

Prüfer: Prof. Dr. Rothermel
Betreuer: Dipl. -Inform. Fritz Hohl

Begonnen am: 1. November 1996
Beendet am: 30. April 1997

CR-Klassifikation: C.2.4, C.4, D.1.3, E.5, H.3.3

Studienarbeit Nr. 1603

**Konzeption und Implementierung einer
Suchmaschine für Usenet-News
unter Verwendung von mobilen Agenten**

Thomas E. Wieger

Kurzfassung

Usenet-News ist ein elektronisches Informationssystem, mit dem weltweit Menschen über ein breites Spektrum an Themen diskutieren und Informationen austauschen. Spezialisierte Rechner, sog. Newsserver, verteilen diese Informationen weltweit.

Die stetige Zunahme des anfallenden Informationsvolumens in Usenet-News haben dazu geführt, daß es einem einzelnen Menschen nicht mehr möglich ist, sich einen Überblick darüber zu verschaffen. Um dieser Problematik zu begegnen, wurden sogenannte Suchmaschinen geschaffen. Dies sind Rechner mit spezieller Software, die eine Recherche nach bestimmten Informationen ermöglichen.

Im Bereich der Entwicklung von Client/Server-Anwendungen, insbesondere bei der Programmierung von Verteilten Systemen, wird seit einiger Zeit ein neuer Ansatz diskutiert, die sog. Mobilen Agenten. Dies sind Programme, die innerhalb einer bestimmten Ablaufumgebung, dem Agentensystem, ausgeführt werden und die Fähigkeit besitzen, zwischen verschiedenen Ablaufumgebungen zu migrieren.

Newsserver sind verteilt, vernetzt und von heterogener Struktur. Für die Recherche in Usenet-News stellen sie die Datenquellen dar. Aufgrund verschiedener Überlegungen paßt diese Aufgabe gut zu den Einsatzgebieten, für die Mobile Agenten vorgesehen wurden. Im Rahmen dieser Studienarbeit wird daher eine Suchmaschine auf Basis eines Mobile-Agenten-Systems implementiert. Dabei soll untersucht werden, wie sich dieser Ansatz von "klassischen" Suchmaschinen, die bisher in diesem Bereich eingesetzt wurden, unterscheidet und welche Vor- und möglicherweise auch Nachteile er mit sich bringt.

Diese Studienarbeit gliedert sich in fünf Hauptabschnitte. Der erste Abschnitt stellt dabei Grundlagen und Konzepte von Usenet-News sowie des Agentensystems Mole vor, auf denen diese Arbeit aufbaut. Ein weiterer Abschnitt befaßt sich mit dem Entwurf der Suchmaschine. Sein Aufbau orientiert sich an den zu lösenden Teilaufgaben, die sich aus den Anforderungen an das System ergaben. In diesem Abschnitt werden Verfahren und Spezifikationen zur Lösung dieser Teilaufgaben dargestellt.

Es schließt sich ein Abschnitt an, in dem die Implementierung beschrieben wird. Die Darstellung orientiert sich dabei sowohl an der konzeptionellen Struktur des Systems, als auch an den Notwendigkeiten, die sich während der Phase der Implementierung ergaben.

Der vierte Abschnitt beschreibt einige mit der Agenten-Suchmaschine durchgeführte Tests und befaßt sich mit dem Vergleich des Systems mit existierenden herkömmlichen Suchmaschinen. Dabei wird versucht, Aussagen über die Performanz der verschiedenen Ansätze zu machen.

Im fünften Abschnitt schließlich erfolgt die Zusammenfassung der Ergebnisse. Hierin sind die, nach Meinung des Verfassers, interessantesten und aufschlußreichsten Erkenntnisse enthalten. Der Ausblick auf zukünftige Entwicklungen resultiert aus diesen Erkenntnissen und soll zu weiterführenden Studien in diesem Themenkomplex und darüber hinaus anregen.

Inhalt

Einführung	1
Verwendete Schriften und Zeichen.....	2

1 Grundlagen

1.1 Usenet-News	4
Verteilung von News.....	4
Informationsquellen	5
1.1.1 Format von Artikeln	6
Regeln für das Simple Header Parsing	7
MIME Erweiterungen	8
Encoded Words	8
Codierungsverfahren	9
Das Quoted-Printable-Verfahren	9
Verwendung von Encoded-Words	9
1.1.2 Zugriff auf Gruppen und Artikel	9
1.2 Das Agentensystem Mole	11
1.2.1 Mobile Agenten	11
1.2.2 Die Architektur von Mole	13
Agenten	14
Grundfunktionalität.....	15
Kommunikationsmechanismen.....	15
Initialisierung des Systems	16

2 Entwurf

2.1 Entwurf der Suchmaschine	18
2.1.1 Spezifikation der Suchmuster.....	18
2.1.2 Realisierung der Newsserver-Dienste	19
2.1.3 Entwurf des Suchverfahrens	20
Sequentielle Suche.....	20
Parallele Suche.....	21
Vergleich.....	22
Terminierung.....	22
Weiterführende Gedanken.....	23
2.1.4 Entwurf der Benutzerschnittstelle	24
2.2 Vertiefung: Spezifikation und Auswertung der Suchmuster	25
2.2.1 Anforderungen.....	25
2.2.2 Pattermatching	25
Jokerzeichen.....	25
Reguläre Ausdrücke	26
Verfahren zum Pattermatching	27

Schlußfolgerungen.....	28
Spezifikation regulärer Ausdrücke	29
2.2.3 Suchausdrücke für Gruppennamen.....	30
2.2.4 Suchausdrücke für Artikel.....	31

3 Implementierung

3.1 Vorbetrachtungen	34
Struktur des Systems.....	34
Gliederung	35
3.2 Grundlegende Funktionalität	35
Einführung der Klasse XtBitSet	35
Sortierfunktionalität für Vector	36
Debug-Funktionen	36
Eine Vector-Klasse für Integer-Zahlen.....	37
3.3 Reguläre Ausdrücke	37
3.3.1 Verwendung des Packages mole.TEW.Lexing	37
Beispiel	38
3.3.2 Intern verwendete Algorithmen und Datenstrukturen	39
3.4 Suchausdrücke	40
3.4.1 Verwendung	40
3.4.2 Interne Realisierung	40
Der EqualsStringMatcher.....	40
Der ContainsStringMatcher	41
3.5 News-Anbindung	42
3.6 Vereinfachter Aufruf von RPCs	45
Problematik.....	45
Lösung	46
3.7 Der Gateway-Agent für Newserver-Dienste.....	47
3.8 Ein einfacher Trading-Systemagent.....	48
Konfiguration.....	48
Integration in das System	48
3.9 Sequentielle und Parallele Suche.....	49
3.9.1 Allgemeines	49
3.9.2 Kommentierter Quellcode.....	49
Initialisierung der Suche durch den SearchController	49
Start und Migration des ParNewsSearchAgenten	51
Das Suchverfahren des NewsSearchAgenten	52
Implementierung der abstrakten Methoden des Suchagenten	54
Die RPC-Methoden des SearchControllers	54
3.10 Die Benutzerschnittstelle	55

4 Tests und Vergleiche der Suchmaschinen

4.1 Test der Agenten-Suchmaschine	58
Testumfang und Bedingungen.....	58
Testergebnisse und Bewertung	58
Schlußfolgerungen.....	59

4.2 Klassische Suchmaschinen.....	59
4.2.1 Vorbemerkungen	59
4.2.2 AltaVista und DejaNews	60
Funktionalität	60
Vergleich von Hard- und Software.....	60
Menge und Aktualität der verfügbaren Daten	61
4.3 Performanz der klassischen Suchmaschinen.....	62
4.3.1 Kriterien	62
4.3.2 Zeitnahme AltaVista - DejaNews.....	63
4.3.3 Zusammenfassung der Ergebnisse.....	63
4.4 Theoretische Betrachtungen	64
4.4.1 Algorithmus der Agenten-Suchmaschine	64
4.4.2 Ein möglicher Suchmaschinen-Algorithmus	64
Datenstruktur	64
Auswertung einer Suchanfrage	65
Schlußfolgerung.....	65
4.4 Klassische Suchmaschinen vs. parallel suchende Agenten	66

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung und Bewertung	68
Vor- und Nachteile	68
Mobile-Agenten-Systeme	68
Erweiterungsmöglichkeiten.....	69
5.2 Ausblick.....	69
Beschränkte Konzepte.....	69
Intelligente Agenten	70
Weitere Forschungen	71
5.3 Fazit.....	71

Anhang

A Eine erweiterte BNF-Notation.....	74
A.1 Vorbetrachtungen	74
A.2 Die Notation	74
Alternative.....	74
Zusammenfassung	74
Wiederholung.....	75
Optionalität.....	75
Grammatik 1: Simple Header Parsing	75
Grammatik 2: Encoded-Word	75
Grammatik 3: Regulärer Ausdruck	76
Grammatik 4: Qualifizierender Ausdruck für Gruppennamen.....	76
Grammatik 5: Qualifizierender Ausdruck für Artikel.....	77
B Test der Agenten-Suchmaschine	78
B.1 Konfiguration.....	78
Engine und Locations	78

Agenten der Locations.....	79
B.2 Parallele Suche – Protokoll	80
C Verzeichnis der Abbildungen.....	83
D Literaturverzeichnis	84
Vorbemerkung	84
Literaturhinweise.....	84

Einführung

Usenet-News ist ein elektronisches Informationssystem, mit dem weltweit Menschen in sogenannten Gruppen (einer Art elektronischem schwarzen Brett) über ein breites Spektrum an Themen diskutieren und Informationen austauschen. Ursprünglich im universitären Umfeld entwickelt und für den Bereich technisch-wissenschaftlicher Informationen gedacht, decken Usenet-News heutzutage fast die Gesamtbreite menschlicher Interessengebiete ab.

Jeder Anwender, der an das Internet angeschlossen ist, kann an diesem Austausch von Informationen teilnehmen, vorausgesetzt er besitzt die entsprechende Software. Mit dieser Software ist er in der Lage eine Mitteilung, einen sog. Artikel, in einer bestimmten Gruppe zu veröffentlichen. Dazu sendet er diese Nachricht einem Newsserver, einem Rechner, auf dem eine spezielle Software installiert ist, deren Aufgabe die Verwaltung und Verteilung von Usenet-News ist.

Das dabei anfallende Datenvolumen ist gewaltig. Schätzungen variieren beim täglichen Datenaufkommen zwischen 500 MB und 778 MB, bei der Anzahl der existierenden Gruppen zwischen 15.000 und 20.000 (s. [DJN_NET], [CALC97], Stand Frühjahr 1997). Und damit nicht genug, es wird von einem jährlichen Anstieg des Datenvolumens von 100% ausgegangen (s. [CALC97]).

Diese stetige Zunahme des täglich anfallenden Informationsvolumens in Usenet-News haben dazu geführt, daß es einem einzelnen Menschen schon lange nicht mehr möglich ist, sich einen Überblick über Themen und Inhalte durch das Lesen aller Artikel aus allen Gruppen zu verschaffen. Natürlich ist es unsinnig, anzunehmen, ein einzelner Mensch würde sich für das gesamte Themenspektrum von Usenet-News interessieren, das sich von den wissenschaftlich anspruchsvollsten bis zu den trivialsten menschlichen Themen erstreckt. Davon abgesehen stellt es jedoch heutzutage schon ein Problem dar, die Diskussionen und dargebotenen Informationen in einigen wenigen Gruppen zu verfolgen.

Um dieser Problematik zu begegnen, wurden verschieden Ansätze entwickelt. So gibt es Programme, die auf dem Rechner eines Anwenders arbeiten und Artikel nach gewissen Regeln "filtern" können. Es ist z.B. eine beliebte Methode, sich mit Hilfe dieser Programme automatisch alle Artikel eines bestimmten Verfassers herausfiltern und entfernen zu lassen, weil dieser sich in der Vergangenheit nach Empfinden des Anwenders durch besonders unqualifizierte Artikel bemerkbar gemacht hat.

