1.3 Überblick 4

KAPITEL 2 Ortsinformationen in Webseiten 5

2.1 Einführung 7
2.2 Ortsinformationen 7
2.2.1 Natürliche geographische Stätten 7
2.2.2 Bauwerke 8
2.2.3 Politische Gebietsaufteilungen 9
2.2.4 Administrative Gebietsaufteilungen 10
2.3 Geodaten 10
2.3.1 The Getty Thesaurus of Geographic Names 11
2.3.2 Datafactory Geocode 14
2.3.3 GeoBase 15
2.4 Ortsinformationen in Webseitenkomponenten 18
2.4.1 Die URL 19
2.4.2 Der Titel 23
2.4.3 Meta-Tags 23
2.4.4 Webseitentext 27
2.4.5 Hyperlinks 30
2.5 Zusammenfassung 31

KAPITEL 3 DCbot 33

3.1 Zielsetzung 35
3.2 Vorgehensweise 35
3.3 Entwicklungsumgebung 36
3.4 Eigenschaften des Webroboters 36
3.4.1 Systemarchitektur 36
3.4.2 Funktionsumfang 37
3.4.3 Konfigurationsmöglichkeiten 42
3.4.4 Rückmeldungen 51
3.4.5 Ergebnisspeicherung 52
3.5 Implementierungsdetails 55
3.5.1 Allgemeine Vorgehensweise 57
3.5.2 Die Klasse *Candidate* *57*
3.5.3 Die Klasse *Address* *59*
3.5.4 Die Klasse *Rule* *60*
3.5.5 Die Klasse *RuleFileHandler* *61*

3.5.6 Die Klasse *DBI_JDBC* *63*
3.5.7 Die Klasse *HTMLHandler* *65*
3.5.8 Die Klasse *Statistics* *68*

KAPITEL 4 Bewertung des Systems 69

4.1 Einleitung 71
4.2 Testumgebung *71*
4.3 Testergebnisse *72*
4.3.1 Erster Testlauf *73*
4.3.2 Zweiter Testlauf *75*
4.3.3 Semantische Regeln *77*
4.4 Fazit *77*

KAPITEL 5 Zusammenfassung 79

ANHANG A DENIC-Einträge 83

A.1 DENIC-Eintrag für *„stuttgart.de“* *85*
A.2 DENIC-Eintrag für *„zugspitze.de“* *86*
A.3 DENIC-Eintrag für *„t-online.de“* *87*

ANHANG B Beschreibung von *TGN*-Tabellen 89

B.1 Eintrag für *„Stuttgart“* aus dem *TGN* *91*
B.2 Wichtigste Tabellen des *TGN* *92*

ANHANG C DCbot-Dateien 95

C.1 Konfigurationsdatei *97*
C.2 Statistik-Datei des ersten Testlaufs *98*
C.3 Statistik-Datei des zweiten Testlaufs *100*

Literaturverweise 103

Abbildungsverzeichnis

Abbildung 2-1: Modellierung einer geographischen Stätte 8
Abbildung 2-2: Grundrisse von Eiffelturm, Golden Gate Bridge und Kölner Dom 9
Abbildung 2-3: Entity-Relationship Modell des *TGN* *12*
Abbildung 2-4: Geographische Topologie von *GeoBase*. *15*
Abbildung 2-5: Entity-Relationship Modell von *GeoBase* *16*
Abbildung 3-1: Architektur von *DCbot* *37*
Abbildung 3-2: Klassenübersicht von *DCbot* *56*
Abbildung 3-3: Darstellung zweier Speicherungsstrukturen 62

Tabellenverzeichnis

Tabelle 2-1: *GeoBase*-Tabelle PLACES 16
Tabelle 2-2: *GeoBase*-Tabelle NAMES 17
Tabelle 2-3: *GeoBase*-Tabelle PLACETYPES 17
Tabelle 2-4: Vorkommenshäufigkeiten von Standard-Meta-Tags 25
Tabelle 2-5: Vorkommenshäufigkeiten von *DCMI*-Meta-Tags 26
Tabelle 4-1: Erreichbarkeitskennzahlen des ersten Testlaufs 73
Tabelle 4-2: Ortsinformationsverteilung des ersten Testlaufs 74
Tabelle 4-3: Ortstypsverteilung des ersten Testlaufs 74
Tabelle 4-4: Erreichbarkeitskennzahlen des zweiten Testlaufs 75
Tabelle 4-5: Ortsinformationsverteilung des zweiten Testlaufs 76
Tabelle 4-6: Ortstypsverteilung des zweiten Testlaufs 77
Tabelle B-1: *TGN*-Tabelle GEOG_MAIN 92
Tabelle B-2: *TGN*-Tabelle GEOG_PARENTS 93
Tabelle B-3: *TGN*-Tabelle GEOG_NAME 93
Tabelle B-4: *TGN*-Tabelle GEOG_PLACETYPE 94
Tabelle B-5: *TGN*-Tabelle PLACETYPE_LIST 94

KAPITEL 1 EINFÜHRUNG

Dieses Kapitel gibt einen Überblick über die zu realisierende Aufgabe und skizziert die zur Lösung führende Vorgehensweise.

1.1 Motivation

In den letzten Jahren hat sich das World Wide Web für viele Menschen zur wichtigsten Informationsquelle entwickelt. Vielfältige Informationen sind rund um die Uhr für jedermann mit einem Internetanschluß zugänglich. Für Unternehmen und Dienstleister ist eine Präsenz im World Wide Web mittlerweile zwingend notwendig, wenn sie nicht rückständig erscheinen möchten. Aber auch nicht-kommerzielle Organisationen und Privatpersonen stellen sich gerne im Web dar. Es wurden schon viele Superlative über das Internet ausgesprochen, deren Liste hier nicht fortgeführt werden muss. Sicherlich hat es aber die letzten Jahre sowohl die Wirtschaft als auch das Leben der Menschen auf der ganzen Welt nachhaltig beeinflusst.

Es wäre wünschenswert, die vielfältigen Daten aus dem World Wide Web nicht nur zu Hause, sondern auch unterwegs nutzen zu können. Obwohl Mobiltelefone und PDAs, also die benötigte Hardware, weit verbreitet sind, haben sich mobile Internetdienste in Europa noch nicht durchsetzen können. Erste Versuche durch das WAP stoßen wegen der langsamen Übertragung über GSM und der hohen Kosten auf geringe Akzeptanz.

In der nahen Zukunft werden neue Mobilfunksysteme, wie GPRS und UMTS, durch deutlich höhere Übertragungsraten eine neue Dimension der Möglichkeiten auf dem Gebiet der mobilen Internetdienste eröffnen. Qualitativ hochwertige Anwendungen werden dann sicherlich eine größere Akzeptanz als die heutigen Lösungen finden. Innovative Ideen, wie zum Beispiel das Anbieten ortsbezogener Informationen, können dann umgesetzt werden.

Die Forschergruppe NEXUS entwickelt eine offene und globale Infrastruktur, die ortsbezogene Anwendungen beim Anbieten solcher Informationen unterstützt. Diese Arbeit soll, im Rahmen des Projekts NEXUS, zu der Entwicklung zukunftweisender Technologien einen wertvollen Beitrag leisten.

1.2 Aufgabenstellung

Eine wichtige Aufgabe im Rahmen des Projekts ist die Einbindung externer Informationsquellen, wie dem World Wide Web, in die NEXUS Infrastruktur. Mobile Anwendungen sollen dadurch in der Lage sein, dem Benutzer, zu einer bestimmten geographischen Position gehörende Webseiten, zu präsentieren. In der Studienarbeit *"Ortsbasierter Web-Zugriff"* [Süt01] wurde die Möglichkeit untersucht, Ortsinformationen aus den Meta-Tags der Webseiten zu extrahieren. Die geringe Verbreitung von Webseiten mit umfangreichen Meta-Informationen hat diesen Lösungsansatz nur begrenzten Erfolg erzielen lassen.

Das Ziel dieser Diplomarbeit ist die Untersuchung, Analyse und Bewertung von Verfahren, die einen Ortsbezug mit auf Webseiten bereits vorhandener Information herstellen können.
Ein denkbarer Ansatz für diese Aufgabe wäre zum Beispiel, das Nachschlagen möglicher Ortsnamen in einem Katalog, wie dem *The Getty Thesaurus of Geographic Names*.

Diese Ortsnamen könnte man aus Meta-Tags oder aus dem Inhalt der Webseite ermitteln.
Eine weitere Lösungsmöglichkeit besteht in der Herstellung des Ortsbezugs über die Information eines *Domain Name Service*, oder nach der Ermittlung der IP-Adresse, in der Befragung einer *Whois*-Datenbank.
Zusätzlich bietet die Analyse von Linkstrukturen um Webseiten mit bekanntem Ortsbezug die Möglichkeit, weitere Webseiten mit Ortsinformationen zu finden.

Nach der Ausarbeitung möglicher Lösungsansätze sollen die erfolgsversprechendsten Verfahren implementiert, getestet und bewertet werden.

1.3 Überblick

Die im vorhergehenden Abschnitt vorgestellten Aufgaben werden auf drei Hauptblöcke verteilt und nachfolgend systematisch gelöst. Kapitel 2 stellt vor, welche Arten von Ortsinformationen für die Aufgabenstellung von Bedeutung sind und wie diese Ortsinformationen mit geographischen Positionen verbunden werden können. Die wichtige Frage, wo auf einer Webseite Ortsinformationen vorhanden sein können, wird ebenfalls in diesem Kapitel beantwortet.

Kapitel 3 stellt den Webroboter *DCbot* vor, der im Rahmen dieser Arbeit maßgeblich erweitert wird und die geforderte Implementierungslösung darstellt. Im diesem Kapitel werden die Architektur, die Konfigurationsmöglichkeiten, die Arbeitsweise und schließlich die Ergebnisspeicherung des Webroboters vorgestellt. Dabei werden mehrere, für die Extrahierung von Ortsinformationen aus Webseiten verwendete, Methoden beschrieben.

Das vierte Kapitel präsentiert Ergebnisse aus 50.000 analysierten Webseiten, die auf zwei Testläufe verteilt untersucht wurden. Durch den Vergleich dieser Testläufe wird die Leistung des Webroboters analysiert und bewertet. Dabei werden in Webseiten gefundenen Ortsinformationen detailliert dargestellt. Sie erlauben zuverlässige Rückschlüsse auf die Gesamtverbreitung von Ortsinformationen in deutschsprachigen Webseiten.

Das letzte Kapitel fasst die Arbeit kurz zusammen und gibt Anregungen für zukünftige Verbesserungen des Webroboters.

KAPITEL 2 ORTSINFORMATIONEN IN WEBSEITEN

Dieses Kapitel stellt verschiedene Ortsinformationsarten vor und zeigt, in welchen Webseitenkomponenten diese Informationen zu finden sind.

2.1 Einführung

Webseiten im World Wide Web haben, wie gedruckte Dokumente, das Ziel, dem Leser etwas mitzuteilen. Hat diese Information einen Bezug zu einem geographischen Ort, dann sprechen wir in diesem Zusammenhang von einer Webseite mit Ortsbezug. Beispiele für entsprechende Webseiten sind die Homepage eines Restaurants oder einer Einzelperson mit zugehöriger Anschrift auf der Seite. Aber auch die Beschreibung eines Freizeitparks oder einer ganzen Stadt durch eine Webseite fallen in diese Kategorie. Dementsprechend bewegt sich die Genauigkeit der gegebenen Ortsinformationen auf einer breiten Skala. Die Hauptaufgabe dieser Arbeit ist, Webseiten mit Ortsinformationen aufzuspüren, die extrahierte Informationen in geographische Koordinaten zu überführen und die Webseiten abzuspeichern.

2.2 Ortsinformationen

Es gibt verschiedene Möglichkeiten, eine Position auf der Erde anzugeben, und viele Länder verwenden eigene, untereinander nicht kompatible Systeme. Im NEXUS Projekt und auch in dieser Arbeit wird das *World Geodetic System 1984 (WGS84)* [IGN98] für die Positionsangabe verwendet. Ein Punkt auf der Erde wird dementsprechend durch zwei Koordinaten (Breiten- und Längengrad) in Dezimaldarstellung festgelegt. So hat zum Beispiel die Stadt Stuttgart folgende Koordinaten:

Breitengrad
in Grad und Minuten: 48 47 N
in Dezimaldarstellung: 48.783

Längengrad
in Grad und Minuten: 009 11 E
in Dezimaldarstellung: 9.183

WGS84 Punkt:(48.667 9.333)

Menschen verwenden Eigennamen wenn es darum geht, einen Ort anzugeben. Im NEXUS werden Orte mit ihrer Position zum Beispiel im *WGS84* Format verknüpft. Aus diesem Grund muss an dieser Stelle untersucht werden, welche Arten von Ortsangaben im Sprachgebrauch verwendet werden, und wie diese Ortsangaben in Koordinaten überführt werden können.

2.2.1 Natürliche geographische Stätten

Mit natürlichen geographischen Stätten sind geographische Gebilde gemeint, die ohne menschlichen Beitrag entstanden sind. Beispiele dafür sind Berge, Seen oder Wüsten. Für die Suche nach Ortsinformationen in Webseiten sind in erster Linie geographische Gebilde, die im Laufe der Geschichte einen Eigennamen erhalten haben, von Interesse. Eigennamen wie „*Zugspitze*“ oder „*Bodensee*“ können auf Webseiten er-

kannt, und mit einer geographischen Position verknüpft werden. Dazu ist ein Katalog, wie zum Beispiel der *The Getty Thesaurus of Geographic Names (TGN)*, notwendig. Dieser Katalog von geographischen Eigennamen (siehe Kapitel 2.3.1 auf Seite 11) beinhaltet unter anderem für jede Stätte die genaue geographische Position.

Geographische Stätten mit großer räumlicher Ausdehnung lassen sich nur ungenau durch eine einzige Position angeben. Ein mittelgroßer Fluss kann mehrere hundert Kilometer lang und damit für die genaue Positionsbestimmung ungeeignet sein. So findet man im *TGN* für den *Nil* die Koordinaten *„30 10 N“* und *„031 06 E“*. Diese Position bezeichnet in der Tat eine Stelle des *Nil* in der Nähe von *Kairo*, der Hauptstadt von Ägypten. Angesicht der Tatsache, dass der *Nil* 6671 Kilometer lang ist und durch den halben Kontinent *Afrika* fließt ist diese Positionsangabe doch sehr ungenau.

Allgemein lässt sich dieses Problem lösen, wenn geographische Stätten nicht durch Punkte, sondern durch Linien oder Polygone repräsentiert werden. Eine alternative Möglichkeit ist die Beibehaltung der Repräsentation einer geographischen Stätte durch einen Punkt und die zusätzliche Angabe der Ausdehnung dieses Objektes. Diese vereinfachte Darstellung entspricht der Modellierung durch einen Kreis (Punkt = Mittelpunkt, Ausdehnung = Kreisfläche). Die Modellierung einer geographischen Stätte durch ein Polygon oder einen Kreis werden am Beispiel des Bodensees in Abbildung 2-1 veranschaulicht.

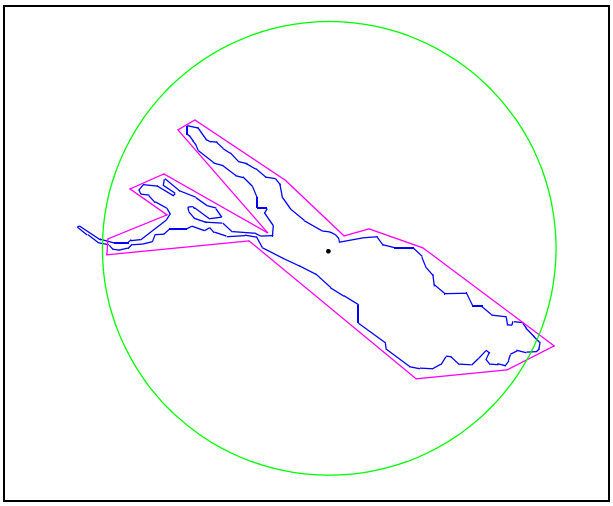

Abbildung 2-1: Modellierung einer geographischen Stätte

Für diese Arbeit wird die vereinfachte Modellierung von geographischen Stätten durch die Kreisdarstellung gewählt. Jede Stätte wird durch ihren Mittelpunkt, also durch eine geographische Position, und durch ihre Ausdehnung (Flächeninhalt) dargestellt.

2.2.2 Bauwerke

Ein offensichtlicher Beweis menschlicher Kreativität sind die zahlreichen Bauwerke, die die Menschheit seit Anfang ihrer Existenz errichtet hat. Ob in Form von Gebäuden, Denkmäler oder Brücken, sie sind überall im täglichen Leben anzutreffen. Die

wichtigsten dieser Bauwerke haben, wie natürlich entstandene geographische Stätten, ihre Eigennamen. Prominente Beispiele sind der „*Eiffelturm*“, die „*Golden Gate Bridge*“ oder der „*Kölner Dom*“. Solche Eigennamen können auf Webseiten erkannt, und mit Hilfe einer Datenbank in geographische Positionen überführt werden.

Bauwerke können, wie natürlich entstandene geographische Stätten, durch Linien oder Polygone dargestellt werden. Dies erweist sich in der Regel als einfach, da die Grundrisse der meisten Bauwerke durch einfache Polygone modelliert werden können. Abbildung 2-2 zeigt einfache Polygonmodelle der oben erwähnten Bauwerke.

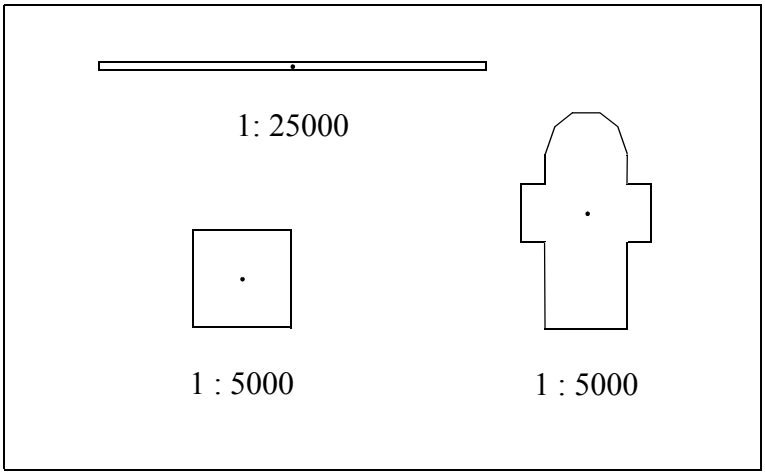

Abbildung 2-2: Grundrisse von Eiffelturm, Golden Gate Bridge und Kölner Dom

Verglichen mit natürlich entstandenen geographischen Stätten haben Bauwerke relativ kleine Grundflächen. Sie werden trotzdem durch die gleiche Modellierung (durch Mittelpunkt und Flächeninhalt), wie natürliche geographische Stätten, dargestellt.

2.2.3 Politische Gebietsaufteilungen

Gebiete, die im Laufe der Zeit aus politischen oder aus gesellschaftlichen Gründen Eigennamen erhalten haben, können ebenfalls zur Positionsbestimmung herangezogen werden. Kontinente sind in Staaten, Staaten wiederum oft in weitere administrative Teilgebiete wie zum Beispiel Bundesländer unterteilt. In vielen Ländern gibt es weitere Stufen dieser hierarchischen Unterteilung. So werden zum Beispiel in Deutschland Dörfer zu Gemeinden, Gemeinden und Städte wiederum zu Kreisen zusammengefasst.

Eigennamen wie „*Stuttgart*“ oder „*Hohenlohekreis*“ können auf Webseiten erkannt und mit Positionen verknüpft werden. Dabei ist der hierarchiebedingte Größenunterschied dieser Gebiete zu beachten. Bundesländer und Kreise umfassen in der Regel größere Gebiete, und können deswegen nur ungenaue Positionen liefern, während Städte und Gemeinden eine feinere Auflösung darstellen.

Eigennamen administrativer Gebiete können in den gleichen Datenbanktabellen wie natürlich entstandene geographische Stätten und Bauwerke gespeichert werden. Die Beibehaltung der bestehenden hierarchischen Struktur ist sinnvoll. Die Modellierung

durch den Mittelpunkt und den Flächeninhalt wird, wie für alle geographischen Objekte in dieser Arbeit, beibehalten.

2.2.4 Administrative Gebietsaufteilungen

Um die nationale und internationale Kommunikation beziehungsweise den Postverkehr zu unterstützen, haben die meisten Länder zusätzliche künstliche Gebietsaufteilungen eingeführt. Für den Postverkehr gibt es Postleitzahlgebiete und die Hausnummerierung, für die Telefonie Vorwahlnummern.

In Deutschland werden seit 1944 (Quelle: *wissen.de*-Lexikon) Postleitzahlen für den Postverkehr verwendet. Am 1. Juni 1993 wurde das alte, vierstellige System durch das fünfstellige Postleitzahlensystem ersetzt. Zurzeit gibt es in Deutschland 8.265 (Quelle: *Deutsche Post AG*) verschiedene Postleitzahlgebiete, die im Durchschnitt eine Fläche von 43,2 Quadratkilometer umfassen. Da die Größe der Postleitzahlgebiete sich nach der Bevölkerungsdichte richtet, ist die Durchschnittsfläche, die einer Postleitzahl in Großstädten zugeordnet werden kann, deutlich kleiner. Am Beispiel von Stuttgart kann dies veranschaulicht werden. Die Landeshauptstadt umfasst eine Fläche von 207,34 Quadratkilometern und besitzt 34 größere Postleitzahlgebiete, was eine Durchschnittsfläche von 6,1 Quadratkilometer pro Postleitzahl ergibt.

Im World Wide Web gibt es eine Vielzahl von Webseiten, die eine Privatperson, eine Firma oder Geschäfte mit verschiedenen Dienstleistungen vorstellen. Diese Seiten bieten oft Kontaktinformationen für potenzielle Interessenten an. Wird während der Suche im World Wide Web eine Kontaktadresse mit zugehöriger Adresse erkannt, kann diese Seite mit einer geographischen Position verknüpft werden. Dazu wird eine Datenbank, die Adressen mit Positionen verbindet (siehe Kapitel 2.3.2 auf Seite 14), benötigt.

Analog können Kontaktinformationen nach Telefonnummern durchsucht werden. Die Vorwahlnummer steht für ein bestimmtes Gebiet, dessen geographische Position mit Hilfe einer entsprechenden Datenbank bestimmt werden kann.

2.3 Geodaten

In der Studienarbeit des Autors [Süt01] wurden unter anderem spezielle Meta-Tags zur Angabe von genauen Ortsinformationen untersucht. Eine der Erkenntnisse aus dieser Arbeit war die Tatsache, dass Webseiten in den seltensten Fällen genaue Positionsangaben (zum Beispiel geographische Koordinaten), die den Webseiteninhalt mit einem geographischen Ort sinnvoll verbinden, enthalten. Aus diesem Grund müssen auf Webseiten vorhandene Ortsinformationen (siehe Kapitel 2.2 auf Seite 7) analysiert und in entsprechende Koordinaten überführt werden.

Dazu wird eine Geodatenbank, die Ortsinformationen mit geographischen Positionen verbindet, benötigt. Zu diesem Zweck wird eine eigene Geodatenbank entworfen, die aus verschiedenen Quellen mit Daten gefüllt werden kann. Der Webroboter *DCbot*,

der nach den nötigen Modifikationen die Sammlung und Analyse von Webseiten übernehmen wird, kann dann auf diese Datenbank zugreifen.

2.3.1 The Getty Thesaurus of Geographic Names

Der *Getty Thesaurus of Geographic Names (TGN)* ist eine hierarchische Datenbasis von über eine Million Eigennamen geographischer Orte. Neben der hierarchischen Einordnung in die Gesamtdatenbasis enthält sie für jeden Eintrag den Ortstyp, die geographischen Koordinaten und eventuell weitere relevante Informationen. Die für diese Arbeit entworfene Geodatenbank *GeoBase* (siehe Kapitel 2.3.3 auf Seite 15) orientiert sich stark an der hierarchischen Struktur des *TGN*. So ist es möglich, Daten aus dem *TGN* direkt in *GeoBase* zu importieren.

2.3.1.1 Beispieldatensatz

Es besteht die Möglichkeit, eine Anfrage an den *TGN* über ein Webinterface [Get00] zu stellen. Für den Ortsnamen *„Stuttgart“* enthält der Thesaurus unter anderem die folgenden Daten (der Gesamteintrag ist in Kapitel B.1 auf Seite 91 zu finden):

Stuttgart (inhabited place)

Lat: 48 47 N Long: 009 12 E (represented in degrees minutes direction)
Lat: 48.783 Long: 9.200 (represented in decimal degress and fractions of degrees)

Note - Located on Neckar river, in forested area of vineyards & orchards; town developed around a medieval fortified manor & stud farm; went to House of Württemberg in 13th cen.; declined during Thirty Years' War in 17th cen.; nearly destroyed in WW II.

Hierarchical Position:

Europe.....(continent)
Deutschland.....(nation)
Baden-Württemberg.....(state)
Stuttgart.....(district (national))

Auf diese Weise erhält man die genaue geographische Position für Stuttgart (Breite *„48° 47′ N“* und Länge *„009° 12′ E“*), was im Zielkontext dieser Arbeit die wichtigste Information darstellt. Wird während einer eventuellen Suche im World Wide Web ein Ortsname auf einer Webseite als Ortsinformation erkannt, dann könnte man in diesem Thesaurus die entsprechende Position nachschlagen. Während die kurze Notiz über die Stadt eher von geringem Interesse ist, könnte die hierarchische Einordnung zur Unterscheidung von mehrfach vergebenen Ortsnamen dienen. Den Ortsnamen *„Dresden“* gibt es nur einmal in Deutschland, aber 14 Mal in den USA.

2.3.1.2 Schnittstellen

Die bereits erwähnte Webschnittstelle ist an Benutzer gerichtet, die wenige Einträge im *TGN* nachschlagen möchten. Um den schnellen Zugriff auf die Datensätze und damit die effiziente Arbeit mit der Datenbasis zu ermöglichen, kann man den gesamten Thesaurus in drei verschiedenen Formaten erwerben.

Für diese Arbeit ist die „*Relational Files Format*"-Distribution des *TGN* [Har01] von besonderem Interesse, da diese aus einer Sammlung von ASCII-Dateien besteht, die Tabellen einer relationalen Datenbank entsprechen. Die Datensätze aus den ASCII-Dateien können direkt in ein beliebiges Datenbanksystem importiert werden. Die acht Tabellen stellen vier Entities und fünf Relationen dar, die in der Abbildung 2-3 zu einem Gesamtbild zusammengefügt werden.

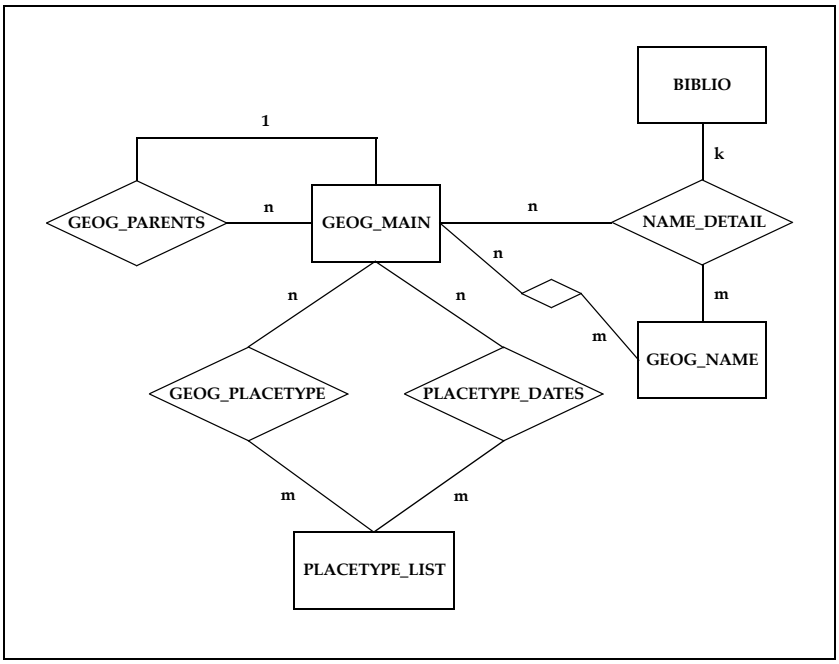

Abbildung 2-3: Entity-Relationship Modell des *TGN*

Nicht alle Daten der Distribution werden im Rahmen dieser Arbeit benötigt. Mehrfach auftretende Beziehungen (Redundanz) sind für diese Arbeit ebenfalls von geringer Bedeutung. Welche Tabellen oder Teiltabellen (Attribute) in *GeoBase* übernommen werden sollen, bedarf einer genaueren Untersuchung. Aus diesem Grund werden nachfolgend alle Tabellen kurz vorgestellt.