Vielfach möchte ein Anwender aber nicht die Diskussionen in verschiedenen Gruppen verfolgen, die seine Interessen möglicherweise berühren könnten (als Entwickler von Macintosh-Software z.B. die Gruppe "comp.sys.mac.programmers"), da er sich dann regelmäßig durch Lesen der Artikel einen Überblick verschaffen müßte, sondern er möchte aufgrund eines gerade aufgetretenen Problems nach bestimmten Artikeln suchen, die sich möglicherweise mit diesem Problem schon befaßt haben.

Zu diesem Zweck existieren sogenannte Suchmaschinen, Rechner mit spezieller Software, die eine Recherche nach bestimmten Artikeln ermöglichen. Auf diesen Rechnern werden für einen gewissen Zeitraum alle Artikel aus einer bestimmten Menge von Gruppen (auf einigen Systemen sogar aus allen Gruppen) archiviert und speziell aufbereitet, damit sie dann einer Recherche zur Verfügung stehen. Eine Suchmaschine ist in der Regel an das Internet angeschlossen und ermöglicht so allen Internet-Teilnehmern den Zugriff auf ihre Recherchefunktionen. Der Zugriff erfolgt dabei gewöhnlich über das World-Wide-Web mit einem sog. Web-Browser. Auf dem lokalen Rechner des Anwenders wird mit dem Web-Browser Kontakt zu einer Suchmaschine aufgenommen. Der Anwender kann dann in einem Formular, das der Browser anzeigt, seine Suchanfrage formulieren und sie an die Suchmaschine abschicken. Diese führt die Recherche aus und liefert dann das Ergebnis in einem Format, das der Browser darstellen kann. Es handelt sich demnach um eine klassische Client/Server-Anwendung.

Im Bereich der Entwicklung von Client/Server-Anwendungen, insbesondere bei der Programmierung von Verteilten Systemen, wird seit einiger Zeit ein neuer Ansatz diskutiert, die sog. Mobilen Agenten (s. z.B. [HCK95]). Mobile Agenten sind Programme, die innerhalb einer bestimmten Ablaufumgebung, dem Agentensystem, ausgeführt werden und die Fähigkeit besitzen, zwischen verschiedenen Ablaufumgebungen (und damit in der Regel zwischen verschiedenen Rechnern) zu migrieren.

Eines der Gebiete, für die Mobile Agenten eingesetzt werden sollen, ist das Beschaffen von Informationen ("Information Retrieval") in heterogenen Systemen (s.S. 8f [HCK95]¹). Darunter kann man einen Verbund unterschiedlicher² miteinander vernetzter Rechner verstehen, von denen einige Informationen bereitstellen. Mobile Agenten scheinen sich für diese Aufgabe besonders gut zu eignen. In der Regel ist die zu untersuchende Datenmenge sehr groß, das gewünschte Ergebnis einer Recherche stellt aber meist nur einen sehr kleinen Ausschnitt daraus dar. Ein klassischer Ansatz würde vorsehen, die gesamte Datenmenge zum Rechner des Anwenders zu übertragen, dort mit einem Programm nach den gewünschten Daten zu suchen und die Ergebnisse dann dem Anwender zu präsentieren. Es scheint viel sinnvoller, diese Aufgabe einem Mobilien Agenten zu übertragen. Dieser erhält vom Anwender Anweisungen, wonach er suchen soll, daraufhin migriert er zu einer oder mehreren Datenquellen (d.h. Rechner, auf denen entsprechende Informationen zur Verfügung stehen), führt dort seine Recherche aus, kehrt anschließend zum Anwender zurück und präsentiert diesem die Ergebnisse. Damit wird der Kommunikationsaufwand zwischen den Rechnern stark verringert, da nicht mehr die Gesamtdatenmenge übertragen werden muß, sondern nur noch der Agent und die Ergebnisse seiner Recherche.

Newsserver sind verteilt, vernetzt und von heterogener³ Struktur. Für die Recherche nach Artikeln in Usenet-News stellen sie die Datenquellen dar. Die Aufgabe paßt daher gut in das beschriebene Einsatzgebiet Mobiler Agenten. Deshalb soll im Rahmen dieser Studienarbeit eine Suchmaschine implementiert werden, die Mobile Agenten als Basis ihrer Infrastruktur verwendet. Dabei soll untersucht werden, wie sich dieser Ansatz von den "klassischen" Suchmaschinen, die bisher in diesem Bereich eingesetzt wurden unterscheidet und welche Vorteile und möglicherweise auch Nachteile er mit sich bringt.

Verwendete Schriften und Zeichen

In dieser Arbeit werden drei verschiedene Schriftfamilien verwendet.

- Helvetica, eine serifenlose Schrift für die Überschriften.
- Palatino, eine serifenbetonte Schrift für den Fließtext.
- Courier, eine nicht-proportionale Schrift für Klassennamen, Methodennamen, Codeauszüge und Syntaxdarstellungen

¹ Zwar liegt der Fokus in diesem Papier diesbezüglich mehr auf dem Einsatz Intelligenter Mobiler Agenten, das dort Gesagte gilt aber größtenteils auch für einfache Mobile Agenten.

² Die Systeme dürfen sich sowohl in der Hardware als auch der darauf installierten Software unterscheiden. Wichtig ist, daß sie in irgendeiner Form miteinander kommunizieren können und im Falle der Mobilien Agenten muß jedes System eine Ablaufumgebung für diese bereitstellen. Es ist aber durchaus möglich, daß nicht einmal alle Systeme das gleiche Kommunikationsprotokoll verwenden, solange es Systeme gibt, welche die Verbindung zwischen diesen Protokollen herstellen können (Stichwort Gateways; dieser Gedanke wird z.B. in [HOHL95], S.13 kurz erwähnt).

³ Neben dem offensichtlichen Aspekt unterschiedlicher Hard- und Software darf man die Verwendung sog. lokaler Gruppen nicht vergessen. Dabei handelt es sich um Gruppen die nicht weltweit unter den miteinander vernetzten Newsservern ausgetauscht werden, sondern nur lokal auf einem oder mehreren Servern gehalten werden (z.B. "stuttgart.misc", eine Gruppe in der Informationen über die Stadt Stuttgart ausgetauscht werden). Des weiteren führt nicht jeder Newsserver dieselbe Menge an Gruppen. Dies führt dazu, daß die Newsserver auch als Datenquellen eine heterogene Struktur bilden.

1 Grundlagen

1.1 Usenet-News

Wie schon in der Einführung erwähnt ist Usenet-News (im folgenden auch einfach als Newssystem bezeichnet) ein System zum weltweiten Austausch von Informationen.

Im Gegensatz zu elektronischer Post, wo eine Nachricht einen oder mehrere bestimmte Empfänger hat, wird eine Nachricht im Newssystem an eine oder mehrere bestimmte Gruppen gerichtet und kann dann von allen an das Newssystem angeschlossenen Benutzern gelesen werden. Der Informationsaustausch findet also öffentlich statt.

Eine Gruppe kann man sich also als eine Art schwarzes Brett vorstellen. Nachrichten, die dort "angeheftet" werden, können von allen anderen Nutzern des schwarzen Bretts gelesen werden. Diese Nachrichten werden auch Artikel genannt.

Zudem ist es möglich, eigene Nachrichten bzw. Antworten und Kommentare zu Nachrichten anderer Teilnehmer dort zu hinterlassen. Analog eines schwarzen Brettes gibt es auch beim Newssystem bestimmte Mechanismen und Übereinkünfte, um Nachrichten nach einer bestimmten Zeit zu löschen ("expiration"; s. z.B. [INN96], insbesondere die Manpages zu "expire") sowie Nachrichten vorzeitig zu entfernen ("canceln" von Nachrichten; s.S. 12 [RFC1036]). Da diese Mechanismen für die Problemstellung der vorliegenden Studienarbeit keine bzw. nur geringe Relevanz besitzen, sei hier auf die entsprechende Literatur verwiesen. Primär von Bedeutung ist für diese, daß Nachrichten im Newssystem als Artikel in bestimmten Gruppen erscheinen. Es ist also im weiteren Verlauf insbesondere zu klären, welche Formate für Artikel und Gruppen definiert wurden.

Verteilung von News

Das Internet ist, wie der Name schon andeutet, ein Netzwerk, das aus vielen verschiedenen Rechnern gebildet wird. Dieses Netzwerk bildet die Grundlage für Usenet-News, um Nachrichten zu verteilen (s. dazu S. 18ff [RFC1036], "The News Propagation Algorithm"). Nur ein geringer Teil der Rechner, die dieses Netzwerk bilden, haben die Aufgabe, News bereitzustellen und zu verteilen (sog. Newsserver); viele andere sind aber an der Verteilung von News beteiligt, da über sie die Kommunikation zwischen den verschiedenen Newsservern abgewickelt wird.

Wenn ein Anwender einen Artikel veröffentlicht, dann gibt er an, in welcher Gruppe (auch mehrere sind möglich) dieser erscheinen soll und überträgt ihn dann mit einer speziellen Software an einen Newsserver. Dieser verteilt den Artikel dann, indem er ihn an benachbarte⁴ Newsserver weitersendet. Ebenso empfängt ein Newsserver von seinen Nachbarn entsprechend Artikel, die ein Anwender an irgendeinen anderen Server übertragen hat. Dabei kann für jeden Newsserver spezifiziert werden, welche Gruppen ihn "interessieren". Er bekommt also nicht Artikel aus allen Gruppen zugesendet, sondern nur aus einer Untermenge, die für ihn definiert wurde. Viele Newsserver stellen daher eine möglicherweise recht unterschiedliche Menge von Gruppen ihren Nutzern bereit. Dies führt dazu, daß die Datenquellen für Usenet-News auch vom Datenbestand her eine sehr heterogene Struktur besitzen können, was eine Motivation für den Einsatz Mobiler Agenten ist, da diese eine solche Struktur besonders gut unterstützen.

⁴ Es handelt sich hier um eine logische Beziehung. Geographisch können "benachbarte" Systeme durchaus sehr weit voneinander entfernt sein. In der Regel werden als Nachbarn Systeme verwendet, mit denen besonders effizient kommuniziert werden kann.