GEOG_MAIN

Dies ist die Haupttabelle, deren Einträge einzelne geographische Orte repräsentieren. Zu jedem Ort werden die entsprechenden geographischen Koordinaten und eine kurze Beschreibung angegeben.

Primärschlüssel: *geog_key*

GEOG_NAME

Tabelle von Ortsnamen. Jeder Ortsname (Attribut: *place_name*) wird mit einer eindeutigen Identifikationsnummer (Attribut: *placename_id)* versehen und einer oder mehreren Orten Attribut: *geog_key*) zugewiesen. Weitere Attribute erlauben die Speicherung von zusätzlichen Informationen, die zum Beispiel angeben, ob der betreffende Ortsname ein aktueller oder ein historischer Name ist.

Primärschlüssel: *geog_key+placename_id*

Fremdschlüssel: *geog_key*

PLACETYPE_LIST

In dieser Tabelle werden Ortstypen (Attribut: *placetype*) gespeichert. Beispiele sind *„capital“*, *„island“* oder *„ruins“*. Jeder Ortstyp hat eine eindeutige Identifikationsnummer *placetype_code*.

Primärschlüssel: *placetype_code*

BIBLIO

Diese Tabelle listet alle Quellen (Attribut: *citation*) auf, aus denen die Ortsnamen stammen. Beispiele sind der *„Times Atlas of World History“* oder der *„Britannica Online“*.

Primärschlüssel: *bib_key*

GEOG_PARENTS

In dieser Tabelle werden die hierarchischen Beziehungen zwischen allen geographischen Orten gespeichert. Zu jedem Ort (Attribut: *geog_key*) wird der übergeordnete Ort (Attribut: *parent_key*) in der Hierarchie angegeben.

Primärschlüssel: *geog_key*

Fremdschlüssel: *geog_key, parent_key*

NAME_DETAIL

Diese Tabelle speichert zusätzliche Informationen über Ortsnamen. Sie gibt unter anderem die Quelle (Attribut: *bib_key*) an, aus der der Ortsname stammt sowie einen Zeitraum (Attribut: *display_date*), innerhalb dessen der angegebene Ortsname Gültigkeit besitzt. Die Zuordnung von Orten zu Ortsnamen, die bereits in der Tabelle *GEOG_NAME* gespeichert sind, wird in dieser Relation wiederholt.

Primärschlüssel: *geog_key+placename_id+bib_key*

Fremdschlüssel: *geog_key, placename_id, bib_key*

GEOG_PLACETYPE

Diese Relation ordnet jedem Ort (Attribut: *geog_key*) einen Ortstyp (Attribut: *placetype_code*) aus der Tabelle *PLACETYPE_LIST* zu. Ähnlich wie bei Ortsnamen werden hier weitere Informationen, wie zum Beispiel die Aktualität des Ortstyps (Attribut: *age_flag*), gespeichert. Eine Stadt kann zum Beispiel den aktuellen Ortstyp „city“

und den, vor hundert Jahren verwendeten, jetzt nicht mehr aktuellen, Ortstyp „village“ besitzen.

Primärschlüssel: *geog_key+placetype_code*

Fremdschlüssel: *geog_key, placetype_code*

PLACETYPE_DATES

In dieser Tabelle wird die Zuordnung von Ort (Attribut: *geog_key*) zu Ortstyp (Attribut: *placetype_code*) aus der Relation *GEOG_PLACETYPE* wiederholt. Zusätzlich wird für jede Zuordnung ein Zeitraum (Attribut: display_date) angegeben, innerhalb dessen der angegebene Ortstyp aktuell war.

Primärschlüssel: *geog_key+placetype_code+display_date*

Fremdschlüssel: *geog_key, placetype_code*

Bei näherer Betrachtung der Tabellen wird klar, dass viele der gespeicherten Informationen für diese Arbeit von geringer Bedeutung sind. So wird zum Beispiel die Quelle eines Ortsnamen für die Positionsbestimmung nicht benötigt. Welche Tabellen beziehungsweise Attribute bei dem Entwurf von *GeoBase* Beachtung finden, wird in Kapitel 2.3.3 auf Seite 15 dargelegt.

2.3.2 Datafactory Geocode

Die im vorangehenden Kapitel vorgestellte Datenbasis enthält Positionsdaten für natürlich entstandene geographische Stätten sowie für politische Gebiete. Um eine möglichst vollständige Positionsdatenbank zu erhalten, sollte man in diese auch Positionsdaten administrativer Gebietsaufteilungen integrieren.

Die Deutsche Post vertreibt eine Positionsdatenbasis, deren künstliche Gebietsaufteilung auf postalische Adressen in Deutschland beruht. Die Datenbank *Datafactory Geocode* liefert zu einer Adresse die entsprechenden geographischen Koordinaten. Die Datenbank ist in vier verschiedenen Auflösungsstufen erhältlich:

- **Postleitzahlgebiete**

  Dies ist die grobmaschigste Einteilung. An Hand der Postleitzahl wird die Adresse zu einer der 8.265 Postleitzahlgebieten zugeordnet.

- **Mikrom-Marktzelle**

  Jeweils 400 Haushalte werden zu 85.000 künstlichen Marktzellen zusammengefasst. Jede postalische Adresse gehört dann zu einer dieser Marktzellen.

- **Straßenebene**

  Die Adressen werden einer der 1,2 Millionen Straßen beziehungsweise Straßenabschnitten (bei längeren Straßen) zugeordnet.

- **Gebäudeebene**

  In der feinsten Auflösung liefert die Datenbank das zur Adresse gehörende Gebäude. Die Datenbasis enthält 3,5 Millionen Einträge.

In allen Auflösungsstufen enthält die Datenbank für jeden Eintrag die genauen geographischen Koordinaten.

2.3.3 GeoBase

Das Implementierungsziel dieser Arbeit ist die Erweiterung des Webroboters *DCbot* (siehe Kapitel 3 auf Seite 33), damit er in der Lage ist, eigenständig Webseiten mit Ortsbezug im World Wide Web aufzuspüren. Zu diesem Zweck sollte der Webroboter auf eine Datenbasis mit geographischen Positionen zugreifen können. Im Kapitel 2.2 auf Seite 7 wurden verschiedenen Ortstypen dargestellt, die auf Webseiten erkannt und mit geographischen Positionen verknüpft werden können. Die hier entworfene Datenbasis *GeoBase* kann all diese Typen aufnehmen. Dementsprechend kann sie aus verschiedenen Quellen (wie dem *TGN* oder der *Datafactory Geocode*) Datensätze importieren.

2.3.3.1 Hierarchische Gliederung

In *GeoBase* werden geographische Objekte gespeichert. Es liegt nahe, diese Objekte in eine geographische Topologie einzuordnen. Dies kann vor allem nützlich sein, wenn zwischen mehreren Objekten mit dem gleichen Namen unterschieden werden muss.

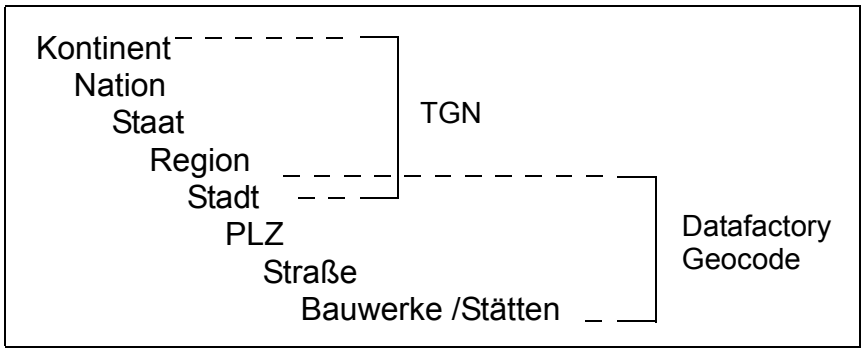

Abbildung 2-4: Geographische Topologie von *GeoBase*

Die in dieser Abbildung gezeigte hierarchische Ordnung vereinigt die Struktur des *TGN* und der *Datafactory Geocode*. Werden Einträge aus beiden Datenbanken übernommen, müssen diese an der Stadt-Ebene zusammengefügt werden. Ein Objekt in *GeoBase* muss nicht zwingender Weise alle Hierarchiestufen als Vorgänger besitzen. Wenn ein Bauwerk zum Beispiel zu groß ist, um einer Straße oder einem Postleitzahlbereich zugeordnet zu werden, dann kann es ohne Probleme eine Stadt als Vorgänger in der Hierarchie haben. Analog können weitere Stufen in die Hierarchie aufgenommen werden. Die einzige Restriktion ist, dass jedes Objekt einen Vorgänger haben muss.

2.3.3.2 Tabellen

Das Datenmodell von *GeoBase* orientiert sich stark am Modell des *TGN*. Die drei Entities *PLACES*, *NAMES* und *PLACETYPES* werden durch drei Relationen miteinander verknüpft. Die folgende Abbildung zeigt das Entity-Relationship Diagramm von *GeoBase*.

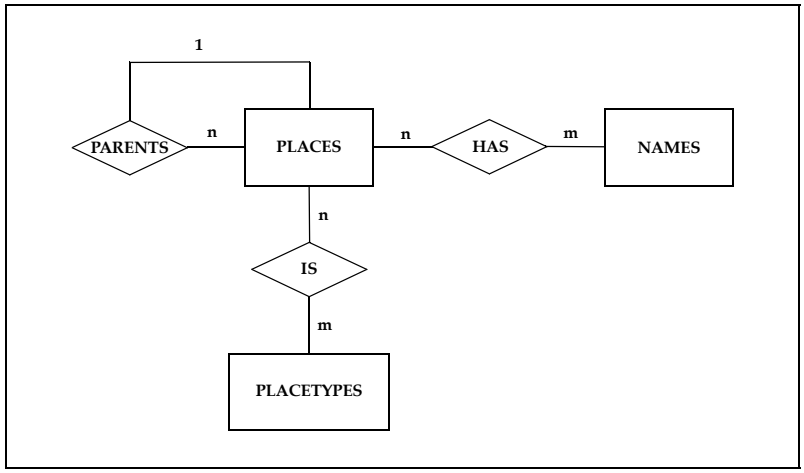

Abbildung 2-5: Entity-Relationship Modell von *GeoBase*

GeoBase besteht aus drei Tabellen, die nachfolgend vorgestellt werden:

PLACES

Dies ist die Haupttabelle, die alle geographischen Orte (Primärschlüssel: *place_id*) mit den zugehörigen Koordinaten in Dezimaldarstellung (Attribute: *latitude, longitude*) enthält. Die geographische Topologie von Orten wird ebenfalls hier gespeichert. Das Attribut *parent* gibt für jeden Datensatz den direkten Vorgänger (*place_id*) in der Hierarchie an. Da jeder Ort nur einen Vorgänger aber mehrere Nachfolger haben kann, ist dies eine 1:n-Beziehung (Relation: *PARENTS*). Das Attribut *area* gibt die Fläche des jeweiligen Ortes in Quadratkilometern an. Die nachfolgende Tabelle zeigt drei Beispieldatensätze.

place_id	latitude	longitude	parent	area
7000084	51.500	10.500	1000003	356970
7003692	47.950	9.900	7000084	35752
7004425	48.783	9.200	7003692	207,34

Tabelle 2-1: *GeoBase*-Tabelle PLACES

Der erste Eintrag (*place_id*: 7000084) repräsentiert *Deutschland* mit dem Vorgänger *Europa*, der dritte steht für *Stuttgart* mit dem Vorgänger *Baden-Württemberg*.

NAMES

Diese Tabelle enthält Namen (Attribut: *name*) von geographischen Orten. Jedem Ort können mehrere Namen zugeordnet werden, zugleich existieren verschiedene Orte

mit dem gleichen Namen. Aus diesem Grund wird hier über das Attribut *place_id* eine n:m-Beziehung zum Entity *PLACES* aufgebaut.

name_id	name	place_id
110	Deutschland	7000084
3857	Baden-Württemberg	7003692
4913	Stuttgart	7004425

Tabelle 2-2: *GeoBase*-Tabelle NAMES

Das Attribut *name_id* ist für jeden Namen eindeutig. Die Attribute *name_id* und *place_id* bilden den zusammengesetzten Primärschlüssel.

PLACETYPES

Diese Tabelle listet Ortstypen (Attribut: *placetype*) auf, denen Orte zugeordnet werden können. Ähnlich wie bei Ortsnamen kann ein Ort mehrere Typen haben, außerdem gibt es sehr viele Orte gleichen Typs. Wieder wird eine n:m-Beziehung über das Attribut *place_id* zum Entity *PLACES* aufgebaut. Das Attribut *placetype_id* ist für jeden Ortstyp eindeutig. Der Primärschlüssel setzt sich aus den beiden Attributen *placetype_id* und *place_id* zusammen.

placetype_id	placetype	place_id
81010	nation	7000084
82135	state	7003692
83040	city	7004425
83371	industrial center	7004425

Tabelle 2-3: *GeoBase*-Tabelle PLACETYPES

Wie die beiden letzten Datensätze demonstrieren, kann ein Ort (hier *Stuttgart*) mehrere Ortstypen haben.

Streng genommen bräuchte man, um Ortsnamen mit Positionen zu verbinden, nur eine Tabelle, die mit wenigen Attributen (*place_id, name*, *latitude*, *longitude*) auskommen könnte. Die hierarchische Ordnung und Ortstypen wurden zusätzlich aufgenommen, damit zwischen Orten mit gleichen Namen unterschieden werden kann. Die Datenbasis ist auf drei Tabellen aufgeteilt, damit unnötige Redundanz vermieden wird.

2.3.3.3 Importieren von Daten

Wie bereits erwähnt, soll *GeoBase* aus verschiedenen Quellen Daten importieren. Da *GeoBase* sich streng an der Struktur des *TGN* orientiert, gestaltet sich die Übernahme von Datensätzen aus diesem Thesaurus besonders einfach.

Die Tabelle *PLACES* ergibt sich aus einem *Join* der *TGN*-Tabellen *GEOG_MAIN* und *GEOG_PARENTS*:

```
SELECT  a.geog_key  AS  place_id,
        a.lat_num   AS  latitude,
        a.long_num  AS  longitude,
        b.parent_key AS  parent
FROM    geog_main a, geog_parents b
WHERE   a.geog_key = b.geog_key;
```

Das Attribut *area* der Tabelle *PLACES* von *GeoBase* erfordert ein wenig zusätzliche Arbeit. Eine ähnliche Angabe ist im *TGN* nicht vorhanden, ist aber für den Webroboter *DCbot* von besonderer Bedeutung. Durch dieses Attribut wird die Angabe eines Schwellwertes möglich, die die Genauigkeit von gesuchten Ortsinformationen festlegt. Das Attribut *area* kann näherungsweise aus dem Typ eines Ortes abgeleitet werden. Ist eine höhere Genauigkeit erwünscht, so muss dieser Wert aus einer entsprechenden Flächengrößen-Datenbank importiert werden.

Die Tabelle *NAMES* ist eine einfache *Projektion* der *TGN*-Tabelle *GEOG_NAME*:

```
SELECT  a.placename_id AS  name_id,
        a.place_name   AS  name,
        a.geog_key     AS  place_id
FROM    geog_name a;
```

Während die Tabelle *PLACETYPES* wieder ein *Join* der *TGN*-Tabellen *GEOG_PLACETYPE* und *PLACETYPE_LIST* ist:

```
SELECT  a.placetype_code          AS  placetype_id,
        a.placetype, b.geog_key   AS  place_id
FROM    placetype_list a, geog_placetype b
WHERE   a.placetype_code = b.placetype_code;
```

Eine kurze Beschreibung der hier verwendeten *TGN*-Tabellen ist in Kapitel 2.3.1.2 auf Seite 12 zu finden, während in Kapitel B.2 auf Seite 92 alle Attribute der betroffenen Tabellen mit Beispieldatensätzen erläutert werden.

2.4 Ortsinformationen in Webseitenkomponenten

Als erstes muss die Frage geklärt werden: Wo sind auf einer Webseite Ortsinformationen zu finden? Aus diesem Grund setzen sich die folgenden Unterkapitel mit einzelnen Komponenten einer Webseite, und mit ihrem möglichen Informationsgehalt, auseinander.

Bei der Ermittlung von Ortsinformationen aus Webseiten sind zwei grundlegende Vorgehensweisen möglich:

- Die Analyse von Webseiten unter Einbeziehung aller Webseitenkomponenten, und die Extrahierung von Kandidaten, die möglicherweise Ortsinformationen enthalten. Solche Kandidaten können Ortsnamen, wie zum Beispiel *„Stuttgart“*, sein. Anschließend müssen diese durch einen Vergleich mit einem Katalog verifiziert werden. Optimalerweise enthält der Katalog gültige Ortsnamen und die dazugehörige Positionen in geeigneter Darstellung (z.B. *WGS 84*).
- Die zweite Möglichkeit besteht in der Ermittlung einer „sicheren“ Ortsinformation aus den Webseitenkomponenten. Eine solche Ortsinformation könnte zum Beispiel eine komplette Adresse sein, die aus dem Webseiteninhalt extrahiert wird. Oder die Telefonnummer des Domaininhabers, die aus einer *Whois*-Datenbank für Domains herausgelesen wird. Anschließend kann man die zum Ortsnamen oder zur Telefonnummer gehörende Position aus einer Datenbank herauslesen.

Der Hauptunterschied zwischen den beiden Ansätzen ist der Aufwand, der für die Extrahierung eines Ortsbezugs aufgebracht werden muss. Der erste Ansatz ist nur sinnvoll, wenn die Anzahl der Kandidaten überschaubar bleibt. Der Vergleich aller Wörter auf einer Webseite mit dem Katalog würde einen unangemessenen Aufwand erzeugen. Ebenso ist der zweite Ansatz für Webseitenkomponenten mit vergleichsweise wenigen Wörter, wie der Titel oder der Text von Hyperlinks, nicht sinnvoll. Es ist unwahrscheinlich, dass diese Komponenten, beispielsweise eine komplette Adresse enthalten.

Bei der Untersuchung der Webseitenkomponenten werden aus diesem Grund beide Ansätze abwechselnd, beziehungsweise geeignet kombiniert, eingesetzt.

2.4.1 Die URL

Durch eine *Uniform Resource Locator* [DMM94] kann eine eindeutige Adresse für jede Ressource im Internet vergeben werden. Um Webseiten im World Wide Web zu adressieren, benutzt man das *http*-Schema (*HyperText Transfer Protocol*). Damit schreibt sich eine URL für Webseiten vereinfacht:

```
http://<host><path>
```

Durch den <host> wird ein Web-Server identifiziert, der die jeweilige Webseite anbietet. Es besteht die Möglichkeit, für <host> eine Domain oder eine IP-Adresse anzugeben. Da eine Domain, wie zum Beispiel *„stuttgart.de“*, einfacher zu merken ist, als die zugehörige IP-Adresse „194.97.218.100“, werden Domains in der Regel bevorzugt.

Der <path> entspricht der Verzeichnisstruktur auf dem Web-Server und gibt damit an, in welchem Verzeichnis eine gegebene Webseite zu suchen ist.

Ob über die Domain oder die Verzeichnisnamen ein Ortsbezug hergestellt werden kann, wird in den folgenden Kapiteln genauer untersucht.

2.4.1.1 Der Host

Der *Host* besteht aus einer Domain und eventuell aus weiteren *Labels*. Eine Domain setzt sich aus zwei Hauptkomponenten zusammen: aus der *Top-Level Domain (TLD)*, wie zum Beispiel *„de“*, *„com“* oder *„org“* und aus der *Second-Level Domain*, wie zum Beispiel *„stuttgart“* (für den oben genannten Beispiel *„stuttgart.de“*). Die allgemeine Schreibweise ist damit:

```
SecondLevelDomain.TopLevelDomain
```

Eine Domain kann durch weitere *Labels* unterteilt, und auf mehrere *Hosts* verteilt werden. Der Webserver der oben genannten Domain kann also *„www.stuttgart.de“* heißen, während der Mail-Server mit *„mail.stuttgart.de“* bezeichnet werden kann. Die *Labels „www“* und *„mail“* nennt man auch *Third-Level Domain*. Die allgemeine Schreibweise für einen *Host* mit einem *Label* ist damit:

```
ThirdLevelDomain.SecondLevelDomain.TopLevelDomain
```

Die meisten *TLDs* signalisieren die Länderzugehörigkeit der darunter erreichbaren Server. So haben alle Länder eine eigene *TLD*, wie zum Beispiel Deutschland mit *„de“*, Italien mit *„it“* oder Ungarn mit *„hu“*. Diese so genannte *country coded Top-Level Domains (ccTLD)* entsprechen der Zweibuchstaben-Codierung der Ländernamen, wie sie im *ISO3166 Standard* [IOS02] festgelegt sind. Zusätzlich gibt es weitere *TLDs*, die Server einer bestimmten Kategorie oder Interessengemeinschaft zusammenfassen. Solche *TLDs* sind *„com“* für kommerzielle Angebote, *„org“* für nicht-kommerzielle Organisationen oder *„edu“* für Angebote im Bereich der Forschung und Lehre.

Es gibt drei Möglichkeiten, einen Ortsbezug über eine Domain beziehungsweise über einen Hostnamen herzustellen:

1. Ist die *TLD* eine *ccTLD,* so ist der Webseiteninhalt oft in der jeweiligen Landessprache und/oder hat einen bestimmten Bezug zu diesem Land. So werden Webseiten mit der *TLD „de“* in der Regel in deutscher Sprache verfasst. Da dieser Ortsbezug nicht gewährleistet werden kann und er ohnehin sehr ungenau ist, was die ermittelbare geographische Position betrifft, wird diese Möglichkeit nur der Vollständigkeit halber erwähnt.

2. Bei weitem mehr Information erhält man durch die Analyse der *Second-Level Domain* oder der weiteren *Label*. Als erstes können diese direkt auf Ortsnamen untersucht werden. So gibt es Domains wie *„zugspitze.de“* oder eben *„staatstheater.stuttgart.de“*, die Informationen über die entsprechenden Orte bereitstellen. Dabei muss man beachten, dass die *Second-Level Domains* oder die weiteren *Label* zusammengesetzte Ausdrücke enthalten können. Beispiele dafür wären *„uni-stuttgart.de“* oder *„partys-in-stuttgart.de“*. Domains sind allerdings eher selten so zutreffend benannt. Nur wenige Webseitenbetreiber haben das Privileg, eine, zum Webseiteninhalt passende Domain, zu besitzen.

3. Eine weitere Möglichkeit ist die Anfrage bei einem Domain-Informationsdienst. Der Betrieb der *TLD „de“* wird in Deutschland von der DENIC eG [Den02] durch-

geführt. Sie ist eine eingetragene Genossenschaft und repräsentiert 183 (Stand: Februar 2003) *Internet Service Provider (ISP)* aus Deutschland. Sie ist für die Verwaltung und die Vergabe aller *Second-Level Domains* unter der *TLD „de“* zuständig. Neben Statistiken zur Entwicklung der Domainanzahl bietet die DENIC eG auch umfangreiche Informationen über einzelne Domains. So kann man mit Hilfe der *Whois*-Datenbank der DENIC eG für jede Domain den *Domaininhaber*, den *administrativen Ansprechpartner* und einen *technischen Ansprechpartner* ermitteln. Für die Domain „*stuttgart.de*“ erhält man unter anderem folgende Informationen (für den Gesamteintrag siehe Kapitel A.1 auf Seite 85), die den Grundstoff für weitere Analyseansätze liefern:

Domaininhaber:

Landeshauptstadt Stuttgart, Haupt- und Personalamt
Abt. Informations- und Kommunikationstechnik
Eberhardstr. 6
D-70173 Stuttgart
Germany
DE

Der Inhaber einer Domain ist in der Regel eine Firma, eine Organisation oder eine Privatperson, die mit Namen und Adresse angegeben sind. Während der Namenseintrag eher selten eine Ortsinformation enthält (*„Landeshauptstadt Stuttgart“* als Inhaber stellt in diesem Fall eine Ausnahme dar), ist die Adresse des Inhabers eine präzise Ortsinformation.

Administrativer Ansprechpartner:

Name:	Roland Schaefer
Kontakttyp:	Person
Adresse:	Landeshauptstadt Stuttgart
	Eberhardstr. 6
Stadt:	Stuttgart
PLZ:	70173
Land:	GERMANY

Ähnlich ist dies bei den Angaben über den administrativen Ansprechpartner. Sie enthalten immer eine Kontaktadresse, die wie beim Domaininhaber eine genaue Ortsinformation darstellt. Die Frage ist nur, wie weit sich der Inhalt einer Webseite zur Adresse des Domaininhabers oder des administrativen Ansprechpartners zuordnen lässt. Während bei einigen Domains, wie „*stuttgart.de*“ oder „*zugspitze.de*“ (für Details siehe Anhang A auf Seite 83), eine enge Verbindung besteht (Adressen jeweils aus Stuttgart und aus Garmisch-Partenkirchen), haben die Inhalte von Webseiten bei großen *Internet Service Provider* keinen Bezug zur Adresse des Providers. Der größte *ISP* in Deutschland, T-Online, bietet mehreren Millionen Kunden die Möglichkeit, Webseiten ins Internet zu stellen; alle diese Seiten sind unter der gemeinsamen *Second-Level Domain t-online.de* erreichbar.

Technischer Ansprechpartner:

Name: Tiscali Hostmaster Role-Account
Kontakttyp: Role
Adresse: Robert-Bosch-Str. 32
Stadt: Dreieich
PLZ: 63303
Land: GERMANY
...

Jede Domain hat eine zugehörige IP-Adresse, unter der der betroffene Web-Server im Internet erreichbar ist. Der technische Ansprechpartner ist für den IP-Adressenbereich verantwortlich, in den diese IP-Adresse fällt. Die Adresse des technischen Ansprechpartners lässt vermuten, wo der entsprechende Web-Server steht. Dieser Ansatz wurde unter anderem in [BCG99] diskutiert. Die Verfasser stellten ein Verfahren vor, das den Webserver einer Webseite auf diesem Wege geographischer Positionen zuordnet. Der Inhalt der jeweiligen Webseite wurde nicht untersucht.

Gerade in dem zuletzt erwähntem Bereich gibt es mehrere Ansätze (siehe [Rös02]), die sich mit der Zuordnung von IP-Adressen zu geographischen Positionen beschäftigen. Hierbei stehen allerdings eher Webseitenbesucher, die ebenfalls durch eine IP-Adresse im Internet präsent sind, im Vordergrund. Durch die Zuordnung von Benutzer zu geographischen Orten, werden qualitativ höherwertige Webangebote möglich. Informationen und Werbeangebote können dann aus der Zielregion geliefert werden.

Aus diesem Grund gibt es Vorschläge für die Erweiterung des *Domain Name System* um ein neues *Resource Record* (siehe [Moc87)] namens *LOC*. Der *DNS-LOC* [DVG96] soll für jede im *DNS*-Server gespeicherte Domain, geographische Informationen enthalten. Dadurch wären komplizierte Berechnungen, um Positionsdaten zu Domains beziehungsweise zu Webservern zu erhalten, überflüssig. Eine kurze Anfrage beim nächsten *DNS*-Server wäre ausreichend.