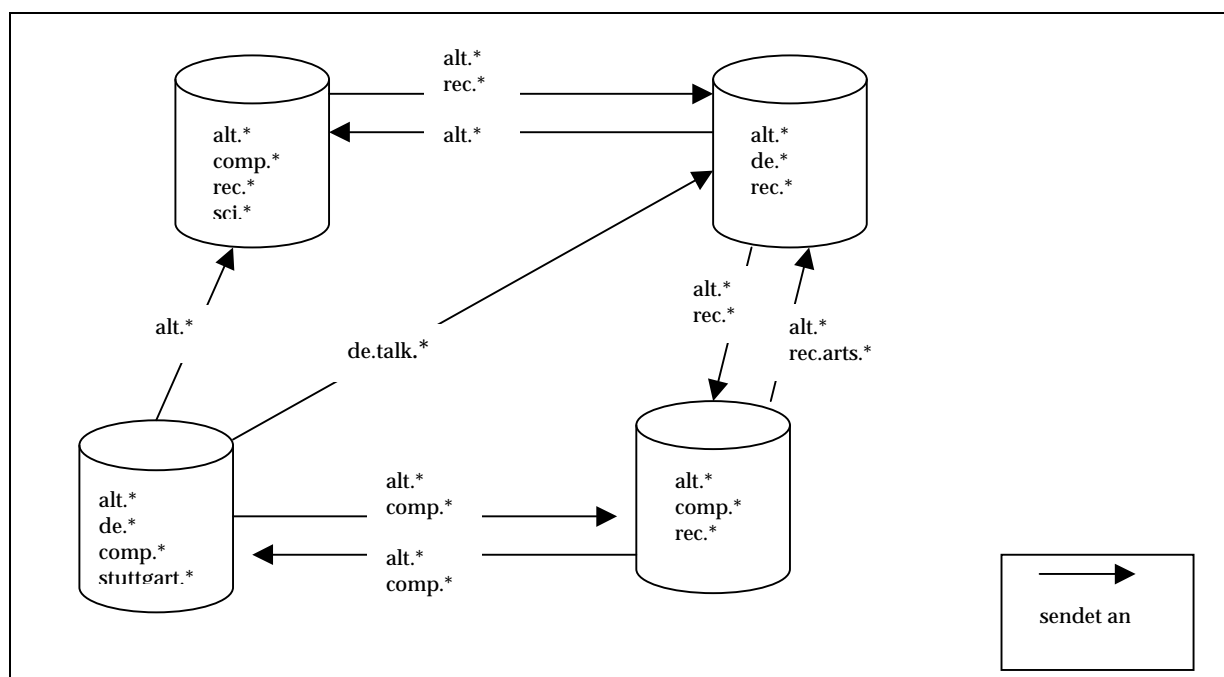


Abb. 1: Verteilung von News

Informationsquellen

Das Newssystem wird mit seinen Datenformaten, Protokollen und Verfahren in einer Reihe sogenannter Request for Comments (im folgenden mit RFC abgekürzt) beschrieben. Hierbei handelt es sich um eine gängige Vorgehensweise, mit der das Internet betreffende neue technische Verfahren oder Erweiterungen vorhandener Verfahren vorgeschlagen werden. Ein RFC gilt bei seiner Veröffentlichung nur als Vorschlag, der dann von allen an das Internet angeschlossenen Teilnehmern diskutiert werden kann (daher Request for Comments). Im Anschluß an diese Diskussion kann ein RFC angenommen oder verworfen werden, bzw. mit den in diesem Prozeß für sinnvoll erachteten Änderungen erneut zur Diskussion gestellt werden. Die RFCs, die das Newssystem beschreiben, sind zum großen Teil schon seit Jahren als technische Grundlage akzeptiert, die darin beschriebenen Verfahren durch eine Reihe von Softwarepaketen implementiert.

Die für diese Studienarbeit herangezogenen RFCs sind:

[RFC1036] "Standard for Interchange of USENET Messages"

Dieses Dokument beschreibt das Standardformat für den Austausch von News-Nachrichten und einige Teilstandards für die Übertragung dieser Nachrichten.

[RFC 822] "Standard for ARPA Internet Text Messages"

Definiert das Format von Textnachrichten. Diese werden gewöhnlich für elektronische Post verwendet ("Mail"), allerdings baut auch RFC 1036 auf den hier definierten Standards auf.

- [RFC1521] *"MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies"*
- [RFC1522] *"MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text"*

Beide Dokumente definieren Erweiterungen des in [RFC822] spezifizierten Formats.

- [RFC977] *"Network News Transfer Protocol"*

Dieser RFC definiert ein Protokoll für den Austausch von Usenet-News.

1.1.1 Format von Artikeln

[RFC1036] definiert das Standardformat für den Austausch von News-Nachrichten und einige Teilstandards für die Übertragung dieser Nachrichten. Das Nachrichtenformat baut dabei auf dem in [RFC822] definierten Format für Internet Mail-Nachrichten auf, wobei die in diesem erlaubte Flexibilität durch einige Übereinkünfte eingeschränkt wird, um eine einfachere Verarbeitung zu ermöglichen⁵.

Grundlage des Mail-Nachrichtenformats ist die Übersetzung der Umschlag/Inhalt-Metapher in eine elektronische Darstellung. Jede Nachricht besteht aus einem Header, analog eines Briefumschlages, der unter anderem die Adresse des Empfängers und den Absender spezifiziert sowie einem Inhalt (Body). Dabei wird keinerlei Definition vorgenommen, woraus dieser Inhalt bestehen soll.

Eine Nachricht besteht aus einer Folge von Textzeilen. Text wird nach dem ASCII-Standard kodiert. Jede Nachricht beginnt mit einer Folge von Zeilen, die den Header bilden. Diesem kann eine Leerzeile folgen, an die sich der Body als Folge weiterer Textzeilen anschließen kann. Ein Header besteht aus Fields, die aus einem Field-Name und einem Field-Body bestehen.

Beispiel eines Artikels	
<pre>From: jerry@eagle.ATT.COM (Jerry Schwarz) Path: cbosgd!mhuxj!mhuxt!eagle!jerry Newsgroups: news.test Subject: Dies ist ein Test Message-ID: <642@eagle.ATT.COM> Date: Fri, 19 Nov 82 16:14:55 GMT Followup-To: news.misc Expires: Sat, 1 Jan 83 00:00:00 -0500 Organization: AT&T Bell Laboratories, Murray Hill</pre>	Header
<p>Nach einer Leerzeile kommt hier der Text, der sogenannte Body.</p>	Body

⁵ "A standard USENET message consists of several header lines, followed by a blank line, followed by the body of the message. Each header line consists of a keyword, a colon, a blank, and some additional information. This is a subset of the Internet standard, simplified to allow simpler software to handle it. The 'From' line may optionally include a full name, in the format above, or use the Internet angle bracket syntax. To keep the implementations simple, other formats (for example, with part of the machine address after the close parenthesis) are not allowed. The Internet convention of continuation header lines (beginning with a blank or tab) is allowed." (s.S. 2f [RFC822])

In [RFC822] werden detaillierte syntaktische Regeln aufgestellt, denen der Header, insbesondere aber die Struktur einiger spezieller Field-Bodies genügen müssen. Diese Regeln müssen bei der Erzeugung und Weiterleitung von Nachrichten genau befolgt werden.

Im Rahmen dieser Studienarbeit soll nur auf vorhandene Artikel zugegriffen und keine neuen Artikel erzeugt werden. Zur Qualifikation eines Artikels wird nur das Subject-Field in dessen Header herangezogen. Es können daher einige vereinfachende Grundannahmen vorgenommen werden. So kann davon ausgegangen werden, daß ein zu Verfügung stehender Artikel einen korrekten syntaktischen Aufbau besitzt. Die Autoren von [RFC822] haben den Bedarf nach einer vereinfachten Verarbeitung von Nachrichten vorhergesehen und eine verkürzte Menge syntaktischer Regeln hierfür angegeben (s.S. 39ff, SIMPLE HEADER PARSING [RFC822]). Diese sollen im folgenden ausführlich dargestellt werden.

Regeln für das Simple Header Parsing

Die Darstellung der Syntax erfolgt in einer erweiterten BNF-Notation. Diese wird in Anhang A erläutert.

Grammatik: Simple-Header-Parsing	
Message	= *Field *(CRLF *text)
Field	= Field-Name ":" [Field-Body] CRLF
Field-Name	= 1*<jedes Zeichen, außer Steuerungszeichen, SPACE und ":">
Field-Body	= *Text [CRLF LWSP-char Field-Body]

Der Header erscheint vor dem Body der Nachricht und wird von diesem durch zwei aufeinanderfolgende CRLF (die ASCII-Zeichen für Carriage-Return und Linefeed), also eine Leerzeile, getrennt.

Der Header besteht aus einzelnen Fields. Ein Field setzt sich aus einem Field-Name gefolgt von einem ":" sowie dem Field-Body zusammen.

Wichtig ist, daß der Field-Name keinen ":", SPACE oder Steuerungszeichen enthält. Ein Field-Name darf sich im Gegensatz zum Field-Body nicht über mehrere Zeilen erstrecken.

Der Field-Body kann sich aus mehreren Zeilen zusammensetzen, wobei die erste Zeile mit dem Field-Name beginnt und jede folgende Zeile mit einem SPACE- oder HTAB-Zeichen.

Die sich über mehrere Zeilen erstreckende Repräsentation eines Fields kann für die interne Verarbeitung zu einer Darstellung, die sich aus einer Zeile zusammensetzt, zusammengefaßt werden. Dieses Verfahren wird als "unfolding" bezeichnet ([RFC822, s.S.4f]). Dabei werden aufeinanderfolgende Zeilen eines Field-Bodys zusammengefaßt, indem ein CRLF mit direkt anschließendem SPACE- oder HTAB-Zeichen als äquivalent zu einem SPACE- bzw. HTAB-Zeichen betrachtet wird. In anderen Worten, die aufeinanderfolgenden Zeilen werden durch Entfernen der CRLF-Zeichen zusammengefaßt.

MIME Erweiterungen

In [RFC822] wurde als einzig zulässige Zeichenmenge zum Austausch von Nachrichten der ASCII-Zeichensatz zugelassen. Dabei handelt es sich um eine 7-Bit-Zeichenkodierung. Bedingt durch die heterogene Struktur des Internets mit einer Vielzahl unterschiedlicher, miteinander verbundener Rechnersysteme und regionale Unterschiede, zeigte sich recht schnell, daß diese Kodierung in vielen Fällen nicht ausreichend ist. Davon abgesehen, daß der ASCII-Zeichensatz die Verwendung lateinischer Schriftzeichen voraussetzt und somit viele Menschen in der Welt nicht in der Lage sind, Texte in der Zeichenmenge ihrer Muttersprache zu übermitteln, ist es mit ihm noch nicht einmal möglich, die im Deutschen verwendeten Umlaute zu übertragen oder die im Französischen verwendeten Accents. Dies ist einer der Gründe warum recht schnell der Bedarf nach einer Erweiterung des bisherigen Standards entstand⁶.

In [RFC1521] und [RFC1522] wurden Erweiterungen des Internet-Mail-Nachrichtenformates definiert. MIME steht dabei für "Multipurpose Internet Mail Extensions". Es handelt sich dabei um Erweiterungen des bisherigen Standards für "ARPA Internet Text Messages" ([RFC822], die dazu dienen, zusammengesetzte Dokumente mit darin eingebetteter Grafik, Ton oder anderen Inhalten zu realisieren. Des weiteren werden Verfahren definiert, mit denen Texte im Header und im Body einer Nachricht, die in anderen Zeichenmengen als ASCII kodiert wurden, übertragen werden können. Dabei befaßt sich [RFC1521] im wesentlichen mit dem Body einer Nachricht, [RFC1522] definiert dagegen die Erweiterungen, die notwendig sind, um Text im Header, der in anderen Zeichenmengen als ASCII kodiert wurde, zu übertragen.

Da in dieser Studienarbeit die Qualifikation von Artikeln nur über das Subject-Field erfolgen soll, stützt sie sich im wesentlichen auf die in [RFC1522] vorgestellte Erweiterungen. Eine Qualifikation eines Artikels über den im Body enthaltenen Text ist denkbar, erfordert jedoch unter Berücksichtigung der MIME-Erweiterungen einen nicht zu vernachlässigenden Aufwand.

Encoded Words

Prinzipiell galt es beim Entwurf der MIME-Erweiterungen, ein Verfahren zu finden, Zeichen aus einer von ASCII verschiedenen Zeichenmenge so zu kodieren, damit eine weltweite Übertragung von Nachrichten möglich ist, ohne daß die daran beteiligte Software, die nur das in [RFC822] definierte Format unterstützt, verändert werden mußte. Dieses Ziel konnte durch die Entscheidung erreicht werden, solche Zeichen in einer geeigneten Kodierung durch ASCII-Zeichen zu repräsentieren. Der dafür vorgesehene Mechanismus definiert ein sog. Encoded-Word. Dabei handelt es sich um eine Zeichenfolge, die folgenden Struktur besitzt:

Encoded-Word = "=? Charset "?" Encoding "?" Encoded-Text "?="

Ein Encoded-Word besteht also aus der Spezifikation einer Zeichenmenge, eines Kodierungsverfahrens und den kodierten Zeichen. Dabei dürfen die zur Angabe von Charset und Encoding verwendeten Zeichenfolgen bestimmte Sonderzeichen nicht enthalten. Dies gilt ebenfalls für den Encoded-Text (s. Anhang A Grammatik 2:Encoded Word).

Ein Charset gibt an, aus welcher Zeichenmenge die kodierten Zeichen stammen, d.h. in welche Zeichenmenge die in Encoded-Text kodierten Zeichen für eine Darstellung oder Bearbeitung dekodiert werden müssen. Der dafür verwendete Name muß bei IANA, der Internet Assigned Numbers Authority registriert worden sein. Da es möglich ist, dort neue Namen und damit neue Zeichenmengen zu registrieren, kann es vorkommen, daß in einer Nachricht eine Zeichenmenge spe-

⁶ Wesentlichen Anteil an der Erweiterung des bisherigen Standards hatte der Bedarf nach Regeln, die den Austausch von zusammengesetzten Dokumenten, die andere Dokumente, Bilder, Ton und sonstige Daten enthalten können, ermöglichen.

zifiziert wird, die nicht von jeder Software unterstützt wird. Es wird empfohlen, die durch die Folge von ISO-8859-* definierten Zeichenmengen bevorzugt zu verwenden, wenn es die Möglichkeit gibt ein Zeichen in verschiedenen Zeichenmengen darzustellen⁷.

Codierungsverfahren

Encoding stellt die Bezeichnung eines Kodierungsverfahrens dar. RFC1522 erlaubt hier zwei Verfahren, zum einen das durch "B" bezeichnete BASE64-Verfahren, zum anderen das mit "Q" bezeichnete sogenannte Quoted-Printable-Verfahren.

In der Implementierung der Studienarbeit wurde bisher nur das Quoted-Printable-Verfahren berücksichtigt, das in den Headern von Artikeln häufiger Verwendung findet, als das BASE64-Verfahren. Die Implementierung stellt jedoch schon die Infrastruktur bereit, in die auch die Integration anderer Verfahren, insbesondere des BASE64-Verfahrens möglich ist. Im Anschluß wird daher nur das Quoted-Printable-Verfahren dargestellt.

Das Quoted-Printable-Verfahren

Es gelten folgende 3 Regeln:

- 1) Jedes mit 8-Bit kodierte Zeichen kann durch ein "=", gefolgt von zwei hexadezimalen Ziffern (möglichst in Großbuchstaben) repräsentiert werden.
- 2) Der hexadezimale Wert 20 eines Zeichen kann durch das Zeichen "_" kodiert werden.
- 3) Durch 8-Bit kodierte Zeichen, die einem ASCII-Zeichen entsprechen, können durch ebendieses dargestellt werden.

Verwendung von Encoded-Words

Encoded-Words dürfen in Subject- oder Comments-Feldern des Headers ein Text-Token ersetzen (s. Anhang A Grammatik 1: Simple Header Parsing). Es ist auch erlaubt normalen ASCII-Text und Encoded-Words zusammen in einem Header-Field zu verwenden. Dabei ist zu beachten, daß in einem Feld, daß durch die Regel *Text definiert wurde ein Encoded-Word durch Linear-White-Space von jedem folgenden Encoded-Word oder Text getrennt werden muß⁸.

1.1.2 Zugriff auf Gruppen und Artikel

Im Rahmen dieser Studienarbeit ist geplant, einen Zugriff auf News direkt über das Filesystem eines Rechners zu implementieren. Rechner mit Ablaufumgebungen für das Mobile-Agenten-System Mole, die Zugriff auf News als einen weiteren Dienst von Mole anbieten wollen, müssen daher zusätzlich ein Newsserver-Programm (z.B. Internet News; s. [INN96]) installiert haben. Dieses Programm muß die Artikel in Dateien in bestimmten Verzeichnissen ablegen. Alternativ reicht es aus, wenn von der Ablaufumgebung der Zugriff auf Massenspeicher eines Newsservers möglich ist, die diese Daten in der beschriebenen Form halten⁹.

⁷ "When there is a possibility of using more than one character set to represent the text in an encoded-word, and in the absence of private agreements between sender and recipients of a message, it is recommended that members of the ISO-8859-* series be used in preference to other character sets." (S. 3 [RFC1522])

⁸ Für andere Felder des Headers gelten gewisse weitere Bedingungen, die jedoch für die vorliegende Studienarbeit keine Bedeutung besitzen, da wir uns auf das Subject-Field beschränken (s.S. 4ff [RFC1522]).

⁹ Unter Unix-Betriebssystemen reicht also die Möglichkeit zum "mounten" der Massenspeicher eines Newsservers als Voraussetzung aus (s. dazu die jeweilige vom Hersteller mitgelieferte Systemdokumentation).

Beim Entwurf des neuen Dienstes, der innerhalb von Mole den Zugriff auf News ermöglicht, soll jedoch eine leichte Erweiterbarkeit berücksichtigt werden, damit auch andere Verfahren des News-Zugriffs unterstützt werden können. Aufgrund dieser Überlegungen werden hier kurz die Möglichkeiten auf Artikel und Gruppen zuzugreifen dargestellt, die das Network News Transfer Protokoll (NNTP, s. [RFC977]) definiert.

NNTP definiert ein Protokoll für die Verteilung, Abfrage, den Zugriff auf und das Veröffentlichen von Artikeln. Hierbei wird ein stream-basiertes Übertragungsverfahren verwendet (z.B. TCP/IP) und von einer zentralen Speicherung der Daten auf einem Rechner ausgegangen (Client/Server-Modell). Das Protokoll sieht einen einfachen Austausch von Informationen vor, bei dem Kommandos von einem Client an den Newsserver geschickt, dort bearbeitet und schließlich Antworten mit Ergebnissen zurückgeschickt werden. Kommandos und Antworten setzen sich einfach aus ASCII-Zeichen zusammen.

Die in [RFC977] definierten Kommandos für den Zugriff auf Gruppen und Artikel bieten folgende Möglichkeiten:

- Es läßt sich eine Liste gültiger (verfügbarer) Gruppen ermitteln (s.S. 13f [RFC977], LIST-Kommando)
- Eine Gruppe, auf deren Artikel zugegriffen werden soll, kann spezifiziert werden (s.S. 10 [RFC977], GROUP-Kommando)
- Es ist ein Zugriff auf den gesamten Artikel, aber auch getrennt nach Header und Body möglich (s.S. 8ff [RFC977], ARTICLE-, BODY-, HEAD-Kommando)

1.2 Das Agentensystem Mole

1.2.1 Mobile Agenten

Im Bereich der Entwicklung von Client/Server-Anwendungen, insbesondere bei der Programmierung von Verteilten Systemen, wird seit einiger Zeit ein neues Programmiermodell diskutiert. Die Rede ist dabei von sog. Mobilen Agenten.

Mobile Agenten (im folgenden einfach mit Agenten bezeichnet) sind Entitäten aus Programmen und Daten. Charakteristisch für diese ist die Fähigkeit, ihre Ablaufumgebung wechseln - migrieren - zu können. Die folgende Abbildung stellt die Architektur eines Mobile-Agenten-Systems vereinfacht dar. Im Anschluß an diese Abbildung werden die dort dargestellten Zusammenhänge näher erläutert.

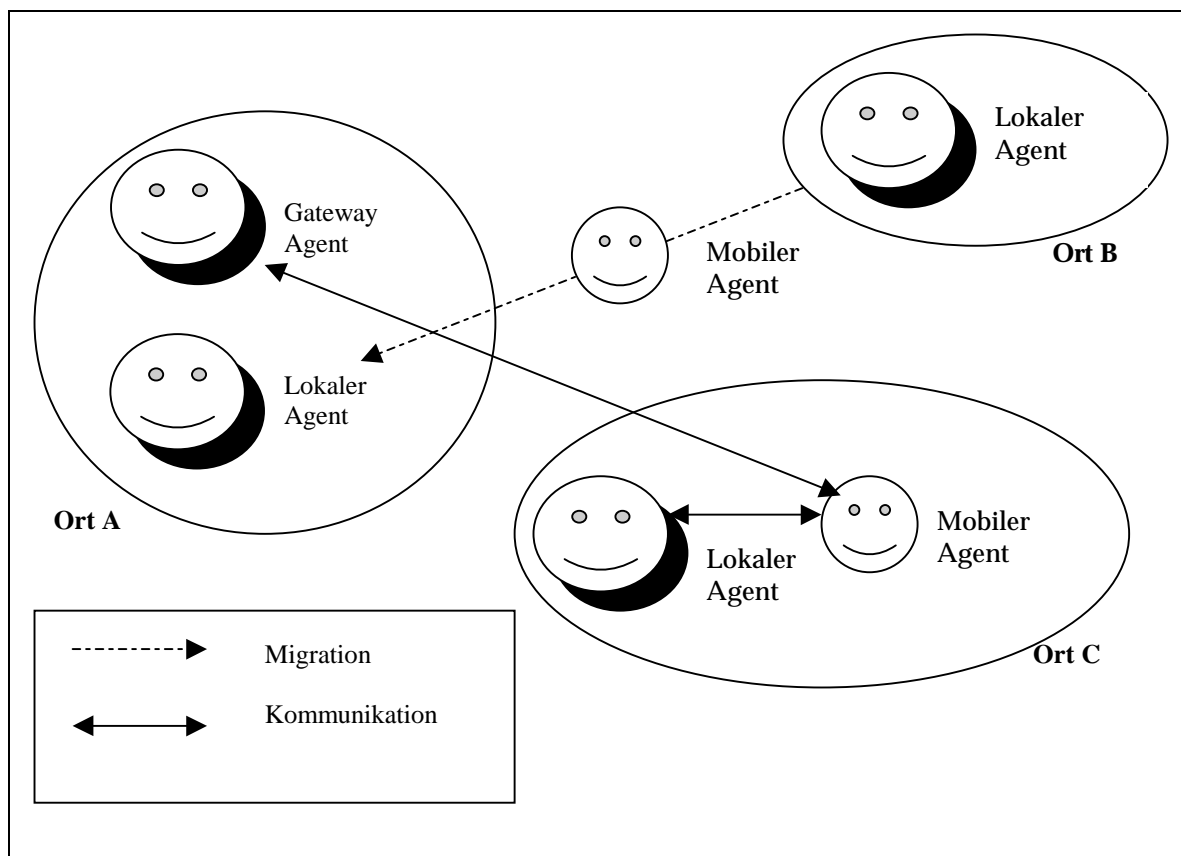


Abb. 2: Ein Mobiles-Agenten-System im Überblick

Ein Agentensystem stellt Orte zur Verfügung. Ein Ort stellt die Ablaufumgebung dar, in welcher der Agent ausgeführt wird und mit anderen Agenten interagieren kann. Ein Agent befindet sich in der Regel an einem Ort, bzw. ist auf dem Weg von einem Ort zu einem anderen.

Zwischen den auf einem Rechner vorhandenen Diensten vermitteln sogenannte Gateway-Agenten. Ein direkter Zugriff auf die genannten Dienste ist einem Agenten aus Sicherheitsgründen nicht möglich.

Jeder Ort besitzt einen Lokalen Agenten, der diesen Ort repräsentiert und dem Nachrichten geschickt werden können.

Agenten und Orte können durch Namen eindeutig identifiziert werden. Die Eindeutigkeit der Namen gilt dabei ortsübergreifend.

Agenten kommunizieren mit anderen Agenten - auch den Gateway-Agenten - über einen Nachrichtenmechanismus. Über diesen Mechanismus besteht die Möglichkeit, mit am selben Ort vorhandenen Agenten Nachrichten auszutauschen. Diese können auch an Agenten geschickt werden, die sich an anderen Orten aufhalten. Letzteres gilt jedoch als "teurer", in dem Sinne, daß hierfür mehr Ressourcen verbraucht werden.