Leider sagt die Position des Webservers auf dem sich eine Webseite befindet, nur wenig über den Inhalt dieser Seite aus. So befinden sich die Webseiten der Domain *„zugspitze.de“*, wenn man die Adresse des technischen Ansprechpartners aus der DENIC *Whois*-Datenbank betrachtet, auf einem Web-Server in München. Die Zugspitze, der höchste Berg Deutschlands, liegt allerdings bei Garmisch-Partenkirchen, das gut 80 Kilometer von München entfernt ist. Bei diesem Beispiel wäre die Adresse des administrativen Ansprechpartners oder des Domaininhabers zutreffender gewesen.

2.4.1.2 Der Path

Wie vorhin bereits erwähnt hat eine URL für Webseiten die Form http://<host><path>. Der *Path* entspricht der Verzeichnisstruktur auf dem entsprechenden Web-Server und hat die allgemeine Form:

/Verzeichnis_1/Verzeichnis_2/ ... /Verzeichnis_n

Mit Hilfe der Verzeichnisebenen kann eine *Website* sinnvoll strukturiert werden. Die Gliederung kann nach Themengebieten erfolgen, wie dies oft bei Online-Katalogen der Fall ist. Der *Path*

```
/sport/basketball/regeln/
```

auf dem *Host „de.dir.yahoo.com“* gibt offensichtlich ein Verzeichnis an, in dem Webseiten die Regeln des Basketballspielens beschreiben. Der Aufbau einer *Website* kann aber auch einer geographischen Gliederung folgen, wie bei dem folgenden Beispiel

```
/html/europa/deutschland/ flughaefen/stuttgart/
```

unter der Domain *„flugplandaten.de“*. Möchte also ein Webseitenbetreiber bereits in der URL dem Besucher eine Ortsinformation mitteilen, kann er dies trotz einer wenig aussagekräftigen Domain durch entsprechend benannte Verzeichnisse tun. Neben der Untersuchung von Domainnamen und *Labels* müssen also auch Verzeichnisnamen, die in einer URL vorkommen, auf Ortsinformationen untersucht werden.

2.4.2 Der Titel

Der Titel einer Webseite wird in der *HEAD*-Sektion eines *HTML*-Dokumentes angegeben. Nach der *HTML 4.01 Spezifikation* (siehe [RHJ99]) muss jede gültige Webseite einen Titel besitzen, der in der *HEAD*-Sektion des *HTML*-Dokumentes mit dem *TITLE*-Tag spezifiziert wird. Bei der *HTML*-Version dieser Arbeit könnte dies so aussehen:

```
<HEAD>
   <TITLE> Ortsbezug für Web-Inhalte </TITLE>
   ...
</HEAD>
```

Der Titel ist eine der wichtigsten Elemente einer Webseite, wenn der Inhalt der Seite im Mittelpunkt steht. Der Verfasser ist gut beraten, einen aussagekräftigen Titel für sein Dokument zu wählen, da Suchmaschinen in diesem Teil der Webseite gefundene Wörter bei der Indexierung sehr hoch bewerten (siehe [BP98] und [Lyc02]). Ist der Titel gut gewählt, dann wird die Webseite bei einer entsprechenden Suchanfrage einen der vordersten Plätze beim *Ranking* belegen.

Hat eine Webseite einen Ortsbezug, dann besteht also eine relativ gute Chance, dass im Titel diese Ortsinformation angegeben wird. Aus diesem Grund muss während der Suche nach Ortsinformationen der Titel einer Webseite gründlich analysiert werden.

2.4.3 Meta-Tags

Seit der Version 2.0 des *HTML*-Standards besteht die Möglichkeit, Metadaten in einem *HTML*-Dokument abzuspeichern. Metadaten sind für den Webseitenbetrachter

unsichtbar; sie sollen die maschinelle Verarbeitung von *HTML*-Dokumenten, zum Beispiel durch Suchmaschinen, vereinfachen.

Metadatensätze werden nach der *HTML 4.01 Spezifikation* durch das *HTML*-Element META angegeben. Der Meta-Tag besteht aus Attribut-Wert-Paaren, wie zum Beispiel [*name : „Author“*] und [*content : „Mihály Jakob“*]. Dabei sind *name* und *content* Attribute, während *„Author“* und *„Mihály Jakob“* die zugehörigen Werte darstellen. Ein Meta-Tag für die *HTML*-Version dieser Arbeit könnte damit so aussehen:

```
<META name="Author" content="Mihály Jakob">
```

Ähnlich wie der Titel von Webseiten, finden Meta-Tags in der *HEAD*-Sektion des jeweiligen *HTML*-Dokumentes Platz:

```
<HEAD>
   ...
   <META ... >
   ...
</HEAD>
```

In der Studienarbeit des Autors [Süt01] wurde die Extrahierung von Ortsinformationen aus Meta-Tags einer Webseite ausführlich behandelt. Die Möglichkeit, Metadaten über eine Webseite in einem separaten Dokument nach dem *Resource Description Framework* anzugeben, wurde ebenfalls untersucht. Der während der Arbeit implementierte Webroboter, *DCbot*, konnte Ortsinformationen in Webseiten nur dann finden, wenn diese einen entsprechenden Meta-Tag oder *RDF*-Satz nach dem *DCMES*-Standard (siehe Kapitel 2.4.3.2 auf Seite 25) enthielten. Ein Beispiel Meta-Tag mit Ortsinformation wäre:

```
<meta name="dc.coverage" content="Lat: 48 47 N Long: 9 12 E">
```

Leider sind spezifische Ortsangaben in Webseiten (zum Beispiel Meta-Tags mit dem Element dc.coverage) wenig verbreitet, obwohl andere *DCMES*-Elemente weitaus öfter verwendet werden. Welche weiteren Metadatenelemente Ortsinformationen tragen können, wird deshalb an dieser Stelle untersucht.

2.4.3.1 Standard Metadaten

Die *HTML 4.01 Spezifikation* beschreibt präzise, wie Meta-Tags in ein *HTML*-Dokument einzubetten sind. Sie gibt jedoch keine Empfehlung, welche Metadatenelemente verwendet werden sollen. Die meisten Suchmaschinen beachten während der Indexierung von Webseiten einen Meta-Tag nur dann, wenn eines der folgenden Metadatenelemente darin vorkommt (siehe [Sul02]):

- **keywords**

  Werte dieses Elements sind wichtige Stichwörter, die den Inhalt der Webseite charakterisieren. Beispiel:

```
<meta name="keywords" content="Ortsinformation, Information Retrieval, ... ">
```

- **description**

  Wert dieses Elements ist eine kurze Zusammenfassung des Inhalts der Webseite. Beispiel:

```
<meta name="description" content="Die Diplomarbeit „Ortsbezug für Web-
                                  Inhalte“ beschäftigt sich mit der Extrah-
                                  ierung von Ortsinformationen aus Web-
                                  seiten. ...">
```

- **robots**

  Dieses Element ist Teil des *Standard for Robot Exclusion* [Kos94], der die Analyse von Webseiten durch Webroboter, wenn unerwünscht, unterbinden soll.

  Das *robots*-Element teilt einem Webroboter (zum Beispiel *crawler* einer Suchmaschine) mit, in welcher Weise er die Webseite untersuchen darf. Für dieses Element sind folgende Werte zulässig:

  noindex: Der Webroboter darf das *HTML*-Dokument nicht untersuchen.

  nofollow: Der Webroboter darf keine Verweise des *HTML*-Dokumentes verfolgen.

  none: Wirkt wie „*noindex*“ und „*nofollow*“ zusammen.

  Beispiel:

```
<meta name="robots" content="none">
```

Ein Testlauf mit *DCbot*, während dessen 25.000 Webseiten untersucht wurden, ergab folgende Häufigkeiten für die Benutzung von Metadatenelementen.

Metadatenelement	Häufigkeit	Anteil
keywords	8.453	33,81 %
description	8.164	32,66 %

Tabelle 2-4: Vorkommenshäufigkeiten von Standard-Meta-Tags

Hat eine Webseite einen Ortsbezug, so ist es wahrscheinlich, dass die Metadatenelemente *keywords* und *description* entsprechende Ortsangaben enthalten. Aus diesem Grund müssen sie auf jeden Fall auf Ortsinformationen untersucht werden.

Das *robots*-Element kann nur die vorhin beschriebenen Werte annehmen, deswegen kann sie keine Ortsinformationen speichern. Der Webroboter *DCbot* unterstützt diesen Standard und hält sich an die Vorgaben in *robots*-Meta-Tags.

2.4.3.2 DCMI Metadaten

Die *Dublin Core Metadata Initiative (DCMI)* ist eine gemeinnützige Organisation, die die Verbreitung von Metadaten, vor allem für die Inhalte des World Wide Web, fördert. Das *Dublin Core Metadata Element Set (DCMES)* (siehe [DCM99]) ist ein Metada-

tenstandard der *DCMI*, für Dokumente jeglicher Art. Eine ausführliche Beschreibung des Standards und der einzelnen Metadatenelemente ist in [Süt01] zu finden. Hier werden nur diejenigen Metadatenelemente vorgestellt, die Ortsinformationen enthalten können.

- **DC.Title**

  Enthält den Titel der Webseite. Der Wert dieses Elements sollte identisch mit dem Eintrag in dem Title-Tag des *HTML*-Dokumentes sein. Eine zusätzliche Untersuchung ist deswegen überflüssig.

- **DC.Subject**

  Dieses Element enthält das Thema der Webseite bzw. Stichwörter, die den Inhalt beschreiben. Diese Stichwörter stammen optimaler Weise aus einem vordefinierten Vokabular bzw. Klassifikationsschema (zum Beispiel aus dem *TGN*).

  DC.Subject ähnelt dem *keywords*-Element was es zur Folge hat, dass diese beiden Elemente sehr selten zusammen auftreten werden.

- **DC.Description**

  DC.Description enthält eine kurze Zusammenfassung des Webseiteninhaltes. Seine Funktion ist identisch, mit der des *description*-Elementes aus dem vorhergehenden Abschnitt. Kommen beide Elemente gleichzeitig in einem *HTML*-Dokument vor, dann kann das zweite außer Acht gelassen werden.

- **DC.Coverage**

  DC.Coverage kann zum Inhalt passende geographische Positionsdaten oder einen Zeitraum, der die Gültigkeit des Webseiteninhalts beschreibt, enthalten.

Während des vorhin erwähnten Testlaufs mit 25.000 untersuchten Webseiten konnten folgende Häufigkeiten für die Benutzung von *DCMI*-Metadatenelementen ermittelt werden.

Metadatenelement	Häufigkeit	Anteil
DC.Title	71	0,28 %
DC.Subject	41	0,16 %
DC.Description	33	0,13 %
DC.Coverage	18	0,07 %

Tabelle 2-5: Vorkommenshäufigkeiten von *DCMI*-Meta-Tags

Werden während der Analyse einer Webseite Meta-Tags mit den Metadatenelementen *DC.Subject* oder *DC.Description* gefunden, dann müssen diese, ähnlich wie der Titel der Webseite, auf Ortsinformationen untersucht werden. *DC.Coverage* ist ein Spezialfall, da es eventuell genaue geographische Koordinaten (zum Beispiel in der Form von Breiten- und Längengrad) enthalten kann. Ist dies der Fall, dann müssen diese nur noch in das geeignete Format konvertiert werden. Enthält dieses Element eine Ortsangabe in natürlicher Sprache, dann muss die genaue Position aus einer entsprechenden Datenbank (wie *GeoBase*) ausgelesen werden.

2.4.4 Webseitentext

Die Analyse des Webseitentextes ist sicherlich die größte Herausforderung wenn es darauf ankommt, Ortsinformationen aus einer Webseite zu extrahieren. Während der Titel oder die Meta-Tags der Webseite in der Regel nur wenige Wörter beziehungsweise Sätze enthalten, kann der Webseitentext mehrere hundert Seiten umfassen. Wie in diesen Seiten Ortsinformationen ausfindig gemacht werden können, ist die zentrale Frage in diesem Kapitel. Die Untersuchung beschränkt sich auf deutschsprachige Inhalte.

2.4.4.1 Nicht-textuelle Inhalte

Mittlerweile existiert eine breite Palette von Werkzeugen, mit deren Hilfe die Erstellung abwechslungsreicher Webseiten möglich geworden ist. Zu diesen Werkzeugen gehören Skript-Sprachen (wie *JavaScript*, *JScript* oder *Perl*), die die dynamische Gestaltung von Webseiten sowohl auf der Seite des Benutzers als auch auf der Serverseite erlauben. Die Programmiersprache *Java* bietet sogar die Möglichkeit, nahezu beliebige Programminhalte in *Java-Applets* in einem *Browser*-Fenster auszuführen. Durch Technologien wie *Maromedias Flash* oder *Apples Quicktime* können beliebige Animationen oder Videos im World Wide Web bereitgestellt werden. Die Darstellung von Bildern auf Webseiten kann sogar mit einfachen *HTML*-Mitteln bewerkstelligt werden.

Die Analyse solcher multimedialer Inhalte ist eine sehr komplexe Aufgabe, die im Rahmen dieser Arbeit nicht gelöst werden kann. Bereits die Erkennung einer bekannten Sehenswürdigkeit auf einem einfachen Bild erweist sich mit der heutigen Computertechnik als eine große Herausforderung.

Webseiten mit Ortsbezug, die keinen auswertbaren Webseitentext enthalten, können trotzdem analysiert werden, indem die bereits erwähnten Webseitenteile, wie die URL, der Titel oder die Meta-Tags, untersucht werden. Wird auf einer Webseite Text gefunden, dann kann dieser Text auf verschiedenen Arten untersucht werden. Welche dieser Ansätze für die Extrahierung von Ortsinformationen in Frage kommen, wird in den nächsten Kapiteln untersucht.

2.4.4.2 Intelligente Agenten

Wenn sehr viele Dokumente untersucht werden sollen, dann ist es nahe legend, diese Aufgabe automatisiert durchzuführen. Im Bereich der *Information Retrieval* nennt man Softwareprogramme, die autonom mit ihrer Umgebung interagieren, Agenten. Sie werden als lernfähig oder intelligent bezeichnet, wenn sie während ihrer Interaktion mit der Umwelt an Erfahrung gewinnen, und durch den Aufbau einer Wissensbasis ihre Effizienz fortwährend verbessern können [Mit97]. Es stellt sich nun folgende Frage: Muss ein Webroboter lernfähig sein um Ortsinformationen in Webseiten zuverlässig aufspüren zu können?

Zwei bekannte, intelligente Agenten sind der *Internet Learning Agent (ILA)* und *ShopBot* [PDE97]. Beide Agenten sind in der Lage, nach einer Trainingphase mit Beispiel-Webseiten die geforderten Informationen aus diesen Seiten sicher herauszulesen. Im Falle von *ShopBot* sind dies die Preise von beliebigen Produkten. Der größte Nachteil dieser Agenten ist, dass sie nach ihrer Anfragen homogene Antworten erwarten. Bei *ShopBot* sind dies Produktbeschreibungen mit Preisangaben, die bei den meisten Onlinehändlern ein ähnliches Format besitzen.

Zufällig gewählte Webseiten aus dem World Wide Web sind auf jeden Fall heterogen. *DCbot* kann leider nicht davon ausgehen, dass Ortsinformationen in diesen Seiten immer eine einheitliche Form besitzen, beziehungsweise, dass sie immer an einer bestimmten Stelle der Webseite zu finden sind. Einige Regelmäßigkeiten, wie zum Beispiel häufig vorkommende Wörter vor Ortsnamen, können trotzdem verwendet und eventuell auch erlernt werden (siehe „semantische Regeln“ in Kapitel 3.4.2.2).

2.4.4.3 Suchmaschinen

Die größten Suchmaschinen (siehe [Sul03]) haben mehrere hundert Millionen indexierte Webseiten in ihrer Datenbank. Um diese immense Menge an Daten effizient verarbeiten zu können, sind Suchmaschinen modular aufgebaut. Die Sammlung von Webseiten (durch so genannte *Crawler*), ihre Untersuchung (*Indexing*) und die Beantwortung von Benutzeranfragen werden durch die verschiedenen Module parallel ausgeführt. *DCbot* vereinigt zwei dieser Komponenten; jede besuchte (*Crawling*) Webseite wird sofort analysiert (*Indexing*) und eventuell gefundenen Ortsinformationen werden anschließend gespeichert. Während die *Crawler* der meisten Suchmaschinen ähnlich arbeiten (rekursive Verfolgung von Hyperlinks), bestehen zwischen den Indexierungskomponenten teilweise große unterschiede. Um Manipulationen, betreffend der Reihenfolge von Webseiten in dem Suchergebnis einer Suchmaschine, zu vermeiden, wird die genaue Indexierungstechnik der jeweiligen Suchmaschine nicht bekannt gegeben. Die grundlegende Verfahrensweise kann in verschiedenen Veröffentlichungen (zum Beispiel [BP98]) trotzdem in Erfahrung gebracht werden. Diese Ansätze können hilfreich sein, um Ortsinformationen in Webseiten aufzuspüren.

2.4.4.4 Heurismen für den Webseitentext

Eine Möglichkeit, den Inhalt einer Webseite zu erfassen, ist die Bildung eines Wortvektors, der die häufigsten Wörter der Seite in absteigender Reihenfolge enthält. Aus den Suchbegriffen einer Benutzeranfrage kann ein ähnlicher Vektor gebildet und mit Vektoren von Webseiten verglichen werden. Dieser Ansatz ist einleuchtend. Kommen Wörter in einer Webseite oft vor, dann ist diese Seite für eine Suchanfrage mit den gleichen Begriffen relevant. Dies gilt auch für Ortsinformationen, weshalb häufig vorkommende Wörter einer Webseite auf einen möglichen Ortsbezug untersucht werden müssen.

Dokumente werden in der Regel erstellt, um von jemandem gelesen zu werden. Gerade im World Wide Web, wo Benutzer durch Hyperlinks zum schnellen Wechseln

zwischen Webseiten „verleitet“ werden, ist es wichtig, das Interesse des Lesers zu erwecken, um ihn zum Fertiglesen zu bewegen. Aus diesem Grund dienen die ersten Sätze einer Webseite oft als Zusammenfassung des Seiteninhaltes. Die Untersuchung dieser Sätze auf Ortsinformationen ist deshalb empfehlenswert.

Das Bild, das sich durch die Verwendung dieser beiden Ansätze von einer Webseite gewinnen lässt, kann den Gesamtinhalt dieser Seite noch nicht einfangen. Eine Webseite kann unter Umständen mehrere tausend Wörter enthalten; nicht alle können mit einer Datenbank verglichen und darauf getestet werden, ob sie eine Ortsinformation darstellen. Es müssen semantische Methoden angewendet werden, die unabhängig von der Position oder der Häufigkeit eines Wortes auf einer Webseite arbeiten.

Die semantische Analyse (im Sinne von dem Verstehen des Inhaltes) von geschriebenen Texten durch Computer ist eine komplexe Aufgabe, die im Rahmen dieser Arbeit nicht einmal ansatzweise gelöst werden kann. Die Verwendung beziehungsweise die Erkennung von einfachen, sprachbestimmten Regelmäßigkeiten, kann die große Anzahl von Wörtern einer Webseite trotzdem auf wenige gute Kandidaten reduzieren. Drei dieser Aspekte sind:

1. Erkennung von Eigennamen

   Ortsnamen, die in Kapitel 2.2 auf Seite 7 vorgestellt wurden sind Eigennamen, die in Deutsch groß geschrieben werden. Aus diesem Grund werden nur groß geschriebene Wörter als mögliche Kandidaten für eine Ortsinformation in Betracht gezogen. Selbstverständlich muss bei dieser Vereinfachung eine gewisse Fehlerrate (falsche Positive) in Kauf genommen werden. Eine alternative Möglichkeit wäre der Einsatz eines grammatikalischen Analysewerkzeuges, das die Kategorisierung durchführt.

2. Schließen von der Syntax auf die Bedeutung

   Die Bedeutung mancher Wörter kann durch eine einfache syntaktische Überprüfung geschätzt werden. In Deutschland werden Postleitzahlen durch fünfstellige Zahlen dargestellt. Wird also auf einer Webseite eine fünfstellige Zahl gefunden, kann ihre Umgebung auf weitere Ortsinformationen (Straßennamen, Städtenamen) untersucht werden. Diese bilden eventuell zusammen mit der Postleitzahl eine Adresse, die mit einer genauen Position verbunden werden kann.

   Auch Straßennamen können oft durch eine einfache Syntaxüberprüfung erkannt werden. Die meisten Straßennamen haben die Form

   <Name> <Typ> (zwei Wörter, z.B. „*Stuttgarter Straße*“)

   wobei <Typ> das Wort „*Straße*“, „*Weg*“, „*Allee*“ usw. ist, oder die Form

   <Name><Endung> (ein Wort, z.B. „*Bahnhofweg*“)

   wobei <Endung> wieder einer der oben genannten Schlüsselwörter ist. Diesmal als Wortendung natürlich klein geschrieben.

Selbstverständlich können mit dieser Methode auch Bauwerke und natürliche geographische Stätten erkannt werden. In diesen Fällen muss eine Webseite auf Schlüsselwörter wie *„Brücke“* oder *„See“* überprüft werden.

3. Kontextanalyse

   Oft ist es einem Wort oder einer Wortgruppe nicht anzusehen, ob sie eine Ortsinformation darstellen. In diesen Fällen kann zusätzlich die Wortumgebung untersucht werden. Dazu werden drei Arten von Regeln aufgestellt:

<wort> <Ortsinformation>	(z.B. *„in <Ortsinformation>“*)
<wort> <wort> <Ortsinformation>	(z.B. *„Nähe von <Ortsinformation>“*)
<wort> <Ortsinformation> <wort>	(z.B. *„in <Ortsinformation> lebend“*)

   Durch diese Regeln können zum Beispiel Ausdrücke, wie *„in Stuttgart“*, *„Nähe von Stuttgart“* oder *„in Stuttgart lebend“* erkannt und Ortsinformationen (in diesem Fall *„Stuttgart“*) gefunden werden. Ein besonderer Reiz besteht darin, nicht mit statischen Regeln zu arbeiten, sondern Regeln während der Suche im World Wide Web vom Webroboter erlernen zu lassen.

2.4.5 Hyperlinks

Einer der wichtigsten Gründe für den großen Erfolg des World Wide Webs ist die Möglichkeit, schnell zwischen benachbarten Webseiten zu wechseln. Die Idee, Dokumente miteinander direkt zu verbinden, gibt es bereits seit 1945, als Vannevar Bush seine Zukunftsvision *Memex*, eine Maschine als Universaldatenbank mit verbundenen Dokumenten, vorstellte. Er hat sicherlich nicht geahnt, dass die *Memex* der Zukunft, das Internet, sich über die ganze Erde erstrecken wird.

Hyperlinks, die Verweise zwischen Webseiten im World Wide Web, sind mehr als einfache Verbindungen zwischen einer Sammlung von Dokumenten. Die Möglichkeit, jedem Hyperlink durch den Hyperlink-Text eine spezielle Bedeutung zu geben, erlaubt die Definition von beliebigen Beziehungen zwischen Webseiten und Webseitenteilen. Diese Freiheit hat leider auch einen Nachteil. Forscher, die sich mit der Bedeutungskonstitution im Hypertext beschäftigen (siehe [Sch01]), sind sich einig, dass eine Kategorisierung von Hyperlinks nicht möglich ist. Die maschinelle Interpretation von Beziehungen zwischen Webseiten führt auf das Problem der maschinellen Verarbeitung (im Sinne von Verständnis) von natürlichsprachlichen Texten zurück.

Es gibt trotzdem Wege, die Informationen, die in Hyperlinks stecken, wenigstens teilweise zu nutzen. Die Suchmaschine Google verwendet einen so genannten *PageRank* Verfahren (siehe [BP98]), um Webseiten zu bewerten. Nach dieser Bewertungsmethode hat eine Webseite, auf die viele andere Webseiten verweisen, eine höhere Bedeutung als Seiten, die selten referenziert werden. Dieser plausible Ansatz kommt zwar ohne die Analyse von Hyperlink-Texten aus, erfordert aber die Untersuchung von vielen Millionen Webseiten und Hyperlinks, um repräsentativ und aussagekräftig zu sein.

Der *PageRank* Ansatz wäre sicherlich auch während der Suche nach Webseiten mit Ortsinformationen nützlich. Eine Webseite, die von anderen Seiten mit gemeinsamer Ortsinformation (z.B. Webseiten über die Stadt *Stuttgart*) referenziert wird, könnte ebenfalls diesem Ort zugeordnet werden. Leider ist der Aufwand für die Berechnung eines *PageRank* dieser Art, wie vorhin angedeutet, unakzeptabel hoch.

Obwohl die Bedeutung von Hyperlink-Texten maschinell noch nicht erfasst werden kann, bieten sich die Analysemöglichkeiten, die in Kapitel 2.4.4.4 auf Seite 28 vorgestellt wurden, für die Untersuchung von Hyperlinks an. Gefundenen Ortsinformationen können mit der referenzierenden oder mit der referenzierten Webseite verbunden werden.

2.5 Zusammenfassung

In den vorangehenden Unterkapiteln wurden zahlreiche Wege aufgezeigt in den verschieden Teilen einer Webseite Ortsinformationen aufzuspüren. Die wichtigsten gesuchten Ortsinformationen dabei waren die Namen von

- natürlichen geographischen Stätten
- Bauwerken
- politischen Gebietsnamen
- und administrativen Gebietsnamen.

Die wichtigsten zu untersuchenden Teile einer Webseite für die Suche nach Ortsinformationen sind

- die URL
- der Titel
- die Meta-Tags
- der Webseitentext
- und die Hyperlinks.

Abhängig von dem jeweiligen Webseitenteil, der analysiert werden soll, kann eine der folgenden Methoden, um mögliche Ortsinformationen aufzuspüren, angewendet werden:

- Untersuchung aller Eigennamen eines Webseitenteils
- Untersuchung der ersten n oder der n häufigsten Eigennamen eines Webseitenteils
- Überprüfung, ob Eigennamen spezielle Wörter sind (z.B. „*Theater*“, „*70569*“) oder spezielle Endungen besitzen (z.B. „*...straße*“)
- Anwendung von semantischen, kontextbezogen Regeln

Viel versprechende Webseitenkomponenten sind auf jeden Fall der Titel sowie der *description*- und der *keywords*-Meta-Tag. Sie sollten mit wenigen Worten den Inhalt einer

Webseite zusammenfassen, was im Falle einer Webseite mit Ortsbezug auf entsprechende Informationen in den erwähnten Komponenten hoffen lässt.

KAPITEL 3 DCBOT

In diesem Kapitel wird die Implementierungslösung der Arbeit, der Webroboter *DCbot*, vorgestellt.

3.1 Zielsetzung

In dem vorangehenden Kapitel wurden verschiedene Möglichkeiten vorgestellt, die zum Auffinden von Ortsinformationen in Webseiten verwendet werden können. Dabei wurden die einzelnen Webseitenteile, und die für diese Komponenten in Frage kommenden Analysemethoden, untersucht. Verfahren, die den meisten Erfolg versprechen, sollen im Rahmen dieser Arbeit implementiert werden. Als Ergebnis soll ein Webroboter entstehen, der autonom im World Wide Web nach Webseiten mit Ortsinformation sucht und gefundenen Seiten zusammen mit der ermittelten geographischen Position speichert. Die Speicherung soll entweder in einem *Spatial Model Server* (siehe Kapitel 3.4.5.1 auf Seite 52) oder in einem herkömmlichen Datenbanksystem erfolgen.

3.2 Vorgehensweise

In der Studienarbeit des Autors [Süt01] wurde die Möglichkeit untersucht, einen Ortsbezug zu Webseiten über Metadaten herzustellen. Dabei wurden insbesondere spezielle Meta-Tags ausgewertet, die geographische Informationen unter Verwendung des *DC.Coverage* Metadatenelements aus dem *Dublin Core Metadata Element Set* enthielten. Als Implementierungsergebnis der Arbeit entstand der Webroboter *DCbot*, der wichtige, für diese Arbeit relevante, Implementierungsaspekte bereits erfolgreich umsetzt. Die wichtigsten dieser Eigenschaften sind:

- autonomes Durchwandern des World Wide Web durch rekursives Verfolgen von Verweisen
- Analyse von *DC.Coverage* Meta-Tags
- Analyse von *RDF*-Beschreibungen in separaten Metadaten-Dateien
- Speicherung von Webseiten im *Spatial Model Server* oder im *DB2 UDB*

Im Rahmen dieser Arbeit wird der Webroboter um folgende Funktionalität erweitert:

- ausführliche Analyse aller Webseitenkomponenten und Extrahierung von möglichen Ortsinformationen aus diesen Komponenten
- Zugriff auf einen Katalog, der Ortsnamen mit geographischen Positionen verbindet um mögliche Ortsinformationen zu verifizieren
- Speicherung umfangreicher Statistiken über den Analyseprozess

Der oben erwähnte Katalog (siehe Kapitel 2.3.3 auf Seite 15) wird ebenfalls im Rahmen dieser Arbeit entworfen und trägt den Namen *GeoBase*. Er wird sowohl bereits existierende Datensätze aus einer kommerziellen Datenbasis (siehe Kapitel 2.3.2 auf Seite 14) als auch einige, zu Testzwecken manuell erstellte Einträge, enthalten.

3.3 Entwicklungsumgebung

Der Webroboter *DCbot* ist in der Programmiersprache *Java* implementiert. Für die Erweiterungen wurde die *Java Version 1.4.0* verwendet. Die Implementierung erfolgte mit dem freien Entwicklungstool *NetBeans IDE 3.4*, beziehungsweise mit dem *Java JDK 1.4* unter den Betriebssystemen *Windows XP* und *SunOS 5.8*. Für die Speicherung der Ergebnisse wurde das Datenbanksystem *DB2 UDB V7.2* verwendet.

3.4 Eigenschaften des Webroboters

In der Studienarbeit des Autors [Süt01] wurden sowohl die bisher vorhandene Funktionalität, als auch Implementierungsdetails der bereits enthaltenen Komponenten des Webroboters ausführlich beschrieben. Im Folgenden wird ein Gesamtüberblick über den Funktionsumfang des Webroboters gegeben, wobei der Schwerpunkt auf den neuen Funktionen liegt. Danach werden die Implementierungsaspekte der Erweiterungen vorgestellt. Auf bereits enthaltene Funktionalität oder Komponenten wird nur dann eingegangen, wenn dies das Gesamtverständnis des Systems erleichtert.

3.4.1 Systemarchitektur

Eine Übersicht über die Hauptkomponenten von *DCbot* wird in Abbildung 3-1 dargestellt. Der Webroboter verwaltet zwei URL-Listen. Eine für neue URLs, die noch zu besuchen sind, und eine für die bereits besuchten URLs. Während einer Iteration holt der Webroboter eine „neue“ URL aus der Liste der neuen URLs und empfängt die dazugehörige Webseite über das *HTTP-Interface*. Der *Content Handler* analysiert die einzelnen Webseitenkomponenten, und speichert die Webseite im Falle der erfolgreichen Ermittlung eines Ortsbezugs mittels dem *SpaSe-Interface* in einem *Spatial Model Server* oder mittels dem *JDBC-Interface* in einer relationalen Datenbank. Der Katalog *GeoBase*, der Ortsnamen mit geographischen Positionen verbindet, wird ebenfalls in der gewählten Datenbank gespeichert, und kann über das *JDBC-Interface* angesprochen werden. Alle Verweise der gerade untersuchten Webseite werden extrahiert und zu der Liste der zu besuchenden URLs hinzugefügt.

Beim Aufruf von *DCbot* muss eine Start-URL übergeben werden. Der Suchprozess startet ausgehend von dieser URL. Ist die Liste der zu besuchenden URLs leer oder wurde die maximale Anzahl der zu besuchenden URLs (diese kann in der Konfigurationsdatei spezifiziert werden) erreicht, endet der Suchprozess.

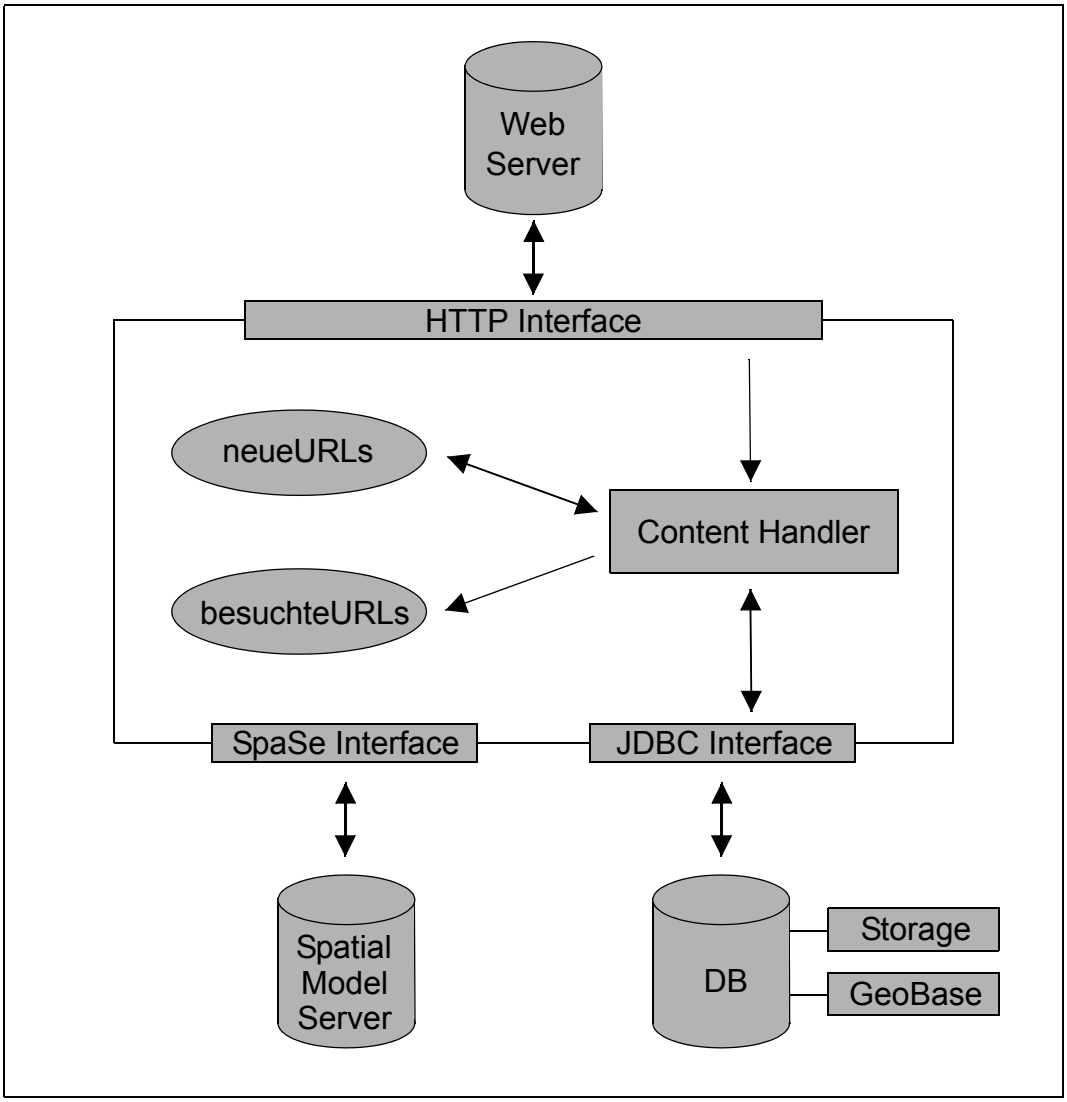

Abbildung 3-1: Architektur von *DCbot*

3.4.2 Funktionsumfang

3.4.2.1 Allgemeine Eigenschaften

- Automatisierte Suche nach Webseiten mit Ortsinformation im World Wide Web ohne benötigte Interaktion des Benutzers.
- Konfigurationsparameter werden in einer Konfigurationsdatei gehalten; damit sind die Einstellungen überschaubar und leicht abänderbar.
- Der Suchbereich des Webroboters kann auf einen URL-Bereich oder auf eine Domain beschränkt werden.
- Die maximale Anzahl der zu besuchenden Webseiten, sowie ihre maximale Entfernung von der Start-URL aus, kann festgelegt werden.

- Möglichkeit der Angabe von Schlüsselwörtern, die das Finden von Ortsinformationen durch den Webroboter positiv beeinflussen.
- Möglichkeit der Angabe von semantischen Regeln in einer Regel-Datei, die das Finden von Ortsinformationen durch den Webroboter positiv beeinflussen.
- Automatische Extrahierung von semantischen Regeln aus Webseiten durch den Webroboter und damit stetige Verbesserung der Effizienz
- Zusätzliche Speicherung von beliebigen Meta-Tag-Inhalten aus untersuchten Webseiten; Konfigurierbarkeit erwünschter Meta-Tags in einer *map*-Datei.
- Unterstützung des *Standard for Robot Exclusion (SRE)*.
- Speicherung der Ergebnisse in einem *Spatial Model Server* oder einem herkömmlichen Datenbanksystem.
- Speicherung ausführlicher statistischer Daten über den Suchprozess in einer Statistik-Datei

3.4.2.2 Analyseverfahren

Die grundlegende Idee hinter der Analysekomponente von *DCbot* ist die systematische Untersuchung von Webseitenteilen und die Generierung von Kandidaten, die eventuell Ortsinformationen darstellen. Alle extrahierten Kandidaten werden mit dem Katalog *GeoBase* auf ihre Richtigkeit hin überprüft. Aus den verifizierten Ortsinformationen wird die bestmögliche Position ermittelt und die Webseite mit dieser Position gespeichert. Die folgenden Verfahren beschreiben Möglichkeiten, aus den Webseitenteilen Kandidaten zu extrahieren. Dabei werden nur mögliche Eigennamen oder aus mehreren Wörtern bestehende Ausdrücke, die mögliche Eigennamen darstellen, beachtet. Im Deutschen werden Substantive und Eigennamen durch die Großschreibung von anderen Wortarten unterschieden. Dies macht die Identifizierung dieser Eigennamen einfach. In anderen Sprachen muss zu diesem Zweck ein entsprechendes Grammatikwerkzeug eingesetzt werden.

1. Extrahierung aller Eigennamen

   Dies ist wohl die einfachste Lösung und eignet sich vor allem für Webseitenteile mit relativ wenig Wörtern und hoher Signifikanz. Diese Methode verwendet *DCbot* für die *URL*, für den *Titel* und für die *Verweise* einer Webseite.

   Außerdem werden alle Eigennamen aus dem *DC.Coverage* und dem *DC.Coverage.Spatial* Meta-Tags extrahiert, wobei diese Tags in der Regel nur eine Ortsangabe enthalten. Da sie speziell für die Angabe von Ortsinformationen verwendet werden können, wird ihr Inhalt zusätzlich analysiert. *DCbot* versucht aus diesen Tags eine Positionsangabe in der Längengrad-Breitengrad-Darstellung oder im *WGS84* Format zu parsen. Schlägt dies fehl, dann werden die Inhalte wie normale Eigennamen behandelt und zu der Kandidatenliste hinzugefügt.

   Zusätzlich werden in manchen Fällen Metadaten über Webseiten in separaten Metadaten-Dateien angegeben. Diese benutzen *XML* und das *Resource Description Framework* und spezifizieren Meta-Tag-ähnlichen Metainformationen (für weitere

Details siehe [Süt01]). Informationen in solchen Dateien werden in Meta-Tags umgewandelt und entsprechend untersucht.

Webseitenteile mit vielen Wörtern erfordern andere Strategien. Es ist aus Effizienzgründen nicht ratsam, alle Wörter beziehungsweise alle Ausdrücke, bestehend aus mehreren Wörtern, auf einer Webseite zu untersuchen. Bei diesem Vorgehen würde die Anzahl generierter Kandidaten rasch ansteigen. Ihre Berechnung und der Abgleich mit dem Katalog würde einen nicht unerheblichen Zeitaufwand verursachen, der insbesondere wegen der Generierung vieler unnötigen Kandidaten nicht gerechtfertigt werden kann. Die nächsten zwei Methoden selektieren nur bestimmte Eigennamen aus einer größeren Webseitenkomponente und beschränken damit die Anzahl generierter Kandidaten.

2. Extrahierung der ersten n Eigennamen

   Die ersten paar Zeilen eines Dokuments geben oft einen guten Überblick über den Gesamtinhalt. Dies ist insbesondere bei Webdokumenten wichtig, wo der Leser oft schell herausfinden möchte, ob die Seite für ihn von Interesse ist. *DCbot* bietet die Möglichkeit, die Anzahl der ersten n $(n \geq 0)$ zu untersuchenden Eigennamen für ein Webseitenteil anzugeben. Diese Vorgehensweise wird zum Beispiel bei dem textuellen Inhalt einer Webseite (*body-Sektion*) angewendet. Der Konfigurationsparameter für diese Einstellung lautet *FirstNBodyWords* und kann in der Konfigurationsdatei von *DCbot* spezifiziert werden.

   Viele Webseiten verwenden einen *description*-Meta-Tag, der eine kurze Zusammenfassung des Webseiteninhalts enthält. Wie viele Wörter aus diesem Meta-Tag untersucht werden sollen, kann mit dem Konfigurationsparameter *MaxDescriptionWords* festgelegt werden.

   Ein weiterer Meta-Tag, der oft in Webseiten verwendet wird und Ortsinformationen enthalten kann, ist der *keywords*-Meta-Tag. Ähnlich wie bei dem *description*-Tag kann die Anzahl der zu untersuchenden Wörter angegeben werden. Der entsprechende Parameter der Konfigurationsdatei lautet *MaxKeywords*.

3. Extrahierung der n häufigsten Eigennamen

   Die Ermittlung der häufigsten n $(n \geq 0)$ Eigennamen macht nur bei größeren Textpassagen Sinn. Aus diesem Grund wird diese Methode ausschließlich bei dem Webseitentext angewendet. Der entsprechende Konfigurationsparameter lautet *FirstNFrequentBodyWords*.

4. Erkennung von Eigennamen durch Schlüsselwörter

   Bisher wurden nur Methoden vorgestellt, die die Generierung von, aus einem Wort bestehenden, Eigennamen erlaubten. Da viele Eigennamen sich aus mehreren Wörtern zusammensetzen, müssen zu ihrer Erkennung andere Verfahren angewendet werden. Um dies zu erreichen, verwendet *DCbot* spezielle Schlüsselwörter um Namen geographischer Stätten, Bauwerke oder Institutionen auf Webseiten zu erkennen. Dazu werden Wörter in Webseitenteilen mit den Schlüsselwörtern verglichen. Ist das gerade untersuchte Wort ein Schlüsselwort

oder endet es auf ein Schlüsselwort, dann werden aus der Umgebung dieses Wortes mehrere Kandidaten generiert.

Wird zum Beispiel das Schlüsselwort *„Turm“* als Endung des Wortes *„Fernsehturm“* in einem Webseitenteil entdeckt, kann der Name *„Stuttgarter Fernsehturm“* aus dem Kontext extrahiert werden. Bei der Untersuchung des Kontextes werden sowohl Wörter vor dem entdeckten Schlüsselwort als auch Wörter danach miteinbezogen, um mögliche Eigennamen mit maximal drei Wörtern zu erkennen. Dadurch hätte der Webroboter auch die Namen *„Eiffelturm“* oder *„Chinesischer Turm München“* erkannt. Dabei können auch falsche Kandidaten generiert werden. Die endgültige Richtigkeit eines Eigennamens kann nur nach der Überprüfung mit dem Katalog festgestellt werden.

Schlüsselwörter können in der Konfigurationsdatei des Webroboters spezifiziert werden. Sie werden für geographische Stätten (zum Beispiel *„Berg“* oder *„See“*) mit dem Konfigurationsparameter *GeoSiteIndicators*, für Gebäude beziehungsweise Institutionen (zum Beispiel *„Theater“* oder *„Universität“*) mit dem Parameter *StructureIndicators* angegeben.

Eine analoge Erkennungsvorschrift wird für die Identifizierung von Straßennamen verwendet. Straßennamen sind allerdings keine eindeutigen Ortsinformationen, da sie alleine in einer Stadt mehrmals vorkommen können. Die Erkennung von Straßennamen wird jedoch bei der Suche nach Adressen auf Webseiten verwendet.

5. Erkennung von Eigennamen durch semantische Regeln

   Es ist nicht immer möglich, Eigennamen, die Ortsinformationen auf Webseiten darstellen, mittels Schlüsselwörter zu identifizieren. *DCbot* arbeitet mit einer zusätzlichen Methode, die weder von der Position von Wörtern auf einer Webseite abhängig ist, noch vom Benutzer angegebenen Schlüsselwörter benötigt. Der Webroboter benutzt semantische Regeln, die den Kontext, in dem Ortsinformationen gefunden wurden, einfangen. Es werden momentan folgende drei Regelarten verwendet:

wort <spatial_info>	(z.B. *„in <spatial_info>“*)
wort wort <spatial_info>	(z.B. *„Nähe von <spatial_info>“*)
wort <spatial_info> wort	(z.B. *„in <spatial_info> zu“*)

   Dabei steht „wort“ für ein beliebiges Wort und „<spatial_info>“ für die zu findende Ortsinformation. Demnach würde die Regel *„in <spatial_info> zu“* aus dem Satz *„Das beste Restaurant im Raum Stuttgart ist in Bad Cannstatt zu finden.“* die Ortsinformation *„Bad Cannstatt“* extrahieren.

   Der Webroboter erlernt diese Regeln während der Analyse von Webseiten automatisch und speichert sie in einer speziellen Regel-Datei. Jedes Mal wenn eine Ortsinformation verifiziert wird, extrahiert *DCbot* aus dem Kontext drei neue Regeln, die den oben vorgestellten Regelarten entsprechen. Dadurch wird eine permanente aber flexible Datenbasis aufgebaut, die die Effizienz des Webrobo-

ters kontinuierlich steigert. Die Datenbasis wird bei jedem Neustart des Webroboters eingelesen, und während dem Analyseprozess durch das Hinzufügen von neuen Regeln stetig verbessert.

Das Erlernen der semantischen Regeln kann durch mehrere Konfigurationsparameter den Bedürfnissen des Benutzers angepasst werden. Diese Parameter werden in Kapitel 3.4.3 auf Seite 42 detailliert beschrieben.

6. Adressenerkennung

   Gemäß §6 des Teledienstegesetzes (TDG) sind kommerzielle Unternehmen und Dienstanbieter, die im World Wide Web ihre Produkte und Dienste vermarkten, verpflichtet, ein Impressum anzugeben, das detaillierte Kontaktinformationen enthält. Ende November 2002 hat das Landgericht Düsseldorf entschieden, dass ein Verstoß gegen die Kennzeichnungspflicht wettbewerbswidrig im Sinne des Gesetzes gegen den unlauteren Wettbewerb ist. Als Folgewirkung dieser Entscheidung ist es in der näheren Zukunft zu erwarten, dass die meisten Unternehmen und Dienstanbieter sich an die Kennzeichnungspflicht halten und unter anderem ihre vollständige Anschrift auf ihrer Webseite veröffentlichen.

   Aus diesem Grund verwendet *DCbot* eine Analysemethode, die die Identifizierung postalischer Adressen auf Webseiten ermöglicht. Dieses Verfahren basiert auf der Erkennung von Straßennamen sowie Postleitzahlen. Ähnlich wie Namen geographischer Stätten oder von Bauwerken, können Straßennamen in vielen Fällen durch spezielle Schlüsselwörter identifiziert werden. Solche Schüsselwörter (wie zum Beispiel *„Straße“* oder *„Allee“*) kann der Anwender in der Konfigurationsdatei von *DCbot* durch den Parameter *StreetIndicators* angeben.

   Wird auf einer Webseite ein möglicher Straßenname oder eine mögliche Postleitzahl erkannt, dann versucht der Webroboter aus dem Kontext eine gültige Adresse zu extrahieren. Dabei werden mögliche Hausnummern und Postleitzahlen durch eine einfache Syntaxüberprüfung identifiziert. Um die Richtigkeit der gefundenen Adresse zu überprüfen werden der Straßenname, der Name der Stadt sowie die Postleitzahl mit dem Katalog abgeglichen. Hierbei wird auch die hierarchische Struktur von *GeoBase* zu Nutze gezogen. Die erkannte Straße muss zu dem Postleitzahlbereich, die Postleitzahl wiederum zu der Stadt passen. Um unvollständige Angaben zu berücksichtigen werden auch partielle Adressen erkannt und verarbeitet. Folgende Formate werden als gültig angesehen:

   <Straße> <Hausnummer> <Postleitzahl> <Stadt>

   <Straße> <Hausnummer> <Stadt>

   <Straße> <Postleitzahl> <Stadt>

   <Straße> <Hausnummer> <Postleitzahl>

   <Postleitzahl> <Stadt>

   Ist eine Adresse vollständig, kann der Webroboter die geographische Position der Straße aus *GeoBase* ermitteln. Dies stellt eine sehr genaue Ortsinformation dar. Bei unvollständigen Adressen kann diese Genauigkeit nicht erreicht werden. Wird

der Straßenname nicht in *GeoBase* gefunden, dann muss die Adresse dem Postleitzahlbereich zugeordnet werden und liefert damit eine deutlich ungenauere Positionsangabe.

3.4.3 Konfigurationsmöglichkeiten

DCbot benötigt während des Suchprozesses keine Interaktion mit dem Benutzer. Es können jedoch einige Einstellungen vor dem Beginn der Suche vorgenommen werden. Nachfolgend werden die Bedienung und die erwähnten Einstellungsparameter detailliert beschrieben. Einige bereits in der Studienarbeit [Süt01] des Autors beschriebene Parameter werden der Vollständigkeit halber in gekürzter Form wiederholt.

3.4.3.1 Programmaufruf

Der Webroboter verwendet den in Kapitel 2.3.3 auf Seite 15 vorgestellten Katalog *GeoBase*, um auf Webseiten gefundenen Ortsinformationskandidaten auf ihre Richtigkeit hin zu überprüfen. Für die sinnvolle Arbeit des Webroboters muss *GeoBase* in der verwendeten Datenbank, die unter dem Parameter *DBName* der Konfigurationsdatei angegeben wird, existieren und mit entsprechenden Daten gefüllt sein. In Kapitel 4.2 auf Seite 71 wird ein Skript vorgestellt, das die benötigten Tabellen anlegt, und sie mit einigen tausend Ortsdaten füllt. Das Skript muss vor dem Aufruf von *DCbot* mindestens ein Mal aufgerufen werden.

Der Webroboter wird durch einen Kommandozeilenaufruf gestartet. Die allgemeine Aufrufsyntax lautet:

```
java DCbot URL [Konfigurationsdatei]
```

Als erster Parameter sollte an *DCbot* die Start-URL übergeben werden, die die URL der ersten zu untersuchenden Webseite darstellt. Die Angabe der Konfigurationsdatei ist freiwillig; wird sie nicht angegeben, versucht das Programm die Datei *DCbot.config* zu laden. Zusätzlich kann dem Webroboter eine Option in der Form

```
java DCbot Option
```

übergeben werden. Mit der Option *„initdb“* werden die benötigten Tabellen für die Speicherung angelegt (im Falle der Datenspeicherung in einem Datenbanksystem), während der Aufruf mit *„deletedb“* die Tabellen wieder aus der Datenbank entfernt.

Der Webroboter unterstützt einige weitere Aufrufoptionen, die in der Studienarbeit des Autors [Süt01] oder in der README-Datei des Webroboters nachgelesen werden können.

3.4.3.2 DCbot Konfigurationsdatei

In der Konfigurationsdatei von *DCbot* kann die Arbeitsweise des Webroboters sehr detailliert festgelegt werden. Parameter können zeilenweise folgender Form definiert werden:

```
Parameter    Wert
Parameter    Wert
...
```

Es besteht die Möglichkeit, einzelne Parameter wegzulassen oder durch das Symbol „#“ auszukommentieren. Für die meisten Konfigurationsparameter existieren sinnvolle Standardwerte.

Die Konfigurationsdatei ist in drei Sektionen unterteilt, die jeweils mehrere Konfigurationsparameter zusammenfassen. Die erste Sektion (*Enviromental Settings*) dient zur Einstellung der allgemeinen Laufumgebung von *DCbot* und enthält folgende Parameter:

- *Homepage*

  Der Webroboter identifiziert sich bei Webservern mit dem *HTTP-Header*

  ```
  User-Agent: DCbot/2.0
  ```

  Zusätzlich kann eine URL übermittelt werden, die mittels dieses Parameters festgelegt werden kann. Für die Beispiel-URL *„http://www.dcbot.org/whoami/“* gestaltet sich der *Header* so:

  ```
  User-Agent: DCbot/2.0 (http://www.dcbot.org/whoami/)
  ```

- *EMail*

  Analog kann mit diesem Konfigurationsparameter eine E-Mail-Adresse (z.B. *„robotadmin@dcbot.org“*) an den Webserver übermittelt werden. Der *HTTP-Header* wird damit durch folgende Zeile ergänzt

  ```
  From: robotadmin@dcbot.org
  ```

- *MapFile*

  In der *DCbot map*-Datei können Meta-Tags spezifiziert werden, deren Inhalte zusätzlich gespeichert werden sollen. Der Standardwert dieses Parameters ist *„DCbot.map“*.

- *RuleFile*

  Die Regel-Datei von *DCbot* enthält semantische Regeln (siehe Kapitel 3.4.2.2), die die Erkennung von Ortsinformationen auf Webseiten unterstützen. Die Standardeinstellung dieses Parameters lautet *„DCbot.rules“*.

- *DBInterface*

  DCbot kann erwünschte Ergebnisse prinzipiell entweder in einem *Spatial Model Server* (Konfigurationswert *„DBI_SpaSe“*) oder in einem herkömmlichen Datenbanksystem (Konfigurationswert *„DBI_JDBC“*) speichern. Wird für die Speicherung das Datenbanksystem gewählt, sollten auch die Parameter *DBJDBCDriver*, *DBJDBCClass* und *DBName* angegeben werden. Wie die Parameternamen andeuten, sollte hierfür ein *JDBC*-Treiber gewählt werden.

- *DBJDBCDriver*

  Mit diesem Parameter kann man den zu verwendenden *JDBC*-Treiber (zum Beispiel *„db2“*) angeben.

- *DBJDBCClass*

  An dieser Stelle sollte der Name der Klasse spezifiziert werden, die den *JDBC*-Treiber implementiert (zum Beispiel *„COM.ibm.db2.jdbc.app.DB2Driver“*).

- *DBName*

  Dieser Parameter bezeichnet den Namen der Datenbank, die für die Speicherung verwendet werden soll.

- *DBTablePrefix*

  Der Präfix der hier spezifiziert wird (zum Beispiel *„dcbot“*), dient zur Benennung der Tabellen, in denen die Ergebnisse der Suche gespeichert werden. Wird der Webroboter mit der Option *„initdb“* aufgerufen, werden zwei Tabellen mit den Namen <*prefix*>webpage (zum Beispiel *„dcbotwebpage“*) und <*prefix*>virtualwebportal (zum Beispiel *„dcbotvirtualwebportal“*) angelegt. Dadurch können Ergebnisse verschiedener Testläufe bequem in verschiedenen Tabellen gespeichert werden.

- *DBPort*

  Hiermit sollte der Port, über den das Datenbanksystem angesprochen werden kann, angegeben werden.

- *DBUser*

  Wenn das Datenbanksystem eine Authentifizierung verlangt, kann mit diesem Parameter die Benutzerkennung festgelegt werden.

- *DBPass*

  Wenn ein Benutzer mittels *DBUser* angegeben wird, muss hier das entsprechende Passwort spezifiziert werden.