Weiterhin können Agenten mit anderen Agenten durch den direkten Aufruf von Methoden kommunizieren (Stichwort "Remote procedure call"). Dabei dürfen sich die Agenten ebenfalls an verschiedenen Orten aufhalten.

1.2.2 Die Architektur von Mole

Mole ist die Implementierung eines Mobile-Agenten-Systems. Es basiert im wesentlichen auf der Arbeit von [HOHL95] und den inzwischen daran vorgenommenen Erweiterungen. Die zuvor erwähnten abstrakten Konzepte sog. Mobile-Agenten-Systeme wurden in Mole durch eine Reihe interagierender Klassen realisiert. Die folgende Abbildung stellt diese Zusammenhänge¹⁰ in einer vereinfachten Form dar.

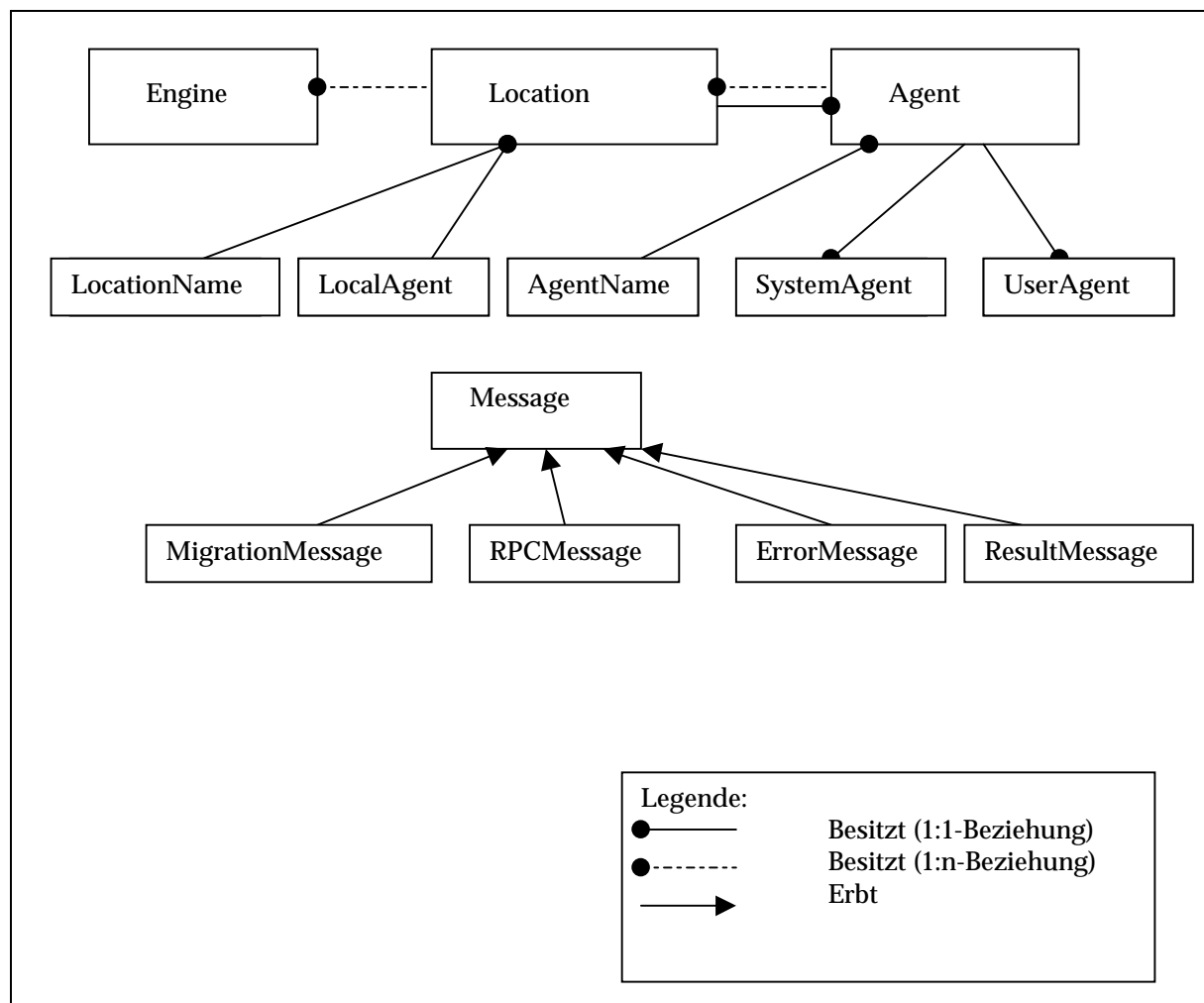


Abb. 3: Systemarchitektur von Mole

Orte werden durch Instanzen der Klasse `Location` repräsentiert. In der jetzigen Implementierung werden mehrere `Locations` von einer `Engine` betrieben. Die `Engine` dient dazu, mehrere `Locations` auf einem Rechner ressourcen-schonend zur Verfügung zu stellen (dies wird unter an-

¹⁰ Die Recherchen hierfür basieren im wesentlichen auf der Diplomarbeit von [HOHL95] sowie der Dokumentation und dem mitgelieferten Quellcode der Alpha-Release 1.0 von Mole [MOLE96]. In Zweifelsfragen wurde versucht, soweit möglich, die Dokumentation bzw. den Quellcode zu konsultieren, um eine dem aktuellen Stand entsprechende Beschreibung des Systems zu ermöglichen.

derem dadurch erreicht, daß eine `Engine` mit mehreren `Locations` von einer Java Virtual Machine ausgeführt wird statt jede `Location` von einer Virtual Machine ausführen zu lassen).

Dienste werden in Mole, wie schon erwähnt, durch Gateway-Agenten angeboten, die von der Klasse `SystemAgent` abgeleitet wurden. Mobile Agenten werden von der Klasse `UserAgent` abgeleitet. Sowohl `UserAgent` als auch `SystemAgent` erben von einer gemeinsamen Basisklasse `Agent`.

Namen wurden auf zwei unterschiedliche Weisen implementiert. Für `Locations` wurde die Klasse `LocationName` geschaffen, analog für `Agents` die Klasse `AgentName`. `LocationNames` kapseln DNS-Namen, `AgentNames` besitzen eine eigene Struktur. Die Klasse `Location` implementiert das Interface `NIC` und ihre Instanzen können dadurch jederzeit neue, noch nicht verwendete eindeutige Namen für neu erzeugte Agenten liefern.

Nachrichten werden durch die Basisklasse `Message` und davon abgeleitete Klassen repräsentiert und mit Hilfe einer `Location` verschickt.

Agenten

In der Basisklasse `Agent` werden einige Methoden definiert, die neu implementierte Klassen, seien es von `UserAgent` abgeleitete Mobile Agenten oder von `SystemAgent` abgeleitete Gateway-Agenten, für ihre Zwecke überladen können. Diese Methoden dienen als Callbacks, die vom System aufgerufen werden, bevor bestimmte Ereignisse eintreten bzw. nachdem diese eingetreten sind. Somit erhält ein Agent die Möglichkeit, auf diese Ereignisse zu reagieren indem er beispielsweise entsprechende Zustandsänderungen vornimmt.

Basismethoden der Klasse Agent	
Methode	Beschreibung
start	wird aufgerufen, wenn ein Agent erzeugt und in das System integriert wurde bzw. nachdem ein Agent zu einer neuen <code>Location</code> migriert ist
stop	wird aufgerufen, bevor ein Agent gestoppt (z.B. wegen bevorstehender Migration) und gespeichert wird
end	wird aufgerufen, bevor ein Agent aus dem System entfernt wird
heartbeat	wird periodisch vom System aufgerufen
receiveMessage	wird aufgerufen, wenn das System eine Nachricht an den Agenten weiterleitet
dispatch	wird aufgerufen, wenn der Agent einen RPC ausführen soll (der Aufruf eines RPC ist in Mole auf besondere Weise gelöst worden)

Grundfunktionalität

Instanzen der Klasse `Location` stellen den Agenten die grundlegende Funktionalität eines Mobile-Agenten-Systems zur Verfügung.

Im wesentlichen gliedert sich diese in folgende Gebiete:

- Erzeugen und Terminieren von Agenten
- Verwalten von Agenten
- Erzeugen von eindeutigen Agentennamen
- Verwalten periodischer Aufrufe von Agentenmethoden (heartbeat-Methode)
- Migration von Agenten
- Kommunikationsmechanismen
- Vermitteln von Diensten innerhalb der Location

Kommunikationsmechanismen

Wie schon erwähnt, erfolgt die Kommunikation zwischen Agenten, seien es Mobile Agenten oder Gateway-Agenten über den Austausch von Nachrichten, dargestellt durch die Klasse `Message`.

`Messages` enthalten die eindeutige Adresse von Sender und Empfänger (gebildet durch jeweils ein Paar aus `AgentName` und `LocationName`) sowie ein Objekt, das den Inhalt der Nachricht repräsentiert. Sie werden von einer Instanz der Klasse `Location` verschickt. In Mole gibt es keine Komponente, die Auskunft über den jeweiligen Aufenthaltsort eines Agenten geben kann, daher ist es möglich, daß eine Nachricht an einen bestimmten Agenten an eine `Location` geschickt wird, an welcher sich der Agent nicht mehr befindet.

Es wurden mehrere Mechanismen implementiert, um diese Situation zu bewältigen.

Zum einen wird beim Erzeugen einer Nachricht neben Absender, Empfänger und Inhalt noch eine Zahl übergeben, die angibt, wie zu verfahren ist, wenn die Nachricht an ihrer Ziel-`Location` nicht übermitteln werden kann, weil der als Empfänger vorgesehene Agent sich nicht dort befindet.

Diese Zahl wird, abhängig von ihrem Wert, auf drei verschiedenen Weisen ausgewertet:

0	bedeutet, es soll nichts geschehen
1	bedeutet, eine Fehlernachricht soll an den Sender zurückgeschickt werden
100 + x	bedeutet, die Nachricht soll für x Sekunden gespeichert werden (in der Alpha 1.0 Release ist der Expiration-Mechanismus jedoch nicht implementiert)

Die `Location` besitzt Methoden, um die Anzahl der gespeicherten `Messages` für einen Agenten zu ermitteln und diese an einen Agenten auszuliefern.

Des weiteren gibt es die Möglichkeit, einen Agenten als Stellvertreter eines anderen Agenten bei einer `Location` anzumelden. Dieser Stellvertreter erhält alle `Messages`, falls sich der als Empfänger vorgesehene Agent nicht mehr bei dieser `Location` aufhält und kann diese dann in beliebiger Form weiterverarbeiten.

Als Äquivalent zum Methodenaufruf wurde die Klassen `RPCMessage` implementiert. Eine `RPCMessage`, die von der Klasse `Message` abgeleitet wurde, enthält dabei als zusätzliche Informationen Methodenname und Parameter einer Methode, die der als Empfänger vorgesehene Agent ausführen soll. Als Ergebnis des Aufruf wird ein beliebiges Objekt zurückgeliefert. Auch `RPCMessages` werden durch die `Location` versendet, allerdings durch synchrone Kommunikation, im Gegensatz zur normalen `Message`, die asynchron verschickt wird, da der aufrufende Agent das Ergebnis des Aufrufs erwartet.

Jeder Agent muß die Methoden `dispatch` und `receiveMessage` implementieren, damit er `RPCMessages` bzw. normale `Messages` verarbeiten kann.

Initialisierung des Systems

Eine Ablaufumgebung von Mole wird durch Start einer Engine bereitgestellt. Wie schon erwähnt, faßt eine Engine mehrere Locations zusammen. Beim Start der Engine wertet diese eine Konfigurationsdatei aus. In dieser Datei werden Parameter der Engine, sowie der von ihr betriebenen Locations angegeben.