- *Proxy*

  Sollen die Webzugriffe von *DCbot* über einen *Proxy-Server* abgewickelt werden, kann mit diesem Parameter der entsprechende Server spezifiziert werden. Als Wert werden der Servername und die Portnummer, durch einen Doppelpunkt getrennt, erwartet. Beispiel:

  ```
  Proxy       webproxy.dcbot.org:8080
  ```

- *NoProxy*

  Wurde ein *Proxy-Server* definiert, können hier Domains angegeben werden, für die kein *Proxy-Server* verwendet werden soll. Domainnamen sollten durch Kommata getrennt werden. Beispiel:

  ```
  NoProxy       .mydomain.com, .ourdomain.net
  ```

Die zweite Sektion (*Retrieval Options*) fasst Einstellungsparameter zusammen, die den Such- und Analyseprozess des Webroboters konfigurieren. Die Optionen im Einzelnen sind:

- *AcceptLanguage*

  Diese Einstellung korrespondiert mit dem gleichnamigen Parameter des *HTTP Request-Headers* und legt fest, welche Sprachen vom *DCbot* akzeptiert werden. Beispiel:

  ```
  AcceptLanguage      de,en;q=0.8
  ```

  Dieser Wert teilt dem entsprechenden Webserver mit, dass Webseiten in Deutsch bevorzugt, aber auch in Englisch akzeptiert werden.

- *ServerMode*

  Dieser Parameter legt fest, welche URL-Bereiche *DCbot* untersuchen soll. Dazu können drei verschiedene Modi verwendet werden:

  - *single*

    In diesem Modus wird der Webroboter auf den URL-Bereich, in dem er gestartet wurde, beschränkt. Dieser URL-Bereich kann eine ganze Domain (z.B. *„http://www.domain.com/“*) umfassen, aber auch nur auf ein Verzeichnis (z.B. *„http://www.domain.com/public/“*) begrenzt werden. Beispiel:

    ```
    ServerMode      single
    ```

  - *multi*

    Im *Multi*-Modus verfolgt *DCbot* beliebige URLs. Eine Beschränkung auf eine spezifische Domain ist jedoch auch in diesem Modus möglich. In diesem Fall wird die gewünschte Domain nach dem Wert *„multi“* durch einen Doppelpunkt getrennt angegeben. Der Webroboter darf damit zwar die angegebene Domain nicht verlassen, wird jedoch nicht auf den URL-Bereich der Start-URL (wie beim *Single-Modus*) beschränkt. Beispiel:

    ```
    ServerMode      multi:.domain.com
    ```

- *indexpage*

  In diesem Modus werden alle Verweise der Startseite in *Single*-Modus verfolgt. Die Startseite dient als ein Index, dessen Einträge eine Liste von URLs, die besucht werden sollen, darstellen. Die Beschränkung der Suche auf eine Domain ist analog zum *Multi*-Modus möglich. Beispiel:

  ```
  ServerMode      indexpage:.domain.com
  ```

- *RequestRate*

  Um eine mögliche Überlastung von befragten Webservern zu vermeiden, kann mit diesem Parameter eine Wartezeit (angegeben in Sekunden) definiert werden. Zwischen zwei Anfragen wird der Webroboter die gegebene Anzahl von Sekunden abwarten. Der Standardwert für diese Einstellung beträgt 5 Sekunden.

- *MaxRequests*

  Mit diesem Parameter kann die gewünschte Anzahl von Webseiten festgelegt werden, die untersucht werden sollen.

- *MaxLinkStorage*

  Dieser Wert reguliert die Größe des URL-Puffers (Liste neuer URLs). Damit lässt sich die „Breite“ der Suche im World Wide Web regeln. Genauer gesagt lässt sich die Anzahl von Webseiten festlegen, die in jeder Entfernung, von der Start-URL aus betrachtet, untersucht werden sollen.

- *MaxDepth*

  Dieser Parameter bestimmt, wie weit sich *DCbot* von der Start-URL aus entfernen darf. Mit Entfernung ist hier die Anzahl der Verweise, die der Webroboter von der Start-URL aus bis zu einer bestimmten Webseite verfolgen muss, gemeint.

- *MaxRules*

  Mit diesem Parameter kann die maximale Anzahl von semantischen Regeln, die in der Regel-Datei gespeichert werden sollen, festgelegt werden. Der Webroboter kann problemlos mit mehreren zehntausend Regeln arbeiten, jedoch sollte dieser Wert nicht zu hoch gewählt werden, damit nicht unnötig Daten im Hauptspeicher gehalten werden müssen. Ein zu kleiner Wert kann jedoch dazu führen, dass eventuell gute Regeln nicht beibehalten werden können. Die Standardeinstellung ist 10.000.

- *GoodRuleThreshold*

  Jede semantische Regel wird durch ihre relative Häufigkeit (Angabe in Prozent) bewertet. Mit diesem Parameter kann festgelegt werden, ab welcher Güte eine Regel von *DCbot* verwendet werden soll. Dieser Schwellwert wird ebenfalls in Prozent angegeben. Alle Regeln die eine höhere Bewertung als der Schwellwert haben, werden für die Analyse von Webseiten eingesetzt.

- *RuleUpdateFrequency*

  Bei jedem Regel-„*Update*“ werden aus Webseiten extrahierten Regeln zur Liste der aktiven Regeln hinzugefügt und auch in die Regel-Datei geschrieben. Dieser Parameter legt fest, wie oft ein „*Update*“ stattfinden soll. Wird die, durch diesen Parameterwert festgelegte Anzahl neuer Regeln erreicht, führt *DCbot* einen Regel-„*Update*“ durch.

- *MaxKeywords*

  Mit diesem Parameter kann festgelegt werden, wie viele Schlüsselwörter aus dem *keywords*-Meta-Tag einer Webseite, untersucht werden sollen.

- *MaxDescriptionWords*

  Dieser Parameter gibt an, wie viele Wörter aus dem *description*-Meta-Tag einer Webseite, untersucht werden sollen.

- *FirstNBodyWords*

  Mit diesem Parameter kann festgelegt werden, wie viele der ersten Wörter des Webseiteninhalts besonders gründlich auf mögliche Ortsinformationen untersucht werden sollen. Andere Methoden (wie z.B. Adressenerkennung, semantische Regeln) werden auf den gesamten Inhalt der Seite angewendet.

- *FirstNFrequentBodyWords*

  Der Webroboter kann die häufigsten, auf einer Webseite verwendeten Wörter, ermitteln. Mit diesem Parameter kann angegeben werden, wie viele der häufigsten Wörter einer Webseite auf Ortsinformationen untersucht werden sollen.

- *StatisticsUpdateFrequency*

  DCbot sammelt ausführliche statistische Daten, die den Suchprozess beschreiben. Die Ergebnisse werden während der Laufzeit des Webroboters regelmäßig in einer Statistik-Datei gespeichert. Nach wie vielen untersuchten Webseiten jeweils die statistischen Daten aktualisiert werden sollen, kann mit diesem Konfigurationsparameter definiert werden.

- *StreetIndicators*

  Dieser Parameter legt Schlüsselwörter fest, die für die Erkennung von Straßennamen verwendet werden. Einzelne Schlüsselwörter werden durch einen Schrägstrich (*Slash*) getrennt. Beispiel:

  ```
  StreetIndicators      Straße/Weg/Allee/...
  ```

- *StructureIndicators*

  Unter diesem Konfigurationsparameter werden Schlüsselwörter, die die Erkennung von Gebäudenamen und Institutionen ermöglichen, definiert. Beispiel:

  ```
  StructureIndicators      Theater/Dom/Universität/...
  ```

- *GeoSiteIndicators*

  Auf ähnliche Weise können mit diesem Parameter Schlüsselwörter spezifiziert werden, die die Erkennung von natürlichen geographischen Stätten unterstützen. Beispiel:

  ```
  GeoSiteIndicators        See/Berg/Fluss/...
  ```

Die dritte Sektion der Konfigurationsdatei enthält zum großen Teil Einstellungen, die die Bewertung auf Webseiten gefundener Ortsinformationen ermöglichen, und damit schließlich diejenigen Webseiten auswählen, für die ausreichend genaue Positionsdaten ermittelt werden konnten.

- *AreaMap*

  Jeder Ortseintrag, der in *GeoBase* gespeichert ist, enthält die näherungsweise angegebene Größe dieses Ortes. Die Größe wird durch den Flächeninhalt in Quadratkilometern dargestellt. Mit diesem Konfigurationsparameter können beliebig viele Flächeninhaltsbereiche festgelegt werden. Orte, die in die entsprechenden Bereiche fallen, können damit differenziert bewertet werden. Beispiel:

  ```
  AreaMap      0/0.1/50/250
  ```

  Die einzelnen Werte definieren die Grenzen der Flächeninhaltsbereiche und legen damit folgende vier Bereiche fest:

  ```
  0    -  0.1  qkm
  0.1  -  50   qkm
  50   -  250  qkm
       >  250  qkm
  ```

Bei der Bewertung gefundener Ortsinformationen spielen drei Faktoren eine Rolle:

- die Genauigkeit (Flächeninhalt) des Ortes
- in welchem Webseitenteil die Ortsinformation gefunden wurde
- wie oft die Ortsinformation auftrat

Für jeden Webseitenteil wird dazu ein eigener Parameter, dessen Parameterwert allerdings jeweils in analoger Weise angegeben wird, verwendet. Diese Parameter werden nachfolgend zusammengefasst vorgestellt.

- *<Webseitenteil>Weights*

  Mit der zusammenfassenden Notation *<Webseitenteil>Weights* werden die acht Parameter der Konfigurationsdatei von *DCbot* bezeichnet, die jeweils die Bewertung von Ortsinformationen, gefunden in den wichtigsten (mit „wichtig“ ist hier „für diese Arbeit relevant“ gemeint) Komponenten von Webseiten, ermöglichen. Dabei dürfen für *<Webseitenteil>* folgende Wörter eingesetzt werden:

  - *Host:* der Teil der URL, der den Webserver identifiziert
  - *Path:* Verzeichnisstruktur auf dem Webserver
  - *Title:* Titel der Webseite

- *Description: description*-Meta-Tag
- *Keywords: keywords*-Meta-Tag
- *Coverage: DC.coverage*-Meta-Tag
- *Body:* Textinhalt der Webseite
- *Anchor:* Verweise der Webseite

Die Bewertung erfolgt durch die Vergabe von Gewichten und von einem Multiplikatorwert, separat für jeden Webseitenteil mittels Parameterwerte der Form:

[*Multiplikator*] - *Gewicht_1* / *Gewicht_2* /... / *Gewicht_n*

Die Basisgewichte 1 bis *n* dienen der Gewichtung von Ortsinformationen, die in die entsprechende Flächeninhaltsbereiche, definiert durch den Konfigurationsparameter *AreaMap*, fallen. Dabei muss *n* immer der Anzahl angegebener Flächeninhaltsbereiche entsprechen. Für das oben gegebene Beispiel also $n = 4$. Diese Basisgewichte dürfen Werte zwischen 1 und 100 einnehmen und können auch als eine Wahrscheinlichkeitsangabe in Prozent aufgefasst werden. Der Multiplikator dient zur Berücksichtigung mehrfach vorkommender Ortsinformationen. Mit seiner Hilfe wird ein Zuschlag berechnet, der im Falle einer mehrfach auftretenden Ortsinformation zum Basisgewicht addiert wird. Der Zuschlag berechnet sich folgendermaßen:

Zuschlag = *Vorkommenshäufigkeit * Multiplikator * Basisgewicht*

Demnach ergibt sich das endgültige Gewicht durch folgende einfache Formel:

Gewicht = min (*Basisgewicht* + *Zuschlag*, 100)

Das Endgewicht darf also den Maximalwert 100 nicht überschreiten, der der hundert prozentigen Wahrscheinlichkeit entspricht.

Die Verwendung der Gewichte soll durch das folgende Beispiel verdeutlicht werden:

```
BodyWeights    [0.5]-75/50/25/0
```

Der Parameter gibt Gewichte für Ortsinformationen, die in dem textuellen Webseiteninhalt gefunden wurden, an. Der Parameterwert (Multiplikator und Gewichte) bezieht sich auf das oben gezeigte Beispiel *AreaMap*. Demnach erhält eine auf der Webseite gefundene Adresse, für die eine Position mit der Genauigkeit von 0.05 qkm ermittelt wurde, das Basisgewicht 75. Diese Ortsinformation fällt offensichtlich in den ersten Flächeninhaltsbereich (0 - 0.1 qkm). Wird das Wort *„Stuttgart“* drei Mal auf der Webseite entdeckt, und ermittelt *DCbot* für diese Ortsangabe die Genauigkeit von 207,34 qkm, dann wird zum Basisgewicht 25 der Zuschlag = 3 * 0.5 * 25 = 37,5 addiert und schließlich das endgültige Gewicht von 62,5 berechnet. Man könnte auch sagen, dass *„Stuttgart“* auf dieser Webseite mit 62,5-prozentiger Wahrscheinlichkeit eine gute Ortsinformation darstellt.

- *WeightThreshold*

  Nach dem Bewertungsvorgang kann die beste Ortsinformation (die mit dem höchsten Gewicht) einfach ermittelt werden. Es bleibt noch die Frage, ob sie nach den Gesichtspunkten des Anwenders gut genug ist? Damit der Webroboter diese Ent-

scheidung treffen kann, besteht die Möglichkeit, mit diesem Konfigurationsparameter einen Schwellwert, der erreicht werden muss, festzulegen. Dementsprechend muss der Wert dieses Parameters zwischen 1 und 100 liegen. Wenn die beste ermittelte Ortsinformation einer Webseite diesen Wert übersteigt, dann wird die Seite gespeichert.

- ForceOverwrite

  Trifft *DCbot* im World Wide Web auf eine Webseite, die während eines früheren Suchvorgangs bereits gespeichert wurde, dann wird diese Seite im *Spatial Model Server* oder in dem Datenbanksystem nur dann aktualisiert, wenn das Datum der letzten Modifikation veraltet ist. Dies ist die Standardeinstellung für diesen Parameter mit dem Wert *„no“*. Sollte eine wieder entdeckte Webseite auf jeden Fall aktualisiert werden, dann sollte man für diesen Parameter den Wert *„yes“* angeben.

3.4.3.3 DCbot *map*-Datei

In der *DCbot map*-Datei können Meta-Tags festgelegt werden, deren Inhalte, wenn vorhanden, zusätzlich gespeichert werden können. Als Standardeinstellung speichert *DCbot* keine Meta-Tag-Daten. Wird ein *Spatial Model Server* für die Speicherung verwendet, dann dürfen nur die folgenden fünf Namen benutzt werden: *„title“*, *„subject“*, *„date“*, *„description“* und *„author“*. Der Grund dafür sind die festen Speicherungsstrukturen, die im *Spatial Model Server* nicht dynamisch verändert werden können. Erfolgt die Speicherung der Ergebnisse in einer herkömmlichen Datenbank, dann dürfen beliebige Meta-Tag-Inhalte für die zusätzliche Speicherung spezifiziert werden. Ausführliche Informationen über die Notation der Einträge der *map*-Datei können in [Süt01] nachgelesen werden.

3.4.3.4 DCbot Regel-Datei

In der *DCbot* Regel-Datei werden die semantischen Regeln des Webroboters gespeichert. Sie werden vom Webroboter während der Suche nach Ortsinformationen in Webseiten selbstständig erlernt. Es ist aber auch möglich, Regeln manuell zur Regel-Datei hinzuzufügen. Im Kapitel 3.4.2.2 wurden die verwendeten Regelarten sowie der Lernprozess vorgestellt. Nachfolgend wird die Syntax der Speicherung erläutert.

Die erste Zeile der Regeldatei hat die Form:

Rulecount: *n*

Sie gibt die Anzahl der jemals durch *DCbot* erlernten Regeln an. Jedes Mal, wenn der Webroboter eine Regel aus einer Webseite extrahiert, wird dieser Zähler um eins erhöht. Sie wird für die Berechnung der relativen Vorkommenshäufigkeit der Regeln verwendet. Hat beispielsweise *DCbot* 200 Regeln erlernt und kam ein bestimmter Re-

gel zehn Mal vor, dann hat diese die absolute Häufigkeit von 10 und die relative Häufigkeit von 0.05 oder 5 Prozent.

Eine Regel der Art

wort <spatial_info> (zum Beispiel „*in <spatial_info>*“)

wird durch die Zeile

[h,wort,<spatial_info>]

dargestellt. Mit *h* wird die relative Häufigkeit einer Regel in Prozent angegeben. Danach folgen Wörter, aus denen die Regel besteht, durch Kommata getrennt. Für das obige Beispiel ergibt sich damit folgende Zeile der Regel-Datei:

[5.0,in,<spatial_info>]

Die relative Häufigkeiten aller Regeln summieren sich sinnvoller Weise auf 100. Dies sollte bei der manuellen Veränderung der Regel-Datei beibehalten werden.

3.4.4 Rückmeldungen

DCbot liefert nach jeder untersuchten Webseite eine kurze Statusmeldung auf das *Standard Out* des Systems, die das Ergebnis der Analyse für die aktuelle Webseite angibt. Diese Meldung hat jeweils die Form:

```
URL: / http://www.domain.com/index.html
     \ Status: OK, Depth: 6, Checked 23 Queued: 258
```

Nach der URL der gerade untersuchten Webseite wird ein Statuscode, die Entfernung der Webseite von der Start-URL (*Depth*), die Anzahl der untersuchten URLs (*Checked*) und die Größe des URL-Puffers (*Queued*) geliefert. Die wichtigsten Statuscodes sind

- *OK*: Webseite wurde erfolgreich untersucht, es konnten aber keine oder keine ausreichend genaue Ortsinformationen gefunden werden.
- OK/SPATIALINFO : In der aktuellen Webseite wurde(n) Ortsinformation(en) gefunden, und die Seite wurde erfolgreich gespeichert.
- OK/UNCHANGED: Die gerade untersuchte Webseite ist bereits in der Datenbasis vorhanden, und ist aktuell.

Die weiteren acht Statuscodes dokumentieren Fälle in denen eine Webseite nicht ordnungsgemäß analysiert werden konnte. Sie können in [Süt01] nachgeschlagen werden.

3.4.5 Ergebnisspeicherung

Hauptergebnisse des Webroboters sind Webseiten mit Ortsinformationen, für die eine ausreichend genaue Position ermittelt werden konnte. Die wichtigsten Informationen, die über diese Webseiten gespeichert werden, sind:

- die URL und der Titel
- die Position in *WGS84* Format
- Daten über die Aktualität der Webseite

Diese Informationen werden entweder in einem *Spatial Model Server* oder in einem Datenbanksystem abgelegt (Konfigurationsparameter *DBInterface*). Über diese Ergebnisse hinaus sammelt *DCbot* während des Suchprozesses statistische Daten, die in einer Statistik-Datei gespeichert werden. Diese Methoden der Ergebnisspeicherung werden in den nachfolgenden Abschnitten beschrieben.

3.4.5.1 Speicherung im *Spatial Model Server*

Ein *Spatial Model Server* ist für die Speicherung von Objekten in der NEXUS Infrastruktur zuständig. Durch die entsprechende Abbildung können in einem solchen Server sowohl Objekte der realen Welt (Straßen, Gebäude, Benutzer, ...) als auch virtuelle Objekte, die der Einbindung externer Datenquellen dienen, modelliert werden. Sie bilden das Weltmodell von NEXUS, das als *Augment World Model* bezeichnet wird. Alle Objekte dieses Weltmodells werden im *Augmented World Class Schema* durch Klassen beschrieben. Die Speicherung von Webseiten mit zugehörigen Positionsdaten wird durch zwei dieser Klassen unterstützt:

- *VirtualWebPortal*

  Virtuelle Web-Portale haben den Zweck, mehrere zu einer Position gehörende Webseiten zusammenzufassen. Dazu hat jedes Web-Portal ein Attribut *pos*, das die Position speichert und eine eindeutige Identifikationsnummer, über die die Verbindung zu gespeicherten Webseiten hergestellt werden kann.

- *WebPage*

  Durch diese Klasse werden Webseiten modelliert. Ein WebPage-Objekt enthält alle, für die Infrastruktur benötigten Informationen (URL, Metadaten, ...), über eine Webseite. Ein spezielles Attribut *partOf* verbindet jede Webseite zu einem virtuellen Web-Portal, das die Position bereithält.

In [Süt01] werden beide Klassen, ihre Attribute und ihre Eingliederung in das *Augmented World Class Schema* detailliert vorgestellt.

3.4.5.2 Speicherung in einem Datenbanksystem

Wie bereits erwähnt können Ergebnisse mittels der *JDBC*-Schnittstelle des Webroboters auch in einem herkömmlichen Datenbanksystem gespeichert werden. Hierfür werden zwei Tabellen, die der im vorherigen Abschnitt vorgestellten Klassen *VirtualWebPortal* und *WebPage* entsprechen, verwendet. Diese Tabellen werden in der Datenbank erstellt, wenn der Webroboter mit der *„initdb“* Option aufgerufen wird. Die Speicherung der Ergebnisse in einem Datenbanksystem hat den Vorteil, dass beliebige Meta-Tag-Inhalte, spezifiziert in der *map*-Datei des Webroboters, mitgespeichert werden können. Dabei werden zu der Tabelle *WebPage* neue Spalten, wenn benötigt (im Falle eines neuen Meta-Tags), dynamisch hinzugefügt. Dieses Vorgehen könnte für die Speicherung in einem *Spatial Model Server* nicht gewählt werden, da die Attribute der Klassen im Vorfeld festgelegt wurden.

3.4.5.3 DCbot Statistik-Datei

Die gespeicherten Ergebnisse des Webroboters sind vereinfacht dargestellt URLs und zugehörige Positionsdaten. Sie sagen nichts über die Art der gefundenen Ortsinformationen oder über die Geschwindigkeit des Webroboters aus. Diese Daten werden zwar für die NEXUS Infrastruktur nicht benötigt, interessieren den Programmierer aber dennoch. Aus diesem Grund wurde zu *DCbot* eine Statistikkomponente hinzugefügt, die diese Fragen beantwortet.

Statistische Daten werden vom Webroboter während der Suche erhoben und in eine Statistik-Datei namens *„DCbot.stats“* geschrieben. Die Aktualisierungshäufigkeit der Daten kann mit dem Konfigurationsparameter *StatisticsUpdateFrequency* in der Konfigurationsdatei des Webroboters eingestellt werden.

Die Statistik-Datei besteht aus vier Sektionen. In der ersten werden Daten, die die Geschwindigkeit des Webroboters beschreiben, gespeichert. Die ersten drei Werte geben den Startzeitpunkt, den Endzeitpunkt und die Gesamtdauer der Suche in folgender Weise an:

```
Start of execution:    Sun  Mar 09 18:30:26 MET 2003
End of execution:      Mon  Mar 10 12:16:26 MET 2003
Execution time:        17 Hours 45 Minutes 59 Seconds 891 Milliseconds
```

Danach folgen die Anzahl der untersuchten Webseiten und die Anzahl der erfolgreich analysierten Webseiten:

```
Number of URLs checked:          25000
Number of web pages analyzed:    15681
```

Detaillierte Informationen werden auch über die wichtigsten Phasen der Webseitenanalyse gegeben. Diese beinhalten unter anderem die Zeiten für das *Parsen* des Dokuments oder für die Berechnung der besten Position aus mehreren möglichen Ortsinformationen.

```
Average time to retrieve document:      614 Milliseconds
Average time to parse document:         544 Milliseconds
Average time to generate candidates:    387 Milliseconds
Average time to calculate best position: 254 Milliseconds
```

Der letzte Wert in dieser Sektion ist wohl der interessanteste. Er gibt die durchschnittliche Zeit, die für die Analyse einer Webseite in diesem Suchlauf benötigt wurde, an.

```
Average time to analyze web page:   2 Seconds 558 Milliseconds
```

Die zweite Sektion der Statistik-Datei listet die Vorkommenshäufigkeiten der einzelnen Statusmeldungen, die Teil der Rückmeldung des Webroboters bilden (siehe Kapitel 3.4.4 auf Seite 51), auf. Beispiel:

```
OK                10413
OK/SPATIALINFO    5268
```

Im dritten Abschnitt sind zwei Tabellen zu finden, die die Anzahl während der Analyse generierten Kandidaten und tatsächlich gefundenen Ortsinformationen detailliert aufschlüsseln. Die Spalten der Tabellen stellen die Analysemethoden dar, während Webseitenkomponenten durch Zeilen repräsentiert werden. Damit ergibt sich folgende (hier abgekürzte) Beispieldarstellung:

	address	...	rule	...	other
host	X	...	X	...	20
...					
title	13	...	421	...	383
...					
body	567	...	187	...	672

Die einzelnen Spalten (für detaillierte Beschreibung der Analysemethoden siehe Kapitel 3.4.2.2) sind:

- *address:* Kandidat oder Ortsinformation wurde durch Adressenerkennung gefunden.
- *specialword:* Kandidat oder Ortsinformation wurde durch die Verwendung eines Schlüsselwortes aus der *DCbot* Konfigurationsdatei gefunden.
- *rule:* Für die Generierung des Kandidaten beziehungsweise für die Erkennung der Ortsinformation wurde ein semantischer Regel verwendet.
- *firstnword:* Kandidat oder Ortsinformation wurde unter den ersten, in der Konfigurationsdatei als wichtig definierten, Wörter eines Webseitenteils gefunden.
- *mostfrequent:* Diese Methode wird momentan nur auf den textuellen Inhalt (*body*) einer Webseite angewendet. Sie sucht Kandidaten beziehungsweise Ortsinformationen unter den häufigsten Wörtern einer Webseite.

- *other:* In manchen Webseiten mit relativ wenig Textinhalt werden alle Wörter untersucht. Findet der Webroboter einen Kandidaten oder schließlich eine Ortsinformation ohne die Verwendung einer bisher vorgestellten Methode, dann werden diese der Spalte *„other“* zugeordnet.

Wie bereits erwähnt stellen die Zeilen der Tabellen die einzelnen Webseitenkomponenten dar, in denen der Webroboter nach Ortsinformationen sucht. Diese Komponenten werden kurz in Kapitel 3.4.3.2 auf Seite 43 unter dem Konfigurationsparameter <*Webseitenteil*>*Weights* (Beschreibung der Gewichtevergabe) aufgelistet, und in Kapitel 2.4 auf Seite 18 ausführlich beschrieben. Daher ist eine Wiederholung hier nicht notwendig.

Aus den vorgestellten Tabellen kann der Anwender schnell herauslesen, in welchem Webseitenteil mit welcher Methode wie viele Kandidaten generiert beziehungsweise wie viele Ortsinformationen gefunden wurden. Wird eine Methode zur Analyse eines Webseitenteils nicht herangezogen, dann steht statt der Anzahl das Symbol *„X“* an der entsprechenden Stelle der Tabelle.

Die vierte und letzte Sektion der Statistik-Datei listet gefundene Ortsinformationen nach Ortstypen auf. Die Ortstypen werden nach der Vorkommenshäufigkeit sortiert aufgeführt. Beispiel:

```
city        586
university  27
lake        14
```

Diese Liste bedeutet, dass unter den gefundenen Ortsinformationen 586 Städtenamen, 27 Universitäten und 14 Seenamen auftraten.

3.5 Implementierungsdetails

In der Studienarbeit des Autors [Süt01] wurden bereits zahlreiche Implementierungsaspekte des Webroboters beschrieben. Nach einem kurzen Systemüberblick werden aus diesem Grund nur neu implementierte Komponenten vorgestellt. Da der Webroboter in erster Linie um neue Funktionen erweitert wurde, mussten alte Komponenten nur in seltenen Fällen modifiziert werden. Abbildung 3-2 gibt einen Übersicht über die wichtigsten Klassen des Webroboters und stellt ihre Zusammenarbeit dar.