Im einzelnen werden dabei für eine Engine folgende Parameter angegeben:

- Die IP-Adresse des Rechners, auf dem sie betrieben wird.
- Die Nummer des Ports über den sie mit anderen Engines kommunizieren kann.
- Eine Datei, in der die Namen aller Klassen aufgeführt werden, die der Engine auf dem Rechner, auf dem sie betrieben wird, zur Verfügung stehen. Nur Klassen, die in dieser Datei eingetragen sind erhalten besondere Rechte (Zugriff auf Dateien, Kommunikation per TCP/IP). Neu erstellte Gateway-Agenten müssen auf jeden Fall in dieser Datei eingetragen werden.
- Der Pfad des Verzeichnisses, in dem alle Klassen des Mole-Packages enthalten sind.
- Der Pfad eines Verzeichnisses, in dem temporär Klassen gespeichert werden können (z.B. weil ein bei einer Location dieser Engine erschienener Mobiler Agent diese Klassen "mitgebracht" hat).

Zusätzlich werden in dieser Datei die Konfigurationsparameter aller Locations, die von dieser Engine betrieben werden sollen, angegeben.

Für jede Location werden nachstehende Parameter angegeben:

- Der Name der Location.
- Eine textuelle Beschreibung der Location.
- Der Name einer Datei, in der alle Agenten (sowohl SystemAgents als auch UserAgents), mit denen die Location nach dem Start der Engine initial belegt werden soll, gespeichert wurden. Eine solche Datei kann durch ein kleines Java Programm angelegt werden.
- Beginn und Ende eines Namensraumes, der für in dieser Location neu erzeugte Agenten zur Verfügung steht.

Zur Veranschaulichung sollen Ausschnitte aus der in der Online-Dokumentation enthaltenen Beispielkonfiguration einer Engine gezeigt werden.

Zu Beginn werden die erwähnten Engine-Parameter definiert.

```
# definition of the engine
ENGINEADDRESS 129.69.210.21
ENGINEPORT 7777
ENGINECLASSES /home/hohlfz/java/mole/bratsche/classes/mole
ENGINESYSTEMLIST systemclasses.dat
ENGINEHOME /home/hohlfz/java/mole/classes/mole
```

Daran anschließend werden die Parameter einer Location angegeben.

```
# definition of the first location
LOCATIONNAME location1.mole.informatik.uni-stuttgart.de
LOCDESCRIPTION the very first location of Mole
STARTUPFILE location1.class
NSSTART 0.0.0.0.0.0.0.1
NSSEND 0.0.0.0.0.0.255.255
```

2 Entwurf

2.1 Entwurf der Suchmaschine

Der grundlegende Entwurf der Suchmaschine ergibt sich aus den folgenden Anforderungen:

1. Der Anwender soll dem System über ein Formular Suchmuster angeben, mit denen Artikel aus bestimmten Gruppen mit bestimmten Themen (Subjects) qualifiziert werden können.
2. Die Recherche soll durch Mobile Agenten erfolgen, die zu einzelnen Newsservern migrieren und dort lokal einen Teil der Suche abarbeiten.
3. Als Ergebnis der Suche sollen die Artikel oder Referenzen auf diese geliefert werden.

Daraus ergeben sich folgende - im weiteren Verlauf zu lösende - Teilaufgaben:

- Spezifikation und Auswertung der Suchmuster
- Realisierung der Newsserver-Dienste
- Entwurf des Suchverfahrens
- Entwurf der Benutzerschnittstelle

2.1.1 Spezifikation der Suchmuster

Für die Beschreibung der Suchmuster sind vielfältige Ansätze denkbar. Deren Bandbreite soll durch Beschreibung einer sehr einfachen und einer besonders umfassenden Variante dargestellt werden.

Eine der einfachsten Varianten würde die Eingabe eines exakten Gruppennamens und die Eingabe eines Wortes erfordern, über die Artikel qualifiziert werden sollen. Der Suchvorgang, der dadurch ausgelöst würde, ließe sich wie folgt beschreiben:

"Liefere mir alle Artikel aus einer Gruppe mit Namen X, deren Subject-Zeile das Wort Y enthält."

Diese Variante ließe sich sehr einfach implementieren, bietet dem Anwender aber nur wenig Möglichkeiten für eine ausgefeilte Recherche. So wäre es sicher wünschenswert, eine Menge unterschiedlicher Gruppen angeben zu können, in denen nach Artikeln gesucht werden soll. Auch ist es in der Praxis sicher nicht ausreichend, nur ein Wort anzugeben, das in der Subject-Zeile des Artikels enthalten sein soll.

Der umfassendste Ansatz wäre die Verarbeitung einer natürlichsprachlichen Anfrage, die z.B. folgende Gestalt haben könnte:

"Liefere mir alle Artikel, die sich mit der Suche nach Artikeln in Usenet-News unter Einsatz von Agenten befassen."

Dieser Ansatz erfordert den Einsatz von Technologien wie Wissensbasierte Systeme, Natürlichsprachliche Verarbeitung, Inferenzmechanismen, Künstliche Intelligenz oder Neuronale Netzwerke. Er geht damit weit über den Einsatz Mobiler Agenten hinaus und berührt Gebiete, die in der Forschung¹¹ unter anderem unter dem Begriff Intelligente Agenten (s. z.B. [JANSEN97], [SEARCHBOTS], [IIR95]) zusammengefaßt werden. Wiewohl dieser Ansatz sicher eine der anwenderfreundlichsten Varianten darstellt, würde er doch den Umfang dieser Studienarbeit bei weitem überschreiten.

Im Rahmen dieser Arbeit soll versucht werden, mit traditionellen Methoden (Patternmatching, formal aufgebauten Suchausdrücken) ein Suchverfahren zu implementieren, das sowohl die Einschränkungen des einfachen wie die außerordentlich hohe Komplexität des umfassenden

¹¹ Unter anderem befassen sich laufende Forschungsprojekte an der Universität von Massachusetts (s. [SEARCHBOTS]) und am Institute for Information Technology in Ottawa (s. [SIGMA]) mit dem Auffinden und Filtern von Informationen basierend auf Verfahren, die Agententechnologie einsetzen.

Verfahrens vermeidet. Die vorgestellte einfache Variante soll durch eine Reihe von Erweiterungen zu einem Werkzeug ausgebaut werden, das mächtig genug ist, auch komplexere Recherchen zu formulieren.

- Es soll die Möglichkeit geschaffen werden, statt einem Namen eine Menge von Gruppennamen anzugeben. Jede Gruppe, deren Name in dieser Menge enthalten ist, muß dann nach Artikeln mit einer passenden Subject-Zeile untersucht werden.
- Statt nur das Enthaltensein eines Wortes in einer Subject-Zeile zu prüfen, darf der Anwender einen logischen Ausdruck angeben. Dieser Ausdruck gibt Worte und logische Verknüpfungen an. Kann dieser Ausdruck für eine Subject-Zeile als "wahr" ausgewertet werden, so gilt der Artikel als "passend". Ein solcher Ausdruck könnte z.B. so aussehen:
"Suche UND Usenet-News UND (Agent ODER Suchmaschine)".
Alle Artikel, welche die Worte "Suche" und "Usenet-News" enthalten sowie das Wort "Agent" oder das Wort "Suchmaschine" würden als "passend" angesehen.
- Statt Namen und Worten, die exakt gefunden werden müssen sollten flexiblere Beschreibungen ermöglicht werden. Es wäre wünschenswert, wenn man eine Art Meta-Wort zum Umschreiben einer Klasse von Worten oder Namen verwenden könnte. So würde man keinen Gruppennamen angeben, sondern könnte z.B. "alle Gruppen, deren Namen mit 'comp.sys' beginnen" durch ein solches Meta-Wort beschreiben. Im folgenden werden wir für diese Meta-Worte den gängigeren Begriff *Muster* verwenden¹².

Die genaue Spezifikation der Suchmuster für eine spätere Implementierung und die Wahl von Verfahren zu ihrer Auswertung erfolgt in Kapitel 2.2 "Vertiefung: Spezifikation und Auswertung der Suchmuster".

2.1.2 Realisierung der Newsserver-Dienste

Der Zugriff auf Usenet-News wird durch Gateway-Agenten, die von verschiedenen *Locations* betrieben werden, realisiert.

Dabei soll auf Artikel, wie schon erwähnt, über das Filesystem eines Rechners zugegriffen werden. Auf jedem Rechner mit einer Ablaufumgebung für Mole innerhalb der Usenet-News als Dienst des Agentensystems angeboten werden soll, muß daher ein Newsserver-Programm installiert sein (z.B. Internet News; s. [INN96]) oder der Zugriff über ein verteiltes Dateisystem auf einen solchen Rechner ermöglicht werden. Letzteres stellt eine deutlich abgeschwächte Bedingung dar, die sich in der Praxis durch administrative Vorkehrungen einfach erfüllen läßt (z.B. "mount" des entsprechenden Dateisystems über das "Network File System").

Es wird davon ausgegangen, daß das Newsserver-Programm die Artikel in bestimmten Verzeichnissen ablegt und dabei insbesondere die hierarchische Struktur der Gruppennamen (z.B. "de.comp.sys.next.programmers") durch eine entsprechende Folge ineinander verschachtelter Verzeichnisse widerspiegelt (beispielsweise "/usr/spool/news/de/comp/sys/next/programmers"). Ferner wird angenommen, daß eine Datei mit Informationen über alle verfügbaren Gruppen auf dem Rechner vorhanden ist (s. [INN96]).

Weiterhin wird ein Gateway-Agent benötigt, der Informationen über an anderen *Locations* angebotenen Diensten zur Verfügung stellt (Stichwort "Trading", s.S. 66 [HOHL95]). Weil es sich hierbei um ein komplexes Thema handelt (insbesondere, wenn versucht wird, den Austausch solcher Informationen zwischen verschiedenen *Locations* zu automatisieren), das den Rahmen dieser Studienarbeit überschreiten würde, wird nur eine einfache Schnittstelle entworfen und die Implementierung mit fest voreingestellten Konfigurationen realisiert.

¹² In der Literatur wird diese Thematik unter dem Begriff *Patternmatching* (Musteranpassung) behandelt (s. z.B. S. 325ff in [SW92], wo die gängigsten Verfahren übersichtlich dargestellt werden).

2.1.3 Entwurf des Suchverfahrens

Für das Suchverfahren können generell zwei Verfahren in Betracht gezogen werden. So kann eine Suche entweder sequentiell oder parallel ausgeführt werden. Sequentiell ist dabei in dem Sinne zu verstehen, daß ein Agent nacheinander alle Locations besucht die Usenet-News anbieten und dort einen Suchvorgang durchführt. Analog dazu wird bei einem parallelen Verfahren an jede Location gleichzeitig jeweils ein Agent geschickt, der dann dort die Suche vornimmt. Zur Ermittlung der Locations, an denen Usenet-News angeboten werden, ist der zuvor erwähnte Gateway-Agent nötig.

Sequentielle Suche

Bei der Ausgestaltung eines sequentiellen Suchverfahrens gibt es keine großen Variationsmöglichkeiten. Es wird ein Agent erzeugt, der alle Locations nacheinander besucht. An jeder Location durchsucht er die lokal vorhandenen Gruppen und Artikel, speichert die Ergebnisse und besucht im Anschluß daran die nächste Location. Wichtig sind in diesem Fall Verfahren, die gewährleisten, daß keine Location mehrfach besucht wird und daß Gruppen, die an mehreren Locations verfügbar sind, nicht mehrfach durchsucht werden.

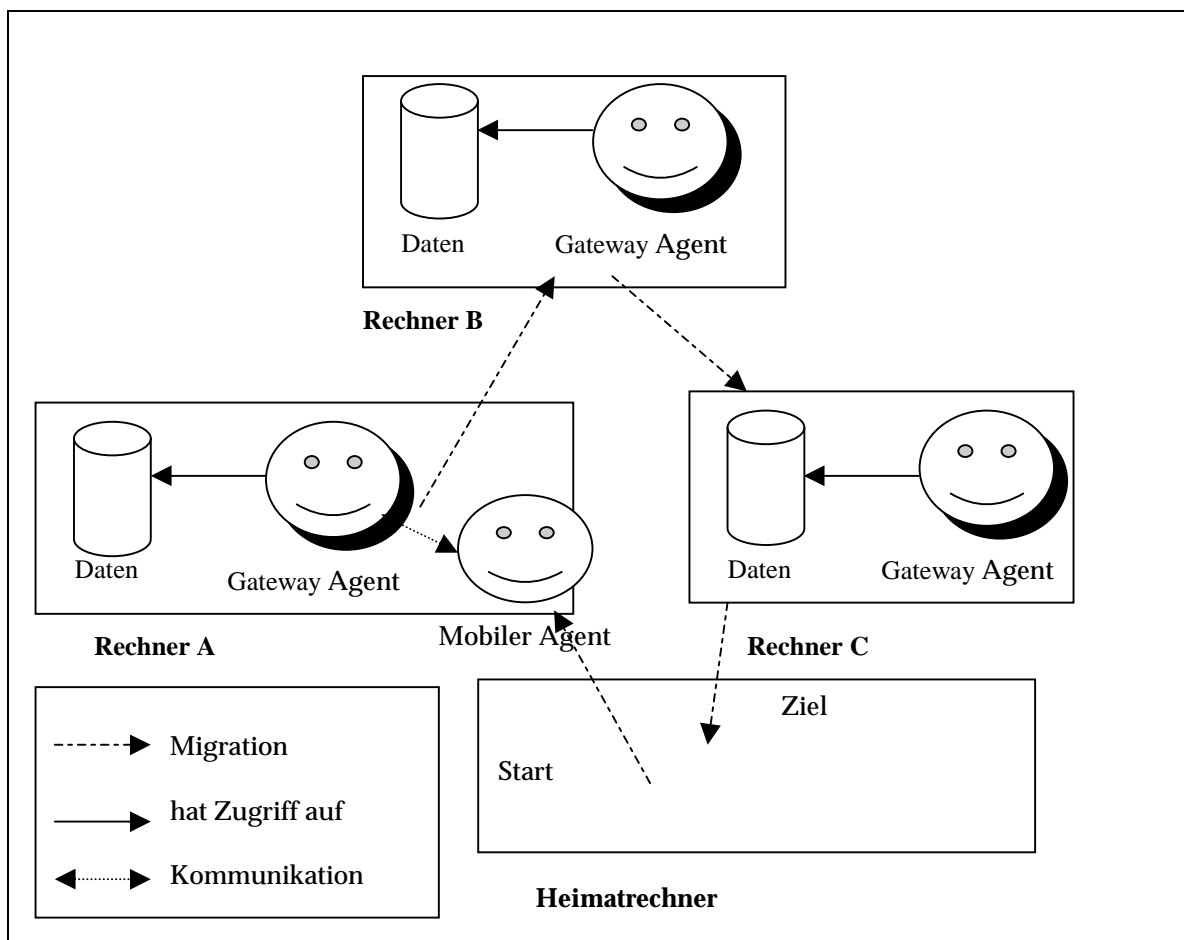


Abb. 4: Sequentielle Suche

Parallele Suche

Für ein paralleles Suchverfahren schlägt der Verfasser folgende Strategie vor:

An der Location, an welcher der Anwender das Suchformular bearbeitet, wird zu Beginn der Suche ein Koordinator eingesetzt. Dieser schickt Agenten zu alle bekannten Locations, die Usenet-News zur Verfügung stellen. Die Agenten beginnen, nach Gruppen zu suchen, die das Gruppenkriterium erfüllen und melden sie dem Koordinator. Dieser sammelt die Ergebnisse. Sobald ein Agent eine Gruppe gefunden hat, richtet er eine Anfrage an den Koordinator, ob er in dieser Gruppe nach Artikeln suchen darf. Der Koordinator erteilt nur dem ersten Agenten, der eine Anfrage für eine Suche in einer bestimmten Gruppe stellt, die Erlaubnis dort zu suchen. So wird vermieden, daß gleichzeitig von mehreren Agenten dieselbe Gruppe durchsucht wird. Bekommt ein Agent die Erlaubnis, in einer Gruppe zu suchen, so untersucht er alle Artikel und liefert diejenigen, die den Kriterien des Benutzers entsprechen an den Koordinator zurück. Der Koordinator präsentiert die Ergebnisse der Suche dem Anwender.

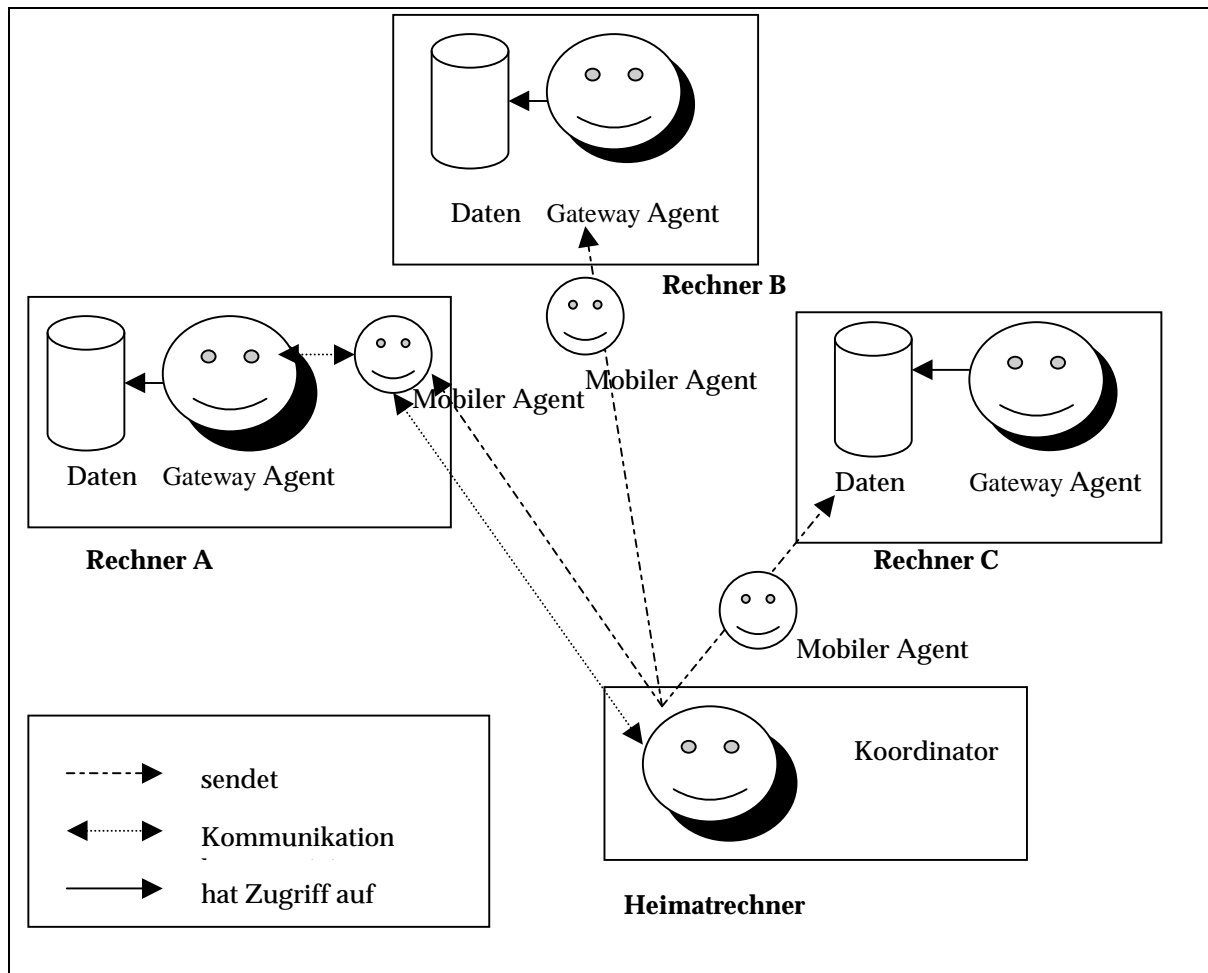


Abb. 5: Parallele Suche

Vergleich

Vorteil des parallelen Verfahrens ist die Möglichkeit, eine Suche durch Verteilung der Aufgaben auf mehrere Agenten zu beschleunigen. Die Granularität einer Verteilung wird dabei durch Anzahl der als "passend" erkannten Gruppen und die Anzahl verfügbarer Locations, an die jeweils ein Agent geschickt werden kann, begrenzt. Im günstigsten Fall (n als "passend" ermittelte Gruppen und n Locations, an denen gesucht werden kann) ist die parallele Suche daher wesentlich schneller, als die sequentielle Suche. Dabei wird jedoch vorausgesetzt, daß der Kommunikationsaufwand zur Koordination relativ gering ist. Dieser Koordinationsaufwand wird bei der sequentiellen Suche vermieden. Ein sequentielles Suchverfahren bietet zudem den Vorteil, daß der suchende Agent bis zur Beendigung des Suchvorgangs völlig unabhängig von seinem Ausgangsort arbeiten kann. Sollte aus irgendwelchen Gründen die Kommunikationsmöglichkeit mit dem Ausgangsort gestört sein, würde die parallele Suche beeinträchtigt, in ungünstigen Fällen sogar beendet werden. Die sequentielle Suche kann dagegen durch ein solches Problem nicht gestört werden.

Für die Implementierung der Suchmaschine soll sowohl das sequentielle als auch das parallele Verfahren realisiert werden.

Terminierung

Unabhängig vom gewählten Verfahren, ist die Frage zu klären, nach welchem Kriterium ein Suchvorgang terminiert. So sollte ein Suchvorgang selbstverständlich enden, wenn alle bekannten Locations, die Usenet-News zur Verfügung stellen, abgesucht wurden, die Suche demnach erschöpfend durchgeführt wurde.

Dieses Kriterium könnte in einer starken und einer schwachen Variante implementiert werden. Die schwache Variante zieht nur die am Ausgangsort der Suche durch einen Gateway-Agenten vermittelten Locations in Betracht. Die starke Variante würde diese Menge, bei Besuch einer Location durch die dem dortigen Gateway-Agenten bekannten Locations ergänzen. Dadurch würde sie alle Locations, die Usenet-News zur Verfügung stellen, in einem großen, zusammenhängenden System besuchen, die schwache nur die am Ausgangsort der Suche bekannten.

Zusätzlich schlägt der Verfasser eine zeitliche Begrenzung des Suchvorgangs kombiniert mit einer Begrenzung der Ergebnismenge vor. Die Begrenzung der Ergebnismenge stellt sicher, daß die Suche möglicherweise schon vor Ablauf der zeitlichen Begrenzung beendet werden kann. Dadurch können die Ressourcen des Gesamtsystems im Hinblick auf die verwendete Rechenzeit geschont werden. Insbesondere die Begrenzung der Ergebnismenge macht Sinn, da bei einer großen Menge von "Treffern" der Anwender in der Regel seine Suchanfrage nicht eng genug formuliert hat. Gewöhnlich wird er seine Suchanfrage beim Überschreiten einer bestimmten Anzahl von Treffern neu formulieren. In diesem Fall erweist es sich dann als unnötig, den ersten Suchvorgang erschöpfend durchzuführen.

Im Falle der sequentiellen Suche wertet der suchende Agent alle genannten Kriterien aus. Bei der parallelen Suchstrategie erfolgt die zeitliche Begrenzung des Suchvorgangs durch den suchenden Agenten. Die Terminierung durch Begrenzung der Ergebnismenge wird durch den Koordinator durchgeführt.