Mit Hilfe der Klasse *ConfigLineParser* werden am Anfang eines Suchvorganges die Konfigurationsdatei und die *map*-Datei von *DCbot* eingelesen. Die Klasse *RuleFileHandler* stellt *DCbot* am Anfang eines Suchlaufes die, in der Regel-Datei vorhandenen, semantischen Regeln zu Verfügung. Sie übernimmt auch die Speicherung von Regeln, die während des Suchlaufes erlernt wurden. Über den *HTTPClient*, der die Schnittstelle zum World Wide Web bildet, erhält der Webroboter die benötigten Webseiten. Die Klasse *RobotExclusionHandler* sorgt dafür, dass nur Webseiten untersucht werden, die eine Analyse durch einen Roboter auch zulassen. *HTMLHandler* ist eine der wichtigsten und umfangreichsten Klassen, die *DCbot* zu bieten hat. Sie übern-

immt die Analyse der Webseiten und verwendet dabei die Pakete *sun.net.www.html* und *rdf* ebenso wie die Klassen *Rule*, *Candidate* und *Address*.

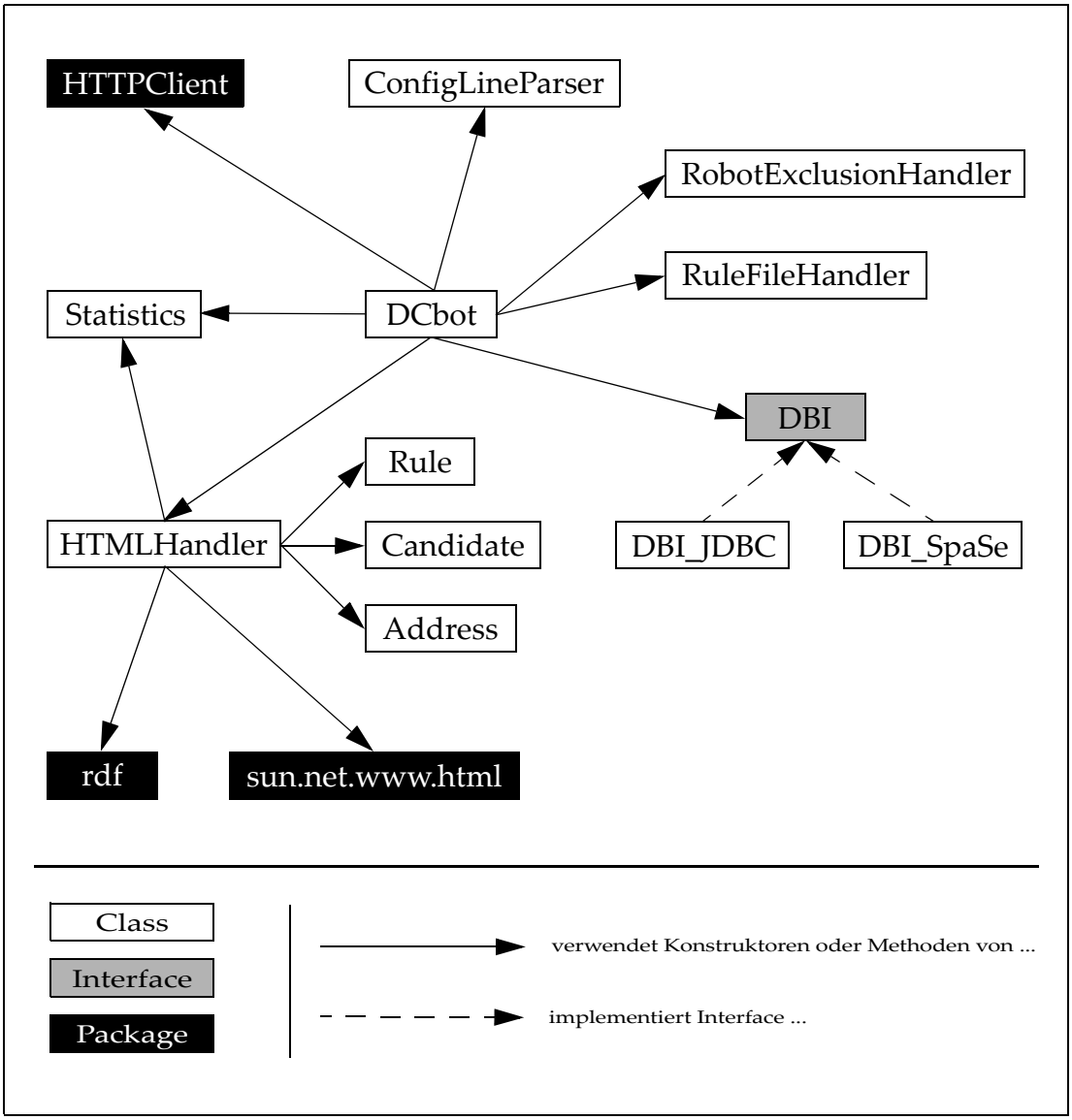

Abbildung 3-2: Klassenübersicht von *DCbot*

Das Paket *sun.net.www.html* übernimmt das Parsen des *HTML*-Dokumentes während das Paket *rdf* eine eventuell vorhandene externe Metadaten-Datei (*XML/RDF*-Format) interpretiert. Die Klasse *Rule* modelliert eine semantische Regel, die Klasse Candidate setzt die Speicherung generierter Ortsbezugskandidaten um, während die Klasse *Address* die effektive Speicherung von Adressen unterstützt.

Die *DBI_JDBC*-Schnittstelle ermöglicht *DCbot* sowohl den Zugriff auf den Katalog *GeoBase* als auch die Speicherung der Ergebnisse in einem Datenbanksystem. Mit Hilfe der *DBI_SpaSe*-Schnittstelle können diese Ergebnisse aber auch in einem *Spatial Model Server* abgelegt werden.

Während der gesamten Laufzeit des Webroboters verwenden die Klassen *DCbot* und *HTMLHandler* die Klasse *Statistics*, um statistische Daten über den Analysevorgang in der Statistik-Datei zu speichern.

3.5.1 Allgemeine Vorgehensweise

Während der Implementierung der Erweiterungen des Webroboters wurde stetig darauf geachtet, dass nur effiziente Speicherungsstrukturen und Algorithmen verwendet werden.

In den meisten Fällen wurden Vektoren für die Speicherung von Daten benutzt. Vektoren haben den Vorteil, dass die einzelnen Elemente auf direktem Weg durch den Index (Zeitkomplexität für den Zugriff: *O(1)*) erreicht werden können. Zudem kann die Größe eines Vektors dynamisch angepasst werden, so dass nur tatsächlich benötigter Speicherplatz allokiert werden muss. War eine Sortierung der Daten in einem Vektor notwendig, so wurde ein Insert-Sort-Verfahren (Zeitkomplexität: *O(n*(log n))*) verwendet, das für die Sortierung einen neuen Vektor aufbaut und den richtigen Platz eines neuen Elementes durch Binärsuche (Zeitkomplexität: *O(log n)*) feststellt.

Oft mussten Elemente zu einer sortierten Menge hinzugefügt werden. In diesen Fällen wurde die richtige Stelle für das Einfügen ebenfalls, wie bei der Sortierung, durch eine Binärsuche festgestellt.

Mussten Elemente aus einer größeren Menge schnell ausgewählt werden, und konnten keine Indexe benutzt werden, so wurden die Daten in einer Hashtabelle gespeichert, die ebenfalls einen direkten Zugriff (Zeitkomplexität: *O(1)*) auf die Elemente gestattet.

Nachfolgend werden einige Klassen beschrieben, die den Großteil der implementierten Erweiterungen darstellen. Der Zugriff auf die Felder dieser Klassen (wenn von anderen Methoden erwünscht) ist immer in analoger Weise möglich. Lesezugriff durch Methoden der Form

```
get<Feldname>()
```

und Schreibzugriff durch Methoden der Form

```
set<Feldname>(Feldwert)
```

Diese Selektoren werden bei der Beschreibung der Klassen wegen ihrer Einfachheit nicht aufgelistet.

3.5.2 Die Klasse *Candidate*

Wie in Kapitel 3.4.2.2 beschrieben, generiert *DCbot* erstmal Ortsinformationskandidaten aus Webseiten, um sie zu einem späteren Zeitpunkt mit dem Katalog *GeoBase* auf ihre Richtigkeit hin zu überprüfen. Ein Kandidat wird durch die Klasse *Candidate* implementiert. die wichtigsten Felder dieser Klasse sind:

- *text:* Dieses Feld enthält die *String*-Darstellung des Kandidaten.
- *found_in:* Aus diesem Webseitenteil heraus wurde der Kandidat generiert.

- *indexes:* Dies sind die Positionen (als Zahlenvektor gespeichert), an denen der Kandidat im Webseitenteil (Dargestellt durch einen Wortvektor) gefunden wurde. Eine Ortsinformation kann selbstverständlich mehrmals in einem Webseitenteil auftreten.

Während der Überprüfung eines Kandidaten mit dem Katalog können für diesen die Position (*WGS84* Format), die Genauigkeit (Flächeninhalt) und der Ortstyp aus *GeoBase* ermittelt werden. Diese Informationen werden ebenfalls von *Candidate*-Objekten getragen. Hierfür besitzt die Klasse folgenden drei Felder:

- *position:* Die ermittelte Position eines Kandidaten.
- *area:* Der ermittelte Flächeninhalt eines Kandidaten.
- *type:* Der ermittelte Ortstyp eines Kandidaten.

Der am häufigsten verwendete Konstruktor der Klasse

```
Candidate ( String spatial_information, int number_of_tokens ,
            String found_in, int index )
```

erwartet an erster Stelle die *String*-Darstellung des Kandidaten, dann die Anzahl der Wörter aus diesem *String*, den Namen des betroffenen Webseitenteils und schließlich die Position des Kandidaten in dem entsprechenden Webseitenteil.

Die Position im Webseitenteil (*index*) und die Anzahl der Wörter eines Kandidaten (*number_of_tokens*) werden benötigt, damit Wörter effizient aus seiner Umgebung (Wortkontext) gewonnen werden können. Dies geschieht zum Beispiel beim Erlernen der semantischen Regeln von *DCbot*. Der Name des Webseitenteils (*found_in*) wird gespeichert, damit Ortsinformationen später abhängig vom „Fundort“ bewertet werden können.

Die Klasse *Candidate* stellt eine Methode zur Verfügung, die den Vergleich zweier Kandidaten ermöglicht. Die Methode

```
compareTo(Candidate candidate, String criterion)
```

wird direkt am ersten zu vergleichenden Objekt aufgerufen und erhält als ersten Parameter das zweite Objekt. Zusätzlich muss durch den zweiten Parameter (*criterion*) bestimmt werden, nach welchem Gesichtspunkt die beiden Kandidaten verglichen werden sollten. Dieser Parameter kann den Wert *„text“*, Vergleich nach der *String*-Darstellung der Kandidaten, oder den Wert *„count“*, Vergleich nach der Vorkommenshäufigkeit der Kandidaten, annehmen.

Ein Vergleich zweier Kandidaten wird immer dann benötigt, wenn ein neuer Kandidat in eine bereits sortierte Menge von Kandidaten an der richtigen Stelle eingefügt werden muss. Dieses Verfahren wird durch die Methode

```
addNewCandidate(Vector candidates, Candidate candidate, String criterion)
```

implementiert. Sie erwartet an erster Stelle einen sortierten Kandidatenvektor, dann den Kandidaten, der hinzugefügt werden soll, und schließlich an dritter Stelle die Angabe, nach welchem Gesichtspunkt (*„text“* oder *„count“*) der Kandidatenvektor sortiert ist.

3.5.3 Die Klasse *Address*

Die Klasse *Address* implementiert, wie der Name schon verrät, eine postalische Adresse, so wie sie in Deutschland verwendet wird. Sie erlaubt die effiziente Speicherung einer Adresse sowie den einfachen Zugriff auf ihre Komponenten. Die wichtigsten Felder dieser Klasse sind:

- *street:* Dieses Feld speichert die Straßenkomponente der Adresse.
- *housenumberstart:* Wenn die Hausnummer ein Nummerbereich ist (zum Beispiel *„20-22“*), dann enthält dieses Feld die untere Grenze dieses Bereiches. Im Falle einer einfachen Nummer, wie zum Beispiel *„78a“* speichert dieses Feld die gesamte Hausnummer.
- *housenumberend:* Wenn die Hausnummer ein Nummerbereich ist, dann enthält dieses Feld die obere Grenze dieses Bereiches, sonst den Wert *„null“*.
- *postalcode:* Dieses Feld speichert eine Postleitzahl. In Deutschland werden hierfür zurzeit fünfstellige Zahlen verwendet.
- *city:* In diesem Feld wird der Name der Stadt gespeichert.
- *address_string:* Dies ist die *String*-Darstellung der kompletten Adresse.

Der einzige Konstruktor der Klasse

```
Address ( String street, String housenumberstart, String housenumberend,
          String postalcode, String city)
```

erwartet die einzelnen vorhin vorgestellten Komponenten in *String*-Darstellung. Die Extrahierung von Adressen aus Webseitenkomponenten übernimmt die Methode

```
checkForAddress(Vector tokens, String found_in, int index, Vector candidates).
```

Diese Methode wird einmal für jedes Wort eines Webseitenteils aufgerufen. Übergabeparameter sind:

- *tokens:* Dieser Vektor enthält alle Wörter eines Webseitenteils. Die einzelnen Wörter sind *Strings*, und sind als Elemente des Vektors gespeichert.
- *found_in:* Der Name des Webseitenteils, der gerade untersucht wird.
- *index:* Eine Zahl, die die Position des gerade untersuchten Wortes angibt.
- *candidates:* Ein Vektor, der bisher generierte Kandidaten enthält.

Durch die Methoden isStreet(...) und isPostalCode(...) wird überprüft, ob das jeweilige Wort ein Straßenname oder eine Postleitzahl ist. Ist dies der Fall, dann versucht die Methode aus der Wortumgebung eine syntaktisch korrekte Adresse zu extrahieren. Hierbei wird zusätzlich die Methode isHouseNumber(...), die die Gültigkeit von Hausnummern überprüft, verwendet.

3.5.4 Die Klasse *Rule*

Die Klasse *Rule* implementiert eine semantische Regel, wie sie in Kapitel 3.4.2.2 vorgestellt wurde. Semantische Regeln speichern den Kontext bereits gefundener Ortsinformationen und werden von *DCbot* verwendet um weitere Ortsinformationen in großen Webseitenteilen ausfindig zu machen. Die wichtigsten Felder der Klasse sind:

- *frequency:* Dieses Feld speichert die relative Vorkommenshäufigkeit der Regel als *Float*-Zahl. Je größer diese Zahl, desto besser ist die Regel.
- *sortstring:* In diesem Feld wird die *String*-Darstellung der Regel gespeichert. Wie der Name des Feldes verrät, wird diese Darstellung verwendet, um Regeln in alphabetischer Reihenfolge zu ordnen.
- *firstword, secondword, thirdword:* Eine Regel besteht aus maximal drei Wörtern. Diese werden in diesen Feldern auch einzeln gespeichert, damit ein schneller Zugriff jederzeit gewährleistet ist.

Die zwei Konstruktoren der Klasse unterscheiden sich lediglich durch die Angabe des dritten Wortes einer Regel. Besteht die Regel aus nur zwei Wörtern, dann wird der Konstruktor

```
Rule(String frequency, String firstword, String secondword)
```

sonst der Konstruktor

```
Rule(String frequency, String firstword, String secondword, String thirdword)
```

verwendet. Wird eine neue Regel erstellt, dann enthält das *frequency*-Feld erstmal die Zahl *eins*. Dies ist die momentane absolute Häufigkeit. Wird die Regel mehrmals zwischen zwei Regel-„*Updates*“ (siehe Konfigurationsparameter *RuleUpdateFrequency*) erlernt, dann wird die absolute Häufigkeit der Regel dementsprechend hochgezählt. Bei einem Regel-„*Update*“ wird festgestellt, ob die Regel bereits im aktiven Regel-„*Set*“ vorhanden war, und die relative Häufigkeit aller neu erlernten Regeln berechnet.

Die Klasse *Rule* implementiert einen einfachen abstrakten Datentyp. Aus diesem Grund werden in dieser Klasse keine besonderen Methoden benötigt. Neben den obligatorischen Selektoren, die den Zugriff auf die Felder der Klasse erlauben, gibt es nur eine weitere Methode, die den Vergleich zweier Regeln ermöglicht.

```
compareTo(Rule rule, String criterion)
```

Ähnlich wie Kandidaten (Klasse *Candidate*) können Regeln alphabetisch (*„sortstring“*) oder nach der relativen Vorkommenshäufigkeit (*„frequency“*) sortiert werden. Die entsprechenden Werte müssen an zweiter Stelle (Parameter *criterion*) der Vergleichsmethode übergeben werden.

3.5.5 Die Klasse *RuleFileHandler*

Die Klasse *RuleFileHandler* implementiert die gesamte Verwaltung der semantischen Regeln. Darunter werden das Erlernen der Regeln, ihre Speicherung und der effiziente Zugriff auf die Regeln verstanden. Die wichtigsten Felder der Klasse sind:

- *rules:* Dies ist das aktive Regel-*„Set“* des Webroboters. Aus dieser Menge können Regeln jederzeit vom *DCbot* verwendet werden, vorausgesetzt, sie haben eine entsprechend gute relative Häufigkeit (Konfigurationsparameter *GoodRuleThreshold*). Dieses Feld enthält einen Vektor, dessen Elemente Regeln (Instanzen der Klasse *Rule*) sind.
- *newrules:* Erlernte Regeln werden erstmal in diesem Vektor gespeichert. Erfolgt ein Regel-*„Update“*, dann werden diese Regeln zu dem aktiven Regel-*„Set“* hinzugefügt.
- *rulecount:* Die Anzahl jemals von *DCbot* erlernter Regeln.
- *newrulecount:* Die Anzahl der, seit dem letzten Regel-*„Update“* erlernten, Regeln.
- *firstword_index_vectors:* Diese Hashtabelle enthält die Indexe derjenigen Regeln, die eine ausreichend gute relative Vorkommenshäufigkeit aufweisen. Regeln werden zusätzlich nach ihrem ersten Wort (Feld *firstword*) gruppiert. Die *Schlüssel* der Hashtabelleneinträge sind *Strings* (erste Wörter der Regeln), die *Werte* sind Vektoren, die Indexe (Position der Regel im aktiven *„Set“*) enthalten. Der Aufbau dieser Hashtabelle wird in Abbildung 3-3 graphisch dargestellt.

Der meistverwendete Konstruktor der Klasse

```
RuleFileHandler(String filename)
```

erwartet einen einzigen Parameterwert, den Namen der Regel-Datei. Unter der Verwendung weiterer Konstruktoren liest er alle Regeln aus der spezifizierten Regel-Datei ein, speichert sie in einem Vektor (*rules*) und baut die Hastabelle (*firstword_index_vectors*) für den schnellen Zugriff auf die Regeln auf. Den Aufbau dieser Hashtabelle übernimmt die Methode

```
buildRuleHashtable()
```

Sie geht die Liste der Regeln durch und überprüft, ob die gerade untersuchte Regel eine ausreichende relative Häufigkeit (*frequency*) besitzt. Ist dies der Fall, dann wird der Index der Regel in der Hashtabelle gespeichert (siehe Abbildung 3-3). Mit dieser Speicherungsstruktur kann sehr schnell (Zeitaufwand: *O(1)*) ermittelt werden, ob zu

einem bestimmten Wort passende Regeln im aktiven Regel-*„Set“* vorhanden sind. Der Aufbau der Hashtabelle erfordert linearen Zeitaufwand.

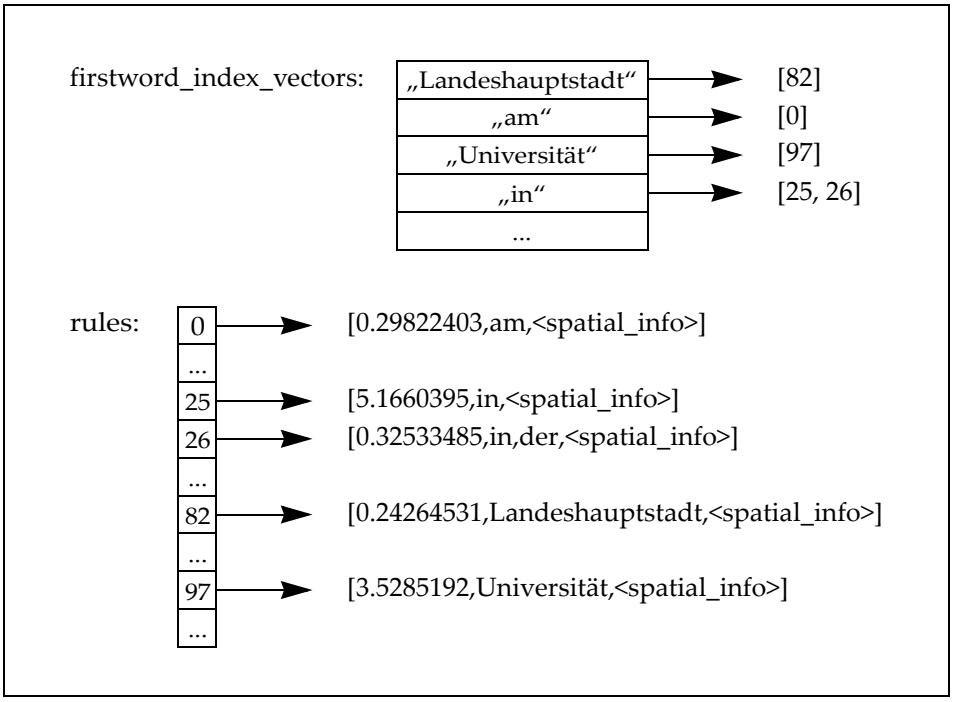

Abbildung 3-3: Darstellung zweier Speicherungsstrukturen

Das Feld *newrules* enthält neu erlernte Regeln, die vom Webroboter jedes Mal nach dem Auffinden einer Ortsinformation auf einer Webseite aus dem Textkontext extrahiert werden. Die „neuen“ Regeln sind alphabetisch geordnet. Die Methode

```
addNewRule(Rule rule)
```

fügt zu dieser Liste eine neue Regel hinzu. Die richtige Stelle in dem sortierten Vektor wird mit einem Binärsuchverfahren festgestellt. Der Zeitaufwand für die Einfügeoperation ist logarithmisch (*O(log n)*).

Das Erlernen (Extrahieren) von neuen Regeln wird von der Methode

```
learnNewRules ( Vector spatialinfo, String pagepart_name,
                Vector tokenizedpagepart )
```

übernommen. Sie erwartet die Liste gefundener Ortsinformationen (*spatialinfo*), den Namen der Webseitenkomponente (*pagepart_name*), aus dem Regeln extrahiert werden sollen, und den Wortvektor (*tokenizedpagepart*) dieses Webseitenteils. Für jede Ortsinformation werden die drei in Kapitel 3.4.2.2 vorgestellten Regelarten ermittelt und als „neue“ Regeln gespeichert.

Das aktive Regel-*„Set“* (*rules*) und die Liste der neu erlernten Regeln (*newrules*) werden intern alphabetisch geordnet gespeichert. Während eines Regel-*„Update“* werden alle Regeln in die Regel-Datei aufgenommen. Damit der Anwender die Regeln auf einfache Art einsehen kann, wird das Regel-*„Set“* für die Externspeicherung nach der

relativen Vorkommenshäufigkeit (*frequency*) sortiert. Die Sortierung des Regel-*„Set“* kann mit der Methode

```
sortRules(String criterion)
```

durchgeführt werden. Sie erwartet als Parameter die gewünschte Art der Sortierung (*„frequency“* oder *„sortstring“*).

Das mehrmals erwähnte Regel-*„Update“* wird von der Methode

```
updateRules()
```

ausgeführt. Zuerst werden die absoluten Häufigkeiten der „neuen“ Regeln mit der Methode normalizeNewRuleFrequencies() in relative Häufigkeiten umgerechnet. Durch das Hinzufügen neuer Regeln zum aktiven Regel-*„Set“* verändert sich die Gesamtregelanzahl. Aus diesem Grund werden mit der Methode updateOldRuleFrequencies() die relativen Häufigkeiten der „alten“ Regeln auf den neuesten Stand gebracht. Danach können die „neuen“ Regeln zum aktiven Regel-*„Set“* hinzugefügt werden. Ergeben sich dadurch mehr Regeln als gewünscht (siehe Konfigurationsparameter *MaxRules*), dann werden die überschüssigen Regeln entfernt. Die Methode updateRuleFile() sorgt dafür, dass die aktualisierten Regeln in die Regel-Datei geschrieben werden und die Methode buildRuleHashtable() berechnet eine ebenfalls aktualisierte Hashtabelle für das Feld *firstword_index_vectors.*

3.5.6 Die Klasse *DBI_JDBC*

Die *DBI_JDBC* Klasse implementiert die Schnittstelle des Webroboters zu einem herkömmlichen Datenbanksystem, das mittels eines *JDBC*-Treibers angesprochen werden kann. Diese Schnittstelle wurde in zwei Bereichen, die nachfolgend beschrieben werden, erweitert, beziehungsweise modifiziert.

3.5.6.1 Speicherungsstrukturen

In der Studienarbeit des Autors [Süt01] wurden bereits Strukturen für die Speicherung der Ergebnisse des Webroboters in einem *Spatial Model Server* vorgestellt. Die Speicherung dieser Daten in einem Datenbanksystem hatte allerdings nur vorläufigen Charakter und diente in erster Linie der Auswertung von Testdaten. Dies wurde nun dahingehend geändert, dass für die Speicherung der Daten in einem Datenbanksystem die gleichen, vollständigen Datenstrukturen des *Spatial Model Servers*, wie in Kapitel 3.4.5.2 auf Seite 53 vorgestellt, verwendet werden. Methoden, die von dieser Änderung maßgeblich betroffen sind oder neu erstellt wurden, werden nachfolgend erläutert.

- createTables()

  Wenn der Webroboter mit der Option *„initdb“* aufgerufen wird, dann kommt diese Methode zum Einsatz. Sie legt zwei Tabellen in der spezifizierten Datenbank an.

Eine für die Speicherung von Metadaten (URL, Titel, Meta-Tags, ...) über Webseiten und eine für die Speicherung der virtuellen Web-Portale, die die Positionsdaten enthalten.

- dropTables()

  Sollten die gespeicherten Daten nicht mehr benötigt werden, dann kann der Webroboter mit der Option *„deletedb“* aufgerufen werden. In diesem Fall wird diese Methode zur Entfernung der, durch die Methode createTables() erzeugten, Tabellen verwendet.

- insertVirtualWebPortal(WGS84Point position)

  Mit dieser Methode kann ein neues, virtuelles Web-Portal erstellt werden. Als einziger Parameter wird die Position des Web-Portals als ein *WGS84Point*-Objekt erwartet. Die Klasse *WGS84Point* implementiert eine Positionsangabe nach dem *WGS84* Format.

- getVirtualWebPortal(WGS84Point position)

  Diese Methode überprüft, ob zu einer bestimmten Position (Parameter *position*) ein entsprechendes virtuelles Web-Portal schon in der Datenbank vorhanden ist. Ist dies der Fall, dann wird das *id* des Web-Portals in der *String*-Darstellung zurückgeliefert. Dieses *id* wird für jedes Web-Portal bei der Erzeugung automatisch generiert. Ist das gesuchte Web-Portal in der Datenbank nicht vorhanden, dann wird der Wert *„null“* geliefert.

- setPosition(String primarykey, WGS84Point position)

  Sind alle Metadaten über eine Webseite (mit der URL *primarykey*) gespeichert, muss die ermittelte Position (Parameter *position*) in der Datenbank vermerkt werden. Diese Methode überprüft mit Hilfe der Methode getVirtualWebPortal(...) ob ein entsprechendes virtuelles Web-Portal in der Datenbank schon existiert. Wenn nicht, wird es mit der Methode insertVirtualWebPortal(...) erstellt. In beiden Fällen wird das *id* des virtuellen Web-Portals in dem Attribut *partOf* der Webseite vermerkt. Durch diese 1:n-Beziehung kann ein Web-Portal beliebig viele Webseiten um eine Position zusammenfassen.

- cleanVirtualWebPortal()

  Wenn eine Webseite im World Wide Web nicht mehr erreichbar ist, wird sie aus der Datenbank entfernt. Dadurch können virtuelle Web-Portale, die von keiner Webseite referenziert werden, in der Datenbank zurückbleiben. Aus Effizienzgründen werden diese nicht jedes Mal, wenn eine Webseite gelöscht wird, ermittelt, sondern immer nur am Ende eines Suchlaufes. Die Entfernung nicht benötigter virtueller Web-Portale wird mit Hilfe dieser Methode durchgeführt.

3.5.6.2 Zugriff auf den Katalog GeoBase

Ein völlig neuer Aspekt, um den *DCbot* erweitert wurde, ist der Zugriff auf den Katalog *GeoBase* (siehe Kapitel 2.3.3 auf Seite 15), damit aus Webseiten extrahierte Ortsin-

formationen verifiziert werden können. Zu diesem Zweck wurde die *DBI_JDBC*-Schnittstelle des Webroboters um die nachfolgend vorgestellten Methoden erweitert.

- checkNames(Vector candidates)

  Dieser Methode wird ein Vektor (Parameter *candidates*) übergeben, der aus Webseiten extrahierte Ortsinformationskandidaten enthält. Diese möglichen Ortsnamen werden mit dem Katalog verglichen. Dazu wird eine einzige *SQL*-Anfrage (*INTERSECT*) verwendet. Diese Methode liefert einen Vektor mit verifizierten, also in dem Katalog gefundenen, Ortsnamen.

- getPositionsForSite(String sitename)

  Wurden verifizierte Ortsnamen mit der Methode checkNames(...) ermittelt, dann können mit dieser Methode die zugehörigen Metadaten aus *GeoBase* herausgelesen werden. Dabei ist zu beachten, dass ein Ortsname (Parameter *sitename*) mehrere geographische Stätten identifizieren kann, und dass möglicher Weise mehrere Ergebnisse geliefert werden können. Für jedes Ergebnis wird ein eigenes *Candidate*-Objekt erzeugt. In diesem Objekt werden folgenden Daten gespeichert:

  - *position:* Die exakte Position (gekennzeichnet durch Längengrad und Breitengrad) wird aus *GeoBase* ermittelt und ein *WGS84Point*-Objekt, das diese Informationen trägt, erzeugt.
  - *area:* Der Flächeninhalt (in Quadratkilometern), der die Genauigkeit einer Ortsangabe darstellt, wird ermittelt.
  - *placetype:* Als letztes wird der Ortstyp für den gegebenen Ortsnamen aus *GeoBase* herausgelesen.

  Alle erzeugten *Candidate*-Objekte werden in einem Vektor gespeichert und dieser Vektor wird als Rückgabewert der Methode geliefert.

- getPositionsForSiteWithParent(String sitename, String parentname)

  Diese Methode arbeitet analog zur Methode getPositionsForSite(...), mit dem Unterschied, dass *Candidate*-Objekte nur für Orte, mit dem passenden, in der Hierarchie übergeordneten, Ort (Parameter *parentname*) erzeugt werden. Die hierarchische Struktur des Katalogs wird in erster Linie bei der Adressenerkennung verwendet.

3.5.7 Die Klasse *HTMLHandler*

Diese Klasse implementiert die Analysekomponente des Webroboters, die den Großteil bisher vorgestellter Klassen und Methoden direkt oder indirekt verwendet. Die grundlegende Vorgehensweise bei der Analyse einer Webseite ist die Zwischenspeicherung aller zu untersuchenden Webseitenteile, die Generierung von möglichen Ortsinformationskandidaten, die Überprüfung dieser Kandidaten mit dem Katalog *GeoBase*, die Ermittlung der besten Position und schließlich (wenn erforderlich) die Speicherung der Webseite. Diese einzelnen Schritte werden in der Regel jeweils durch eine eigene Methode, die meistens weitere Hilfsmethoden zur Durchführung kleine-

rer Teilaufgaben verwenden, umgesetzt. Diese Hauptmethoden werden nachfolgend kurz vorgestellt.

- setDocParts(InputStream docstream)

  Diese Methode ruft den *HTMLParser* des Webroboters auf und speichert die wichtigsten Webseitenteile in einer *String*-Darstellung. Dazu gehören zum Beispiel der Textinhalt, der Titel oder die Verweise der Webseite. Meta-Tags oder Informationen aus einer *XML/RDF*-Metadatendatei werden erst zu einem späteren Zeitpunkt durch die Methode setAllTags() in *METATag*-Objekten gespeichert.

- tokenizeParts()

  Diese Methode zerlegt die bisher als *Strings* gespeicherten Webseitenteile in einzelne Wörter und speichert diese in Wortvektoren. Ein Wortvektor ist ein Vektor, der einzelnen Wörter eines Webseitenteils als Elemente trägt. Diese Darstellung wurde gewählt, da durch den Index des Vektors ein schneller Zugriff auf die einzelnen Wörter des Webseitenteils gewährleistet wird. Eine Iteration über alle Wörter kann mit einer einfachen Schleife realisiert werden. Während des Zerlegungsprozesses wird die Methode tokenizeElement(...), die die Zerlegung eines einzelnen Webseitenteils erledigt, oft verwendet.

- generateAllCandidates()

  Diese Methode bildet den Rahmen für die Extrahierung von Ortsinformationskandidaten aus den verschiedenen Webseitenteilen. Im Prinzip wird für jedes Webseitenteil die Methode generateCandidateVector(...) aufgerufen.

- generateCandidateVector(Vector tokens, String found_in)

  Diese Methode erwartet als ersten Parameter den Wortvektor des zu untersuchenden Webseitenteils. Als zweiten Parameter dann dessen Namen. Durch diese Methode werden im Prinzip die, in Kapitel 3.4.2.2 vorgestellten, Analyseverfahren umgesetzt. Wichtige Methoden, die dabei verwendet werden sind:

  - checkSpecialWords(...)

    In der *DCbot*-Konfigurationsdatei kann der Anwender Schüsselwörter angeben, die die Erkennung natürlicher geographischen Stätten (Konfigurationsparameter *GeoSiteIndicators*) und Gebäudenamen (Konfigurationsparameter *StructureIndicators*) ermöglichen. Diese Methode verwendet solche Schlüsselwörter, um Ortsinformationen in den Webseitenteilen zu finden. Dabei wird jedes Wort einer Webseitenkomponente mit den Schlüsselwörtern verglichen. Um dies effizient durchführen zu können, werden die Schlüsselwörter in Hashtabellen gehalten.

  - checkRules(...)

    Diese Methode wendet die semantischen Regeln des Webroboters auf die einzelne Webseitenteilen an. Dabei wird für jedes Wort einer Webseitenkomponente überprüft, ob eine passende Regel in dem aktiven Regel-*„Set“* des Webroboters vorhanden ist. Zu diesem Zweck werden effiziente Zugriffstrukturen (siehe *firstword_index_vectors* in Kapitel 3.5.5 auf Seite 61) auf das Regel-*„Set“* verwendet.

  - checkForAddress(...)

    Diese Methode realisiert die Adressenerkennungskomponente des Webroboters und wurde in Kapitel 3.5.3 auf Seite 59 bereits vorgestellt.

  - getMostFrequentWords(Vector vector, int n)

    Mit dieser Methode werden die *n* häufigsten Wörter eines Webseitenteils berechnet. Bei der Ermittlung dieser Wörter werden nur mögliche Eigennamen (im Deutschen groß geschrieben) beachtet.

- setPositions(DBI db)

  Diese Methode erwartet als Parameter die Speicherungsschnittstelle (z.B. eine Instanz der Klasse *DBI_JDBC*) des Webroboters. Sie überprüft mit der Methode checkNames(...), welche Kandidaten tatsächlich Ortsinformationen darstellen. Danach werden die Positionen für die verifizierten Kandidaten mit Hilfe der Methode getPositionsForSite(...) aus *GeoBase* ermittelt. Kommen unter den Kandidaten potenzielle Adressen vor, werden die zugehörenden Positionen mit der Methode getPositionForAddress(...) berechnet.

- getPositionForAddress(Candidate candidate, DBI db)

  Die Ermittlung der Position für eine Adresse gestaltet sich umfangreicher als die relativ einfache Überprüfung eines Ortsnamens mit dem Katalog. Wichtige Komponenten einer Adresse für die Berechnung der bestmöglichen Position sind der Straßenname, die Postleitzahl und der Name der Stadt. Jede dieser Komponenten kann eventuell auf der Webseite falsch geschrieben sein, oder gar nicht in *GeoBase* vorkommen. Die jeweils beste Position für eine Adresse wird mit Hilfe der Methode getPositionsForSiteWithParent(...), die die hierarchische Ordnung des Kataloges ausnutzt, ermittelt. Die genauere Vorgehensweise, die durch diese Methode implementiert ist, wurde in Kapitel 3.4.2.2 bereits beschrieben.

- getBestSpatialInfo()

  Nachdem für alle gefundenen Ortsinformationen die Positionen ermittelt wurden, kann mit dieser Methode die beste Position und damit die beste Ortsinformation für eine Webseite ermittelt werden. Für die Bewertung der Ortsinformationen benutzt der Webroboter die, vom Benutzter in der Konfigurationsdatei spezifizierten, Gewichte (siehe Konfigurationsparameter *<Webseitenteil>Weights* in Kapitel 3.4.3.2 auf Seite 43). Mit Hilfe dieser differenzierten Bewertung wird für jedes Webseitenteil und schließlich für die gesamte Webseite die beste Position berechnet und als Rückgabewert der Methode geliefert. Da dieser Methode alle, aus einer Webseite extrahierten, Ortsinformationen zur Verfügung stehen, bietet sich an dieser Stelle an, die semantischen Regeln zu erlernen. Für jede Ortsinformation aus dem Textinhalt der Webseite (*body*) und aus dem *description*-Meta-Tag werden deshalb mit Hilfe der Methode learnNewRules (...) semantische Regeln extrahiert.

3.5.8 Die Klasse *Statistics*

Das Statistik-Modul des Webroboters wird durch diese Klasse implementiert. Sie fasst im Grunde mehrere Variablen zusammen, die durch die Speicherung verschiedener Größen die Suche nach Webseiten mit Ortsinformation erfassen. Solche Größen sind zum Beispiel die Anzahl besuchter URLs, die durchschnittliche Zeit, die für die Analyse einer Webseite benötigt wurde oder die Anzahl gefundener Ortsinformationen. Der Aufbau der Statistik-Datei wurde bereits in Kapitel 3.4.5.3 auf Seite 53 vorgestellt. In diesem Abschnitt werden einige Methoden erläutert, die die Sammlung und Speicherung dieser Werte ermöglichen.

- update(String updatetype, Object updatevalue)

  Mit Hilfe dieser Methode können die einzelnen statistischen Größen aktualisiert werden. Als erster Parameter (*updatetype*) wird die Art der Aktualisierung erwartet, als zweiter dann ein neuer Wert oder ein Erhöhungsschritt für die ändernde Größe. So wird dem Statistik-Modul zum Beispiel durch den Methodenaufruf

  update(„*timestamp*“, „*startparsing*“)

  mitgeteilt, dass das Parsen der momentan untersuchten Webseite gestartet wurde. Ein anderes Beispiel wäre der Aufruf

  update(„*spatialinfo*“, „*bodyaddress*“),

  der das Finden einer Adresse in dem Textinhalt einer Webseite signalisiert.

- writeStatsToFile()

  In bestimmten Zeitintervallen (siehe Konfigurationsparameter *StatisticsUpdateFrequency*) werden die gesammelten Informationen in die Statistik-Datei *„DCbot.stats“* übernommen. Diese Aufgabe übernimmt die Methode writeStatsToFile(). Dabei werden die zahlreichen Zahlenwerte durch mehrere Hilfsmethoden jeweils in eine entsprechende *String*-Darstellung gebracht.

KAPITEL 4 BEWERTUNG DES SYSTEMS

Dieses Kapitel stellt zwei Testläufe vor und bewertet an Hand dieser Ergebnisse die implementierte Softwarelösung.

4.1 Einleitung

In der Studienarbeit des Autors wurde versucht, einen Ortsbezug zu Webseiten über speziellen Meta-Tags, die das Metadatenstandard der *Dublin Core Metadata Initiative* verwenden, herzustellen. Das ernüchternde Ergebnis zeigte, dass Meta-Tags nur sehr selten (ca. 2 Prozent der Webseiten) *Dublin Core* Metadaten enthielten, und dass das spezielle Metadatenelement *DC.Coverage*, die zur Angabe von Ortsinformationen verwendet werden kann, so gut wie nie (ca. 0,05 Prozent der Fälle) vorkam.

Um so spannender war die Frage, welche Ergebnisse die neu implementierten Analysekomponenten des Webroboters liefern. Um dies festzustellen, wurden mehrere Testläufe mit einigen tausend Webseiten durchgeführt. Zwei dieser Testläufe, die das Gesamtergebnis und die Leistung des Webroboters repräsentativ darstellen, werden nachfolgend vorgestellt.

4.2 Testumgebung

Der Webroboter wurde auf einem *UltraSPARC-60* mit 640 MB RAM unter dem Betriebssystem *SunOS 5.8* und unter der *Java Version 1.4.0* getestet. Die Speicherung der Ergebnisse erfolgte mit dem Datenbanksystem *DB2 UDB V7.2*. Der Katalog *GeoBase*, den der Webroboter für die Verifizierung von Ortsinformationen benötigt, wurde in der gleichen Datenbank, wie die Ergebnisse, gespeichert.

Um die Testergebnisse richtig interpretieren zu können, ist es wichtig zu wissen, welche Datensätze der Katalog enthält. Da der Webroboter deutschsprachige Webseiten untersucht, wurde für die Testläufe Deutschland als Testgebiet gewählt. Aus diesem Grund enthält der Katalog Orte, Gebiete oder Gebäude ausschließlich aus Deutschland.

Speziell für diese Arbeit wurde ein Teil der Datenbasis *Datafactory Geocode*, die von der Deutsche Post AG vermarktet wird, erworben. Durch manuelles Hinzufügen einiger weiterer Datensätze konnten folgende Ortsnamen in *GeoBase* importiert werden:

- 1 Planet (Erde)
- 1 Kontinent (Europa)
- 1 Land (Deutschland)
- 16 Bundesländer
- 6.345 Städte
- 8.264 Postleitzahlbereiche

Diese 14.628 Datensätze enthalten genaue (die Genauigkeit hängt selbstverständlich von der Größe des Ortes ab) Positionsdaten. Zu Testzwecken wurden jedoch weitere Datensätze, deren Positionsangaben nur Näherungswerte sind, erstellt und in *GeoBase* importiert. Einige dieser Ortsnamen sind:

- 4.195 Straßen aus Stuttgart
- 35 Universitäten aus Deutschland
- 77 Fachhochschulen aus Deutschland
- 21 Theater aus Stuttgart
- 59 Hotels aus Stuttgart
- 57 Restaurants aus Stuttgart

Die insgesamt 19.159 Datensätze liegen in Form von *ASCII*-Dateien dem Webroboter bei. Für jede der Tabellen von *GeoBase* existiert eine separate Datei, die die Datensätze für die entsprechenden Tabelle bereit hält. Mit Hilfe einer Import-Funktion können diese Datensätze einfach in ein Datenbanksystem eingefügt werden. Zu den *ASCII*-Dateien wurde das Skript *create_geobase_tables.sql* hinzugefügt, das die Erstellung der benötigten Tabellen und das Importieren der Daten für das Datenbanksystem *DB2* erledigt.

4.3 Testergebnisse

Während der zwei nachfolgend vorgestellten Testläufe wurden jeweils 25.000 URLs untersucht. Die komplette Konfigurationsdatei, die bei den Testläufen verwendet wurde, ist in Kapitel C.1 auf Seite 97 zu finden. Damit die Ergebnisse der einzelnen Testläufe vergleichbar bleiben, wurden jeweils die gleichen Analyseeinstellungen verwendet. Die Werte zweier Konfigurationsparameter (*MaxDepth* und *MaxLinkStorage*), die bei den Testläufen abwichen, werden später erläutert. Die Regel-Datei des Webroboters war zum Anfang der Testläufe leer, so dass *DCbot* nur semantische Regeln verwendete, die er selber während des Suchprozesses aus Webseiten extrahiert hat.

Während früherer Testläufe wurde festgestellt, dass Webseiten im Durchschnitt ungefähr neun Verweise enthielten. Dies bedeutet, dass die Anzahl von Webseiten bis zur Entfernung d=5 bereits

$$AnzahlWebseiten = \sum_{n=1}^{5} 9^n = 9 + 81 + 729 + 6561 + 59049 = 66429$$

beträgt. Damit der Webroboter nicht nur Webseiten in der direkten Umgebung der Startseite untersucht, muss der URL-Puffer (Konfigurationsparameter *MaxLinkStorage*) sinnvoll begrenzt werden.

Der Webroboter bietet dem Anwender die Möglichkeit, auf Webseiten gefundene Ortsinformationen detailliert zu bewerten, damit nur Seiten mit erwünschter *„Qualität“* gespeichert werden. Für die beiden, nachfolgend vorgestellten, Testläufe wurde der Konfigurationsparameter *WeightThreshold* niedrig eingestellt, damit möglichst viele Webseiten mit Ortsinformationen gefunden werden.

Während der Testläufe hat der Webroboter zahlreiche statistische Größen in seiner Statistik-Datei festgehalten. Die kompletten Statistik-Dateien der zwei Testläufe sind

im Anhang C enthalten. Wie diese Kennzahlen zu interpretieren sind wurde in Kapitel 3.4.5.3 bereits vorgestellt.

4.3.1 Erster Testlauf

Der erste Testlauf wurde von der Homepage der Zeitschrift *Spiegel*, „*http://www.spiegel.de*“, gestartet. Von dieser Seite aus sollten Webseiten mit breit gefächerter Themenauswahl erreicht werden. Die wichtigsten Konfigurationseinstellungen für diesen Testlauf waren:

```
ServerMode      multi
MaxRequests     25000
MaxDepth        25
MaxLinkStorage  1000
```

Der Webroboter sollte demnach 25.000 URLs bis zur maximalen Entfernung (vom Start-URL aus) von 30 Verweisen untersuchen. Dabei sollte er auch Verweise außerhalb der Domain *spiegel.de* verfolgen, aber in jeder Entfernungsstufe maximal 1.000 URLs besuchen.

Kennzahlen, die die Erreichbarkeit von Webseiten in diesem Testlauf widerspiegeln sind in der Tabelle 4-1 zusammengefasst.

Status	Webseitenanzahl	Anteil
Unerreichbar	1.299	5,2 %
Kein HTML-Dokument	650	2,6 %
Kein Zugriff	1.023	4,09 %
HTML ohne Ortsinformationen	16.903	67,61 %
HTML mit Ortsinformationen	5.125	20,5 %

Tabelle 4-1: Erreichbarkeitskennzahlen des ersten Testlaufs

Demnach waren 5,2 Prozent der Webseiten unerreichbar und in 4,09 Prozent der Fälle war die Untersuchung der Seite durch einen Webroboter nicht erwünscht. Weitere 2,6 Prozent der URLs referenzierten kein *HTML*-Dokument. Der Großteil (88,11 Prozent) der URLs und der dahinter stehenden Webseiten konnte jedoch ordnungsgemäß analysiert werden. In 23,7 % der 22.028 analysierten Webseiten wurden Ortsinformationen gefunden. Die Untersuchung einer Webseite dauerte im Durchschnitt 3,2 Sekunden.

Wo auf einer Webseite und mit welcher Methode Ortsinformationen gefunden wurden, wird in der Tabelle 4-2 dargestellt. Demnach hat der Webroboter die meisten Ortsinformationen (52,41 Prozent) im eigentlichen Webseitentext gefunden. Überraschend ist der hohe Anteil (29,87 Prozent) der, in Verweistexten gefundenen, Ortsangaben. Andere Webseitenteile wie der Titel (5,5 Prozent) oder die URL (5,45 Prozent) enthielten selbstverständlich in weitaus weniger Fällen die gesuchten Daten. Jedoch sind diese Informationen nicht zu unterschätzen. Eine Ortsinformation in dem Titel,

in den Meta-Tags oder in der URL einer Webseite stellt einen sehr sicheren Ortsbezug dar und ist deshalb viel höher zu bewerten.

method / pagepart	address	special-word	rule	first-n-words	most-frequent	other	total
host	X	X	X	X	X	610	610
path	X	X	X	X	X	298	298
title	36	61	59	X	X	760	916
description	3	6	96	396	X	X	501
keywords	X	X	X	628	X	X	628
coverage	0	0	X	X	X	0	0
body	2.821	513	1.489	1.629	2.281	X	8.733
anchors	X	111	578	X	X	4.288	4.977
total	2.860	691	2.222	2.653	2.281	5.956	16.663

Tabelle 4-2: Ortsinformationsverteilung des ersten Testlaufs

Die Differenzierung nach der Suchmethode zeigt, dass die Adressenerkennung (17,16 Prozent), die Anwendung semantischer Regeln (13,33 Prozent), die Untersuchung der ersten Wörter einer Webseitenkomponente (15,92 Prozent) und die Analyse der häufigsten Wörter einer Seite (13,69 Prozent), relativ gleichmäßig verteilt, zwischen 13 und 17 Prozent der Ortsinformationen lieferten. Die Erkennung von Schlüsselwörtern konnte weniger Erfolg verbuchen (4,15 Prozent), was sicherlich mit der Tatsache zusammenhängt, dass durch Schlüsselwörter wie *„Museum“* oder *„Universität“* stets nur sehr genaue Ortsinformationen gefunden werden können, während die anderen Methoden auch ungenauere Daten sammeln. Ein weiterer Grund war die relativ kleine Anzahl von verwendeten Schlüsselwörtern.

Neben der Aufschlüsselung gefundener Ortsinformationen nach der Suchmethode beziehungsweise nach Webseitenteilen wäre es interessant zu wissen, welche Genauigkeit diese Daten haben. Die Antwort auf diese Frage liefert die Tabelle 4-3, in der die wichtigsten, während diesem Testlauf gefundenen, Ortsinformationen nach ihren Ortstyp aufgelistet sind.

Ortstyp	Anzahl Ortsinformationen
city	10.759
lake	70
postal code area	2.351
university	157
college	49
street	116

Tabelle 4-3: Ortstypsverteilung des ersten Testlaufs

Es ist leicht zu erkennen, dass die häufigsten Ortsinformationen (64,57 Prozent) Städtenamen sind, während genauere Ortsbezüge, wie Adressen oder Namen von Hochschulen (16,46 Prozent), seltener vorkommen.

4.3.2 Zweiter Testlauf

Für den ersten Testlauf wurde die Start-URL für die Suche themenunabhängig gewählt. Damit sollte sichergestellt werden, dass die gesammelten Ergebnisse eine repräsentative Übersicht über die Verbreitung von Ortsinformationen in Webseiten lieferten. Möchte der Anwender Webseiten zu einem bestimmten Gebiet (zum Beispiel zu der Stadt *„Stuttgart“*) finden, so ist es sinnvoll, die Startseite entsprechend zu wählen. Durch den zweiten Testlauf soll festgestellt werden, wie viele Ortsinformationen, ausgehend von einer themenspezifischen Startseite, gefunden werden.

Die Startseite des zweiten Testlaufs war eine Ergebnisseite der Suchmaschine *Google*, die die ersten hundert Webseiten dieser Suchmaschine mit URLs, die das Wort *„Stuttgart“* enthielten, auflistete. Aus zwei Gründen wurden signifikant mehr Webseiten mit Ortsinformationen, als bei dem ersten Testlauf, erwartet. Der erste, offensichtliche Grund war die stark themenbezogene Startseite. Ein weiterer Grund für diese Vermutung waren die *„Stuttgart“*-spezifischen Datensätze (Straßen, Theater, Restaurants aus Stuttgart) in *GeoBase*.

Die wichtigsten Konfigurationseinstellungen für diesen Testlauf waren:

ServerMode	multi
MaxRequests	25000
MaxDepth	10
MaxLinkStorage	2500

Die maximale Entfernung von der Startseite aus betrachtet, in der der Webroboter Webseiten untersuchen sollte, wurde auf zehn Verweise beschränkt. Diese Einstellung sollte sicherstellen, dass der Webroboter sich nicht zu weit vom Themengebiet entfernt, aber gleichzeitig auch nicht nur die direkte Umgebung der Startseite untersucht.

Die Erreichbarkeit von Webseiten während dieses Testlaufs wird in der Tabelle 4-4 dargestellt.

Status	Webseitenanzahl	Anteil
Unerreichbar	1.043	4,17 %
Kein HTML-Dokument	631	2,52 %
Kein Zugriff	646	2,58 %
HTML ohne Ortsinformationen	10.480	41,92 %
HTML mit Ortsinformationen	12.200	48,81 %

Tabelle 4-4: Erreichbarkeitskennzahlen des zweiten Testlaufs

Die Anzahl nicht untersuchbarer Webseiten des zweiten Testlaufs (insgesamt 11,89 Prozent) war mit den entsprechenden Kennzahlen des ersten Testlaufs (insgesamt 9,27 Prozent) vergleichbar. Die Anzahl von Webseiten mit Ortsinformationen (aus 22.680 in diesem Testlauf analysierten Webseiten) nahm jedoch, wie erwartet, stark (von 23,7 Prozent auf 53,79 Prozent) zu.

Die Aufschlüsselung gefundener Ortsinformationen nach der Suchmethode beziehungsweise nach dem Fundort wird in der Tabelle 4-5 dargestellt.

method / pagepart	address	special-word	rule	first-n-words	most-frequent	other	total
host	X	X	X	X	X	5.654	5.654
path	X	X	X	X	X	863	863
title	13	421	202	X	X	3.838	4.474
description	7	192	356	1.698	X	X	2.253
keywords	X	X	X	2.813	X	X	2.813
coverage	0	0	X	X	X	45	45
body	5.092	1.657	2.374	3.487	5.874	X	18.484
anchors	X	820	514	X	X	8.926	10.260
total	5.112	3.090	3.446	7.998	5.874	19.326	44.846

Tabelle 4-5: Ortsinformationsverteilung des zweiten Testlaufs

Während die Anzahl gefundener Webseiten mit Ortsinformationen sich, verglichen mit dem ersten Testlauf, mehr als verdoppelte (238 Prozent), hat die Gesamtanzahl der Ortsinformationen noch stärker (269 Prozent) zugenommen. Dies bedeutet, dass Webseiten im Durchschnitt auch mehr Ortsinformationen (von 3,25 auf 3,68 pro Seite) enthielten.

Die meisten Ortsbezüge (41,22 Prozent) wurden erneut im Webseitentext gefunden, während den zweiten Platz wieder die Verweistexte (mit 22,88 Prozent) belegen. Bestimmt durch die Art der Startseite ist es nicht verwunderlich, dass in relativ vielen URLs (12,61 Prozent) ebenfalls Ortsinformationen gefunden wurden. Der höhere Anteil von Ortsbezügen in den Titeln (9,98 Prozent), in den *description*-Meta-Tags (5,02 Prozent) und in den *keywords*-Meta-Tags (6,27 Prozent) der Webseiten zeigen, dass nicht nur die Quantität, sondern auch die Qualität gefundener Webseiten stieg.

Wenn man die Verteilung gefundener Ortsinformationen nach der angewendeten Methode betrachtet, kann man feststellen, dass in diesem Testlauf relativ betrachtet weniger Ortsbezüge mit der Adressenerkennung (11,4 Prozent) und mit den semantischen Regeln (7,68 Prozent) entdeckt wurden. Die Leistung der anderen Methoden war mit dem ersten Testlauf vergleichbar.

Die Verteilung nach dem Ortstyp, dargestellt in der Tabelle 4-6, zeigt ebenfalls, dass die Genauigkeit der gefundenen Ortsinformationen, im Vergleich mit dem ersten Testlauf, zugenommen hat. Die Position für die Straße einer Adresse konnte bedeutend öfter (24,96 Prozent statt 4,06 Prozent) ermittelt werden. Die Ursache dafür sind

die entsprechenden *„Straßen"*-Datensätze von Stuttgart in *GeoBase*. Aber auch nicht *„Stuttgart"*-spezifische Daten, wie Namen deutscher Hochschulen wurden doppelt so oft erkannt. Einen noch größeren Teil als bei dem ersten Testlauf (73,92 Prozent) der gefundenen Ortsinformationen machten Städtenamen aus.