Weiterführende Gedanken

Bisher wurden weder Annahmen über die Qualität, der an einer Location zur Verfügung stehenden Dienste gemacht¹³, noch die Kosten in Betracht gezogen, die eine Suche verursachen kann. Die vorliegende Arbeit geht davon aus, daß alle Locations, falls sie Usenet-News zur Verfügung stellen, Artikel für den gleichen Zeitraum bereithalten. Weiterhin wird angenommen, daß dieser Dienst kostenlos angeboten wird. Eine zufriedenstellende Berücksichtigung der erwähnten Problematik würde eine Erweiterung der Systemarchitektur des Agentensystem erfordern (als Stichworte seien hier "Accounting" und "Banking" genannt¹⁴), die jedoch nicht im Rahmen dieser Studienarbeit geleistet werden kann.

¹³ Die Qualität eines Dienstes, der Usenet-News zur Verfügung stellt ließe sich unter anderem durch die Anzahl der verfügbaren Gruppen und den Zeitraum, in dem die Artikel verfügbar gehalten werden, definieren.

¹⁴ In [HOHL95] wird auf die genannten Stichworte ein wenig näher eingegangen (zu Accounting s.S. 54f und zu Banking S. 59f).

2.1.4 Entwurf der Benutzerschnittstelle

Eine sehr einfache Benutzerschnittstelle wird ebenfalls über einen Gateway-Agenten realisiert. Dem Benutzer präsentiert sich die Suchmaschine über ein Dialogfenster, das zwei Eingabefelder besitzt. In einem der Felder kann der Benutzer einen Text eingeben, der das Kriterium für die Auswahl eines Artikels über den Gruppennamen spezifiziert. Im zweiten Feld kann der Anwender das Kriterium zur Auswahl eines Artikels über die Subject-Zeile angeben. Darüber hinaus erhält der Anwender die Möglichkeit, das gewünschte Suchverfahren (Sequentiell oder Parallel) festzulegen und Kriterien für eine Terminierung anzugeben. Ein Suchvorgang wird dann durch einen Knopf im Dialogfenster ausgelöst. Dabei werden die eingegebenen Kriterien auf korrekten Aufbau geprüft und der Anwender im Fehlerfall durch eine einfache Rückmeldung informiert. Die Ergebnisse der Recherche werden als Artikelverweise in einer Liste angezeigt.

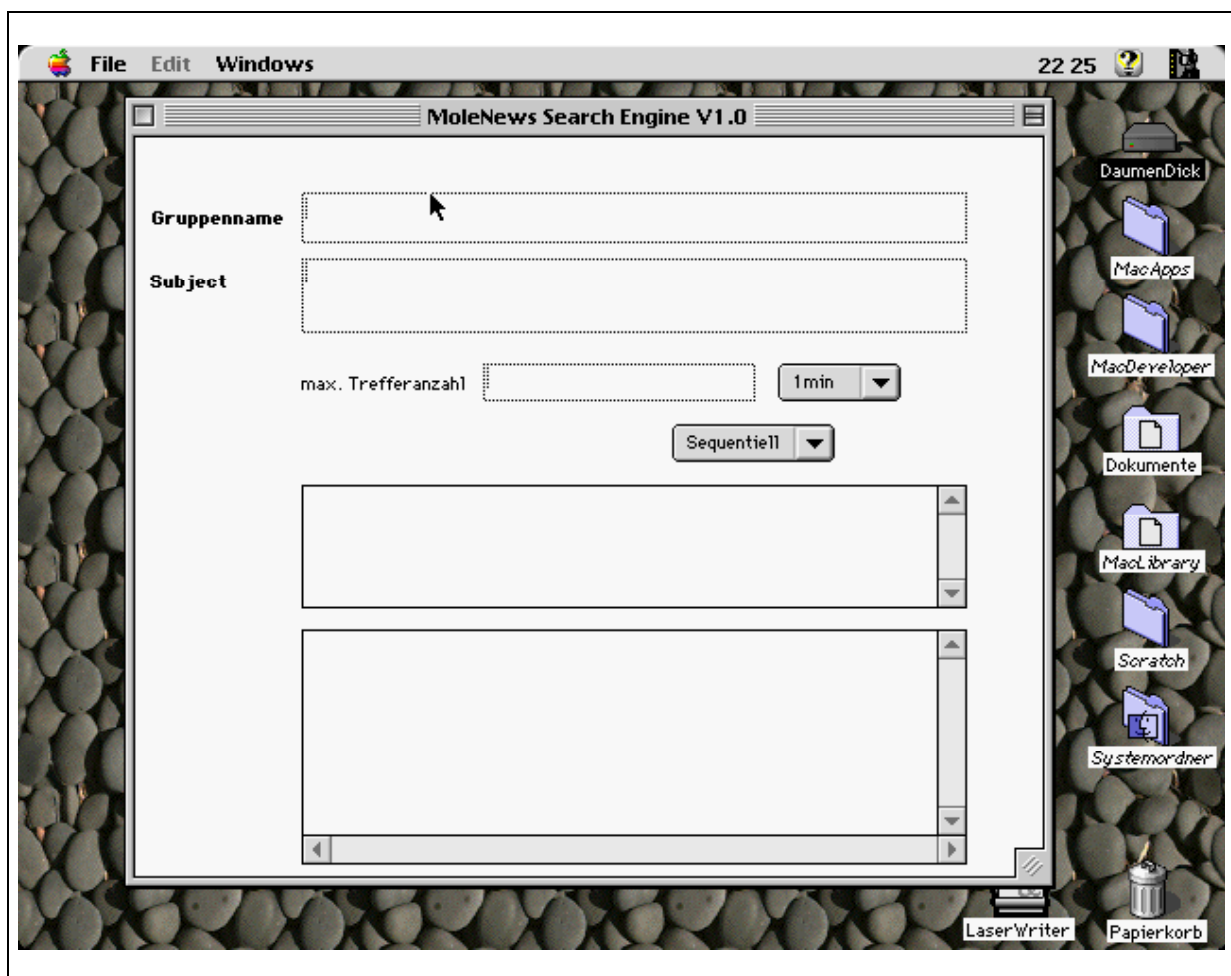


Abb. 6: Dialogfenster der Benutzerschnittstelle

2.2 Vertiefung: Spezifikation und Auswertung der Suchmuster

2.2.1 Anforderungen

Aus den bisherigen Überlegungen zur Spezifikation von Suchmustern haben sich einige Anforderungen ergeben. Diese sollen hier erneut kurz dargestellt werden, da sich aus ihnen eine Reihe von Teilaufgaben ergeben, für deren Lösung im weiteren Verlauf Verfahren zu finden und darzustellen sind.

Es wurde gewünscht, daß Gruppennamen über eine Menge alternativer Begriffe, die Subject-Zeilen der Artikel über einen logischen Ausdruck (ähnlich z.B. AltaVista) qualifiziert werden können. Außerdem sollen Muster eingesetzt werden, die eine Klasse von Worten bzw. Namen beschreiben können.

Die sich daraus ergebenden Teilaufgaben sind:

- Wahl einer geeigneten formale Spezifikation für die erwähnten Muster und eines Verfahrens zu deren Auswertung
- Formale Spezifikation der Suchausdrücke und Wahl eines Verfahrens zu ihrer Auswertung

2.2.2 Patternmatching

Das Konzept der Muster wurde bisher schon mehrfach erwähnt. Diese sollen eine Klasse von Worten oder Namen, allgemeiner eine Menge von Zeichenfolgen beschreiben. Wird eine Zeichenfolge aus dieser Menge in einem Text erkannt, so gilt das Muster als in diesem Text enthalten.

Im folgenden sollen zwei gängige Möglichkeiten vorgestellt werden, die es erlauben, solche Muster zu spezifizieren. Die eine Methode verwendet sog. Jokerzeichen, die andere reguläre Ausdrücke.

Anschließend werden Verfahren betrachtet, mit denen das Auftreten solcher Muster in einem Text erkannt werden kann. Diese Thematik wird in der Informatik allgemein unter dem Begriff Patternmatching (Musteranpassung) dargestellt (s.S. 326 [SW91]). Es handelt sich dabei um ein Gebiet, daß relativ gut erforscht ist und das teilweise sehr stark von der theoretischen Forschung profitiert hat¹⁵.

Jokerzeichen

Als Jokerzeichen werden in der Praxis¹⁶ gewöhnlich das Fragezeichen ("?") und der Stern ("*") verwendet. Der Stern steht für eine Folge beliebiger Zeichen mit beliebiger Länge (die Länge Null ist ebenfalls erlaubt und steht für kein Zeichen), das Fragezeichen für genau ein beliebiges Zeichen. So beschreibt "Agent*" die Menge aller Worte, die mit "Agent" beginnen und mit einer beliebigen Folge

¹⁵ So sind die regulären Ausdrücke Gegenstand umfangreicher theoretischer Forschungen auf den Gebieten der Automatentheorie und der formalen Sprachen (s. dazu z.B. die sehr ausführliche Darstellung in [HU94]). Einige der in der Theorie vorgestellten Beweise verwenden Konstruktionsverfahren (Algorithmen), die als Grundlage einer praktischen Implementierung dienen können. Beispielsweise sei das Verfahren genannt, das zum Beweis der Äquivalenz zwischen Deterministischen Endlichen Automaten und Nichtdeterministischen Endlichen Automaten vorgestellt wurde (s.S. 22ff [HU94]). Diese Konstruktion des sog. Potenzautomaten ist Grundlage eines Algorithmus zur effizienten Erkennung regulärer Ausdrücke (s.S. 117ff [ASU86]).

¹⁶ Jokerzeichen werden z.B. bei fast allen gängigen Betriebssystemen in der Shell zur Angabe von Dateinamen verwendet. Unix, Windows NT, selbst MS-DOS bieten diese Möglichkeit.

von Zeichen enden. "?ber" beschreibt Zeichenfolgen, die aus einem beliebigen Zeichen, gefolgt von "ber" bestehen (z.B. "Aber", "Ober", "Über", aber auch "Xber", "xber", etc.).

Reguläre Ausdrücke

Weitaus größere Möglichkeiten als die Verwendung von Jokerzeichen bietet der Einsatz regulärer Ausdrücke. Ihre Mächtigkeit gewinnen reguläre Ausdrücke dadurch, daß sich diese aufbauend auf einfachen Ausdrücken durch eine Reihe von Operationen zu komplexeren Ausdrücken zusammensetzen lassen. Dabei lassen sich, wie im Anschluß gezeigt wird, durch reguläre Ausdrücke auch alle Zeichenmengen beschreiben, die durch Einsatz von Jokerzeichen definiert wurden.

Der einfachste reguläre Ausdruck beschreibt ein Zeichen (z.B. "a"). Beliebige reguläre Ausdrücke können durch die Operationen "Verkettung", "Oder" und "Hüllenbildung" miteinander verknüpft werden, die nachfolgend näher beschrieben werden. Da ein regulärer Ausdruck immer eine Menge von Zeichenfolgen beschreibt (im einfachsten Falle eine Menge, die eine Zeichenfolge aus einem Zeichen enthält), lassen sich diese Operationen auf Operationen über Mengen zurückführen (Produkt, Vereinigung, Stern).

1. "Verkettung". Sind A und B reguläre Ausdrücke, so ist auch AB ein regulärer Ausdruck. AB beschreibt die Menge aller Zeichenfolgen, die mit einer von A beschriebenen Zeichenfolge beginnen, an die sich direkt eine von B beschriebene Zeichenfolge anschließt. So lassen sich beispielsweise überhaupt erst längere Zeichenfolgen beschreiben, die durch Verkettung des einfachsten regulären Ausdrucks, des Zeichens (der Zeichenfolge mit Länge eins), gebildet werden ("Agent", stellt so die Verkettung der Zeichen "A", "g", "e", "n" und "t" dar).
2. "Oder". Sind A und B reguläre Ausdrücke, so