Ortstyp	Anzahl Ortsinformationen
city	33.151
lake	363
postal code area	3.504
university	1.941
college	285
street	1.276
museum	24
theater	26

Tabelle 4-6: Ortstypsverteilung des zweiten Testlaufs

4.3.3 Semantische Regeln

Wie bereits des Öfteren erwähnt, extrahiert der Webroboter während der Analyse von Webseiten semantische Regeln aus diesen Webseiten (für Einzelheiten siehe Kapitel 3.4.2.2). Einige der besten Regeln, die der Webroboter während seiner Suche fand, waren:

[5.1660395,in,<spatial_info>]
[3.5285192,Universität,<spatial_info>]
[0.38768813,Region,<spatial_info>]
[0.30093467,Stadt,<spatial_info>]
[0.24806738,Fachhochschule,<spatial_info>]
[0.24264531,Landeshauptstadt,<spatial_info>]
[0.21282288,Standort,<spatial_info>]
[0.14233387,Flughafen,<spatial_info>]
[0.13826753,Messe,<spatial_info>]

Wie erwartet hat sich die Qualität der Regeln während der Testläufe kontinuierlich verbessert. Je mehr Webseiten untersucht wurden, desto mehr „sinnvolle" (nachvollziehbare) Regeln konnten in der Regel-Datei gefunden werden.

4.4 Fazit

Die Ergebnisse der beiden Testläufe zeigen, dass der Webroboter seine Aufgabe zuverlässig erledigt. In 34,65 Prozent der 44.708 analysierten Webseiten konnten Ortsin-

formationen gefunden werden, und eine Webseite enthielt im Durchschnitt 3,55 Ortsangaben. Die Analyse einer Webseite dauerte im Durchschnitt 2,88 Sekunden. Der überwiegende Teil gefundener Ortsinformationen (71,39 Prozent) waren Städtenamen. Die Genauigkeit dieser Ortsbezüge ist für die meisten zukünftigen Anwendungen nicht zufrieden stellend. Genaue Ortsinformationen, wie vollständige Adressen, Namen von Museen oder Hochschulen kamen in 4,77 Prozent der Fälle vor. Dies bedeutet allerdings keines Falls, dass genauere Ortsinformationen auf Webseiten selten vorkommen, oder dass *DCbot* sie nicht finden kann. Wenn man bedenkt, dass nur 1,75 Prozent der Datensätze des Katalogs *GeoBase* kleine Stätten repräsentieren, und damit die Erkennung von Namen solcher Stätten auf Webseiten ermöglichen, wird schnell klar, dass die Erweiterung des Katalogs die Leistung des Webroboters stark positiv beeinflussen kann.

Die implementierten Verfahren haben, ohne Ausnahme, signifikant zum Gesamtergebnis beigetragen und damit ihre Berechtigung als Teil des Systems bewiesen. Zwei der verwendeten Methoden können jedoch ohne großen Aufwand weiter optimiert werden. Wie bereits erwähnt, verbessern sich die semantischen Regeln des Webroboters während den Suchläufen automatisch. Für die Bildung einer soliden Regeldatenbasis müssen jedoch sicherlich noch einige Suchläufe stattfinden. Die zweite Methode ist die Erkennung von Schlüsselwörtern auf Webseiten. Durch das Hinzufügen weiterer Schlüsselwörter zu der Konfigurationsdatei des Webroboters kann die Leistung dieser Methode weiter erhöht werden.

Der wichtigste Ansatz zur Verbesserung der Gesamtleistung des Webroboters ist die Erweiterung des Katalogs um weitere, und vor allem genauere, Datensätze. Dies ist insbesondere wichtig, weil dadurch die Leistung des gesamten Systems, unabhängig von der verwendeten Analysemethode, verbessert wird.

DCbot bietet die Möglichkeit, gefundene Ortsinformationen nach ihrer Genauigkeit und nach dem „Fundort“ (Webseitenkomponente in dem die jeweilige Ortsinformation gefunden wurde) zu bewerten (siehe Konfigurationsparameter *<Webseitenteil>Weights* in Kapitel 3.4.3.2). Man kann den Webroboter zum Beispiel so einstellen, dass er nur nach genauen Ortsinformationen (Stätten mit kleinem Flächeninhalt), ausschließlich in aussagekräftigen (z.B. Titel und Meta-Tags) Webseitenkomponenten sucht. In diesem Fall ist zu erwarten, dass zwar viel weniger Webseiten gefunden werden, diese dafür aber sehr gute Ortsinformationen enthalten. Da die Einstellungsmöglichkeiten des Webroboters in dieser Hinsicht sehr vielfältig sind wird es dem Anwender überlassen, eine für seine Bedürfnisse zugeschnittene Konfiguration zu erstellen und die Suchergebnisse mit seinen Erwartungen zu vergleichen.

KAPITEL 5 ZUSAMMENFASSUNG

Dieses Kapitel fasst die Arbeit kurz zusammen und gibt Empfehlungen zur künftigen Verbesserung der implementierten Softwarelösung.

Die Zielsetzung dieser Arbeit war die Untersuchung und Implementierung von Verfahren, die Ortsinformationen auf Webseiten identifizieren, und dadurch die Sammlung von Webseiten mit einem Ortsbezug ermöglichen. Im ersten Teil der Arbeit wurden verschiedene Arten von Ortsinformationen und Webseitenkomponenten, die diese Informationen enthalten können, vorgestellt. Darauf folgte die Präsentation der Implementierungslösung, in der Form der Erweiterung des Webroboters *DCbot*. Den Schluss bildeten die Darstellung zweier Testläufe und die Bewertung des Systems.

In der Studienarbeit des Autors [Süt01] wurde die Möglichkeit untersucht, einen Ortsbezug zu Webseiten über Meta-Tags herzustellen. Dabei wurden in erster Linie spezielle Meta-Tags (*DC.Coverage*), die die Angabe von Ortsinformationen versprachen, untersucht. Leider wurden entsprechende Meta-Tags nur äußerst selten (in 0,08 Prozent der untersuchten Webseiten) verwendet, und wenn, dann enthielten sie Ortsinformationen in natürlicher Sprache. Diese Angaben konnten damals noch nicht sinnvoll interpretiert werden.

Durch die Anwendung verschiedener Analysemethoden konnte der, in Rahmen dieser Arbeit erweiterte, Webroboter *DCbot* in rund ein Drittel (34,65 Prozent) der Webseiten Ortsinformationen aufspüren. Während des zweiten Testlaufs hat der Webroboter sogar in 0,18 Prozent der Webseiten Ortsinformationen in *DC.Coverage*-Meta-Tags aufspüren können.

Dieses hervorragende Ergebnis muss allerdings etwas relativiert werden. Ein großer Anteil der, in Webseiten gefundenen, Ortsinformationen waren eher ungenau (Städtenamen). Dieser Teil der Ergebnisse kann deshalb die meisten zukünftigen Anwendungen nicht zufrieden stellen. In Kapitel 4.4 auf Seite 77 werden mehrere Möglichkeiten angedeutet, die die Leistung des Webroboters unter anderem in dieser Hinsicht erhöhen. Die wichtigste zukünftige Aufgabe ist sicherlich die Erweiterung des Katalogs *GeoBase*, damit der Anteil gefundener Webseiten, mit genauer Ortsinformationen, gesteigert werden kann.

Der große Anteil von ungenauen Ortsinformationen stellt den Webroboter trotzdem vor keine allzu großen Probleme. *DCbot* bietet dem Anwender die Möglichkeit, gefundene Ortsinformationen, sowohl nach ihrer Genauigkeit, als auch nach dem Webseitenteil, in dem die entsprechenden Informationen gefunden wurden, zu bewerten. Damit ist die Sammlung von Webseiten, mit zuverlässigen und genauen Ortsinformationen, mit dem Webroboter auf einfache Art möglich.

ANHANG A DENIC-EINTRÄGE

Dieser Anhang enthält einige Einträge aus der *DENIC* Whois-Datenbank.

A.1 DENIC-Eintrag für *„stuttgart.de“*

Domaininhaber:

Landeshauptstadt Stuttgart, Haupt- und Personalamt
Abt. Informations- und Kommunikationstechnik
Eberhardstr. 6
D-70173 Stuttgart
Germany
DE

Administrativer Ansprechpartner:

Name:	Roland Schaefer
Kontakttyp:	Person
Adresse:	Landeshauptstadt Stuttgart Eberhardstr. 6
Stadt:	Stuttgart
PLZ:	70173
Land:	GERMANY

Technischer Ansprechpartner:

Name:	Tiscali Hostmaster Role-Account
Kontakttyp:	Role
Adresse:	Robert-Bosch-Str. 32
Stadt:	Dreieich
PLZ:	63303
Land:	GERMANY
Telefon:	+49 6103 9160
Telefax:	+49 6103 916222
E-Mail:	guardian@nacamar.net

A.2 DENIC-Eintrag für *„zugspitze.de“*

Domaininhaber:

Bayerische Zugspitzbahn AG
Olympiastrasse 27
82467 Garmisch-Partenkirchen
D

Administrativer Ansprechpartner:

Name:	Karlheinz Gruebler
Kontakttyp:	Person
Adresse:	Bayerische Zugspitzbahn AG
	Olympiastrasse 27
Stadt:	Garmisch-Partenkirchen
PLZ:	82467
Land:	GERMANY

Technischer Ansprechpartner:

Name:	Christian Storch
Kontakttyp:	Person
Adresse:	InfraNet AG
	Hansastr. 136
Stadt:	Muenchen
PLZ:	81373
Land:	GERMANY
Telefon:	+49 89 7435230
Telefax:	+49 89 74352323
E-Mail:	storch@infra.net

A.3 DENIC-Eintrag für *„t-online.de“*

Domaininhaber:

T-Online International AG
Waldstrasse 3
D-64331 Weiterstadt
DE

Administrativer Ansprechpartner:

Name:	Daniel Kaufmann
Kontakttyp:	Person
Adresse:	Waldstrasse 3
Stadt:	Weiterstadt
PLZ:	64331
Land:	GERMANY

Technischer Ansprechpartner:

Name:	Hostmaster T-Online
Kontakttyp:	Person
Adresse:	Waldstrasse 3
Stadt:	Weiterstadt
PLZ:	64331
Land:	GERMANY
Telefon:	+49 6151 680 537
Telefax:	+49 6151 680 519
E-Mail:	hostmaster@t-online.net

ANHANG B BESCHREIBUNG VON TGN-TABELLEN

Dieser Anhang enthält unter anderem die Beschreibung mehrerer Tabellen des *The Getty Thesaurus of Geographic Names*.

B.1 Eintrag für *„Stuttgart“* aus dem *TGN*

Stuttgart (inhabited place)

Lat: 48 47 N Long: 009 12 E (represented in degrees minutes direction)
Lat: 48.783 Long: 9.200 (represented in decimal degress and fractions of degrees)

Note - Located on Neckar river, in forested area of vineyards & orchards; town developed around a medieval fortified manor & stud farm; went to House of Württemberg in 13th cen; declined during Thirty Years War in 17th cen; nearly destroyed in WW II.

Hierarchical Position:

Europe.....(continent)
Deutschland.....(nation)
Baden-Württemberg.....(state)
Stuttgart.....(district (national))

Names:

Stuttgart (C,V)
Stuotgarten (H,V)..... name of stud farm on the site ca. 950

Place Types:

inhabited place (C)..... prehistoric & Roman settlements were nearby, but this site was first settled in 10th cen. AD, chartered in 13th cen.
city (C)
state capital (C)
district capital (C)
port (C).....opened in 1958
agricultural center (C)..... has extensive wine & fruit trade
industrial center (C)..... grew in 19th cen., today produces automobiles, machinery, textiles, clothing, precision instruments, beer, shoes & chemicals
transportation center (C)..... important rail junction on natural route connecting the Danube with N Germany & the Rhein
publishing center (C)
university center (C)
cultural center (C).....noted for museums & institutes of art, music, architecture, opera & ballet
noble residence (H)..... of counts, dukes & kings of Württemberg, from 1320

Sources:

Stuotgarten..... Encyclopædia Britannica (1988), X, 337 [VP]
Stuttgart..... USBGN: Foreign Gazetteers [BHA]
Times Atlas of the World (1992), 190 [VP]
Canby, Historic Places (1984), II, 894 [VP]
Webster's Geographical Dictionary (1984) [GCPS]
Webster's Geographical Dictionary (1988), 1160 [VP]

B.2 Wichtigste Tabellen des *TGN*

COLUMN	TYPE	Len	DESCRIPTION	EXAMPLE	NOTE
geog_key	int	64	Unique key for the place.	7011179	
descr_note	text	500	Note describing the history, physical characteristics, and significance of the place.	Founded by Etruscans; ruled by Romans 3rd cen.	Note that no value in this column is currently greater than 500, even though this is a text field.
lat	varchar	5	The degrees and minutes of the latitude, which is the angular distance north or south of the equator, measured in degrees and minutes along a meridian.	43 19	Format is two digits, one space, two digits.
lat_dir	char	1	Letter indication if latitude is north or south of equator.	N	Legal values = N or S.
long	varchar	6	The degrees and minutes or the longitude, which is the angular distance east or west of the Prime Meridian at Greenwich, England	011 21	Format is three digits, one space, two digits.
long_dir	char	1	Letter indicating if latitude is east or west of the Prime Meridian.	E	Legal values = E or W.
lat_num	dec(5,3) or float		Latitude expressed as decimal fractions of degrees.	43.316	Whole degrees are represented by a two-digit decimal number ranging from 0 through 90, with any decimal fraction af a degree separated ba a decimal point; latitudes south of the equator are designated by a minus sign.
long_num	dec(6,3) or float		Longitude expressed as a decimal fractions of degrees.	011.350	Whole degrees are represented ba a three-digit decimal number ranging from 0 through 180, with any decimal fraction of a dgree separated by a decimal point; longitudes wast of the Prime Meridian are designated by a minus sign.

Tabelle B-1: *TGN*-Tabelle GEOG_MAIN

COLUMN	TYPE	Len	DESCRIPTION	EXAMPLE	NOTE
geog_key	int	4	Unique key for the place.	7011179	
parent_key	int	4	Unique key for a parent of the place.	7003168	
pref_flag	varchar	1	Flag indicating if parent is P = prefered, or N = Non-preferred.	P	Legal values = P or N.

Tabelle B-2: *TGN*-Tabelle GEOG_PARENTS

COLUMN	TYPE	Len	DESCRIPTION	EXAMPLE	NOTE
geog_key	int	4	Unique key for the place.	7011179	
placename_id	int	4	Unique key for this name (unique across entire database)	47413	All names habe unique IDs, including homonyms.
seqn	int	1	Sequence position in which the name should be ordered in displays.	0	Values = 0-x.
place_name	varchar	122	Name string.	Siena	
sort_name	varchar	122	Normalized name; all caps with punctuation, diacritic codes, numbers, and spaces removed.	SIENA	To be used for sorting and retrieval.
lang	char	1	Flag indication if name is V = Vernacular, or O = Other language.	V	Legal values = V or O. In restructuring of data for new system, this will be expanded to handle multiple languages.
age_flag	char	1	Flag indicating if the name is C = Current, H = Historical, or B = Both.	C	Legal values = C, H or B.
pref_flag	char	1	Flag indicationg if this is the P = Preferred name, or V = Variant name.	P	Legal values = P or V.

Tabelle B-3: *TGN*-Tabelle GEOG_NAME

COLUMN	TYPE	Len	DESCRIPTION	EXAMPLE	NOTE
geog_key	int	4	Unique key for the place.	7011179	
seqn	int	1	Sequence position in which the place type should be ordered in displays.	0	Values = 0-x.
pref_flag	char	1	Flag indicating if the place type is P = Preferred, or N = Non-preferred.	P	
placetype_ code	int	4	Numeric code for the place type.	83002	For a given *geog_key*, a place type may be linked only once.
age_flag	char	1	Flag indicationg if the place type is C = Current, H = Historical, or B = Both.	C	

Tabelle B-4: *TGN*-Tabelle GEOG_PLACETYPE

COLUMN	TYPE	Len	DESCRIPTION	EXAMPLE	NOTE
placetype_ code	int	4	Numeric code for the place type.	83002	
placetype_ search	char	4	Character string version of the code; to allow searching by truncation.	83002	
placetype	varchar	64	Place type term.	inhabited place	

Tabelle B-5: *TGN*-Tabelle PLACETYPE_LIST

ANHANG C DCBOT-DATEIEN

Dieser Anhang enthält die Konfigurationsdatei von *DCbot* und die zwei Statistik-Dateien, die während der zwei Testläufe erzeugt wurden.

C.1 Konfigurationsdatei

```
#
# DCbot configuration file
#


#Enviromental Settings

Homepage            http://www.informatik.uni-stuttgart.de
EMail               suetoemy@hermes.informatik.uni-stuttgart.de
MapFile             DCbot.map
RuleFile            DCbot.rules
DBInterface         DBI_JDBC
DBJDBCDriver        db2
DBJDBCClass         COM.ibm.db2.jdbc.app.DB2Driver
DBName              exercise
DBTablePrefix       dcbot


#Retrieval Options

AcceptLanguage              de;q=0.8
ServerMode                  multi
RequestRate                 0
MaxRequests                 25000
MaxLinkStorage              1000
MaxDepth                    25
MaxRules                    10000
GoodRuleThreshold           0.1
RuleUpdateFrequency         500
MaxKeywords                 20
MaxDescriptionWords         75
FirstNBodyWords             75
FirstNFrequentBodyWords     10
StatisticsUpdateFrequency   500
StreetIndicators            weg/allee/straße/hof/str/passage/platz/...
StructureIndicators         museum/theater/kino/galerie/bühne/brücke/...
GeoSiteIndicators           see/fluss/wüste/berg/meer/...
```

```
#Rating Section

AreaMap                 0/0.1/50/250

HostWeights             [0.1]-100/100/100/0
PathWeights             [0.5]-85/65/55/0
TitleWeights            [0.1]-100/100/100/0
DescriptionWeights      [0.5]-85/60/50/0
KeywordsWeights         [0.1]-85/85/85/0
CoverageWeights         [0.1]-100/100/100/0
BodyWeights             [0.5]-85/60/40/0
AnchorWeights           [0.5]-80/60/40/0

WeightThreshold         10
ForceOverwrite          no
```

C.2 Statistik-Datei des ersten Testlaufs

```
************************************
* Statistical figures for DCbot  *
************************************

Start of execution:     Wed Mar 05 11:29:37    MET 2003
End of execution:       Thu  Mar 06 09:39:56   MET 2003
Execution time:         22 Hours 10 Minutes 19 Seconds 283 Milliseconds

Number of URLs checked:             25000
Number of web pages analyzed:       22028

Average time to retrieve document:                  801  Milliseconds
Average time to parse document:          1 Seconds  254  Milliseconds
Average time to generate candidates:                412  Milliseconds
Average time to calculate best position:            167  Milliseconds

Average time to analyze web page:        3 Seconds  193  Milliseconds

*** Status codes ***

ERROR                       78
UNKNOWN HOST                41
OK                       16866
OK/UNCHANGED                37
OK/SPATIALINFO            5125
PAGE GONE                    0
NO HTML                    650
FAILURE                   1180
NO ACCESS                 1023
NO FOLLOW                    0
```

*** Generated candidates ***

	address	specialword	rule	firstnwords	mostfrequent	other
host	X	X	X	X	X	54627
path	X	X	X	X	X	78936
title	145	1831	2199	X	X	56504
description	27	340	9025	51018	X	X
keywords	X	X	X	130504	X	X
coverage	0	7	X	X	X	132
body	17522	25789	93623	223273	183665	X
anchors	X	6428	17641	X	X	534028

*** Spatial information ***

	address	specialword	rule	firstnwords	mostfrequent	other
host	X	X	X	X	X	610
path	X	X	X	X	X	298
title	36	61	59	X	X	760
description	3	6	96	396	X	X
keywords	X	X	X	628	X	X
coverage	0	0	X	X	X	0
body	2821	513	1489	1629	2281	X
anchors	X	111	578	X	X	4288

*** Spatial information types ***

city	10759
postal code area	2351
nation	1259
state	987
continent	825
university	157
street	116
planet	90
lake	70
college	49

C.3 Statistik-Datei des zweiten Testlaufs

****
* Statistical figures for DCbot *
****

Start of execution: Sun Mar 09 18:30:26 MET 2003
End of execution: Mon Mar 10 12:16:26 MET 2003
Execution time: 17 Hours 45 Minutes 59 Seconds 891 Milliseconds

Number of URLs checked: 25000
Number of web pages analyzed: 22680

Average time to retrieve document: 614 Milliseconds
Average time to parse document: 544 Milliseconds
Average time to generate candidates: 87 Milliseconds
Average time to calculate best position: 254 Milliseconds

Average time to analyze web page: 2 Seconds 558 Milliseconds

*** Status codes ***

ERROR	47
UNKNOWN HOST	53
OK	10413
OK/UNCHANGED	67
OK/SPATIALINFO	12200
PAGE GONE	0
NO HTML	631
FAILURE	943
NO ACCESS	646
NO FOLLOW	0

*** Generated candidates ***

	address	specialword	rule	firstnwords	mostfrequent	other
host	X	X	X	X	X	63576
path	X	X	X	X	X	63792
title	39	3242	3146	X	X	59070
description	29	1470	5159	40489	X	X
keywords	X	X	X	74197	X	X
coverage	0	0	X	X	X	285
body	27334	26080	123468	178921	165352	X
anchors	X	7271	16994	X	X	402980

*** Spatial information ***

	address	specialword	rule	firstnwords	mostfrequent	other
host	X	X	X	X	X	5654
path	X	X	X	X	X	863
title	13	421	202	X	X	3838
description	7	192	356	1698	X	X
keywords	X	X	X	2813	X	X
coverage	0	0	X	X	X	45
body	5092	1657	2374	3487	5874	X
anchors	X	820	514	X	X	8926

*** Spatial information types ***

city	33151
postal code area	3504
nation	2135
university	1941
street	1276
state	1059
continent	959
lake	363
college	285
planet	102
theatre	26
museum	24
mountain	20
hotel	1

Literaturverweise

[BCG99] Buyukkokten, O.; Cho, J.; Garcia-Molina, H.: *Exploiting geographical location information of web pages.* Department of Computer Science, Stanford University, Palo alto, 1999

[BFI98] Berners-Lee, T.; Fielding, R.; Irvine, U.C.; Masinter, L.: *Uniform Resource Identifiers (URI): Generic Syntax.* The Internet Society, 1998

[Bec01] Becker, T.: *Verbreitung mobiler Internetlösungen.* Cap Gemini Ernst & Young, 2001, Available from: http://www.marketing-marktplatz.de/Marktforschung/CapGeminiMobile.htm

[BMM94] Berners-Lee, T.; Masinter, L.; McCahill, M.: *Uniform Resource Locators (URL).* Network Working Group RFC 1738, 1994

[BP98] Brin, S.; Page, L.: *The Anatomy of a Large-Scale Hypertextual Web Search Engine*. Computer Science Department, Stanford University, 1998

[BSY95] Balabanovic, M.; Shoham, Y.; Yun, Y.: *An Adaptive Agent for Automated Web Browsing.* Department of Computer Science, Stanford University, Stanford, 1995

[Bus45] Bush, V.: *As We May Think.* The Atlantic Monthly, Volume 176, 1945

[CFM98] Craven, M.; Freitag, D.; McCallum, A.: *Learning to Extract Symbolic Knowledge from the World Wide Web.* School of Computer Science, Carnegie Mellon University, Pittsburgh, 1998

[Cun97] Cunningham, H.: *Information Extraction - A User Guide.* Institute for Language, Speech and Hearing and Department of Computer Science, University of Sheffield, Sheffield, 1997

[DCM99] DCMI: *Dublin Core Metadata Element Set, Version 1.1: Reference Description.* Dublin Core Metadata Initiative, 1999, Available from: http://www.dublincore.org/documents/1999/07/02/dces/

[Den02] DENIC eG: *Über die DENIC eG.* 2002, Available from: http://www.denic.de/doc/DENIC/index.html

[DVG96] Davis, C.; Vixie, P.; Goodwin, T.; Dickinson, I.: *A Means for Expressing Location Information in the Domain Name System.* Network Working Group RFC 1876, 1996

[Eil98] Eilebrecht, L.: *Resource Discovery und Information Retrieval.* Universität-Gesamthochschule Siegen, 1998

[FGM97] Fielding, R.;Gettys, J.;Mogul, J.: *Hypertext Transfer Protocol -- HTTP/1.1.* Network Working Group RFC 2068, 1997

[FHJ98] Raggett, D.; Le Hors, A.; Jacobs, I.: *HTML 4.0 Spezification.* World Wide Web Consortium, 1998, Available from: http://www.w3.org/TR/REC-html40

[FSP94] Farrell, C.; Schulze, M.; Pleitner, S.: *DNS Encoding of Geographical Location.* Network Working Group RFC 1712, 1994

[Get00] The J. Paul Getty Trust: *Getty Thesaurus of Geographic Names.* 2000, Available from: http://www.getty.edu/research/tools/vocabulary/tgn/index.html

[Goo02a] Goodman, A.: *An End to Metatags.* Traffick.com, 2002, Available from: http://www.traffick.com/article.asp?aID=102,

[Goo02b] Goodman, A.: *Google Uses Meta Tags Sparingly, But Should You?* Traffick.com, 2002, Available from: http://www.traffick.com/article.asp?aID=102

[Har01] Harping, P.: *User´s Guide to the TGN Data Releases Version 2.0*. The Getty Vocabulary Program, 2001

[IGN98] Institut of Geodesy and Navigation: *WGS 84 Implementation Manual.* Institut of Geodesy and Navigation, University FAF Munich, 1998

[IOS02] Iternational Organization for Standardisation: *ISO 3166 English country names and code elements.* ISO 3166 Maintenance Agency, 2002

[Kos94] Koster, M.: *A Standard for Robot Exclusion.* 1994, Available from: http://www.robotstxt.org/wc/norobots.html

[Lyc02] Lycos Europe: *Webmaster FAQs*. 2002, Available from: http://www.HotBot.lycos.de/help/webmasterfaq.html

[Mit97] Mitchell, M. T.: *Machine Learning.* McGraw-Hill Series in Computer Science, 1997

[Moc87] Mockapetris, P.: *Domain Names - Implementation and Specification.* Network Working Group RFC 1035, 1987

[PDE97] Perkowitz, M.; Doorenbos, P. R.; Etizioni, O.: *Learning to Understand Information on the Internet: An Example-Based Approach*. University of Washington, Seattle, 1997

[RHJ99] Raggett, D.; Le Hors, A.; Jacobs, I.: *HTML 4.01 Specification*. World Wide Web Consortium, 1999, Available from: http://www.w3.org/TR/html401/

[Rös02] Röscheisen, E.: *Spurensuche - Von IP-Adressen zu Ortsinformationen.* Magazin iX Ausgabe 08/2002

[Sch01] Schönefeld, T.: *Bedeutungskonstitution im Hypertext.* NETWORX-Online Publikationen, 2001, Available from: http://www.websprache.net/networx/

[ST99] Schlobinski, P.; Tewes, M.: *Graphentheoretische Analyse von Hypertexten.* NETWORX-Online Publikationen, 1999, Available from: http://www.websprache.net/networx/

[Sul02] Sullivan, D.: *How to use HTML Meta Tags.* SearchEngineWatch.com, 2002, Available from: http://searchenginewatch.com/webmasters/meta.html

[Sul03] Sullivan, D.: *Search Engine Sizes.* Search Engine Watch, 2003, Available from: http://searchenginewatch.com/reports/sizes.html

[Süt01] Sütö, M.: *Ortsbasierter Web-Zugriff.* Studienarbeit Nr. 1834, Institut für Parallele und Verteilte Höchstleitungsrechner, Universität Stuttgart, 2001

Erklärung

Ich versichere, dass ich diese Arbeit selbständig verfasst und nur die angegebenen Quellen verwendet habe.

_____

(Mihály Jakob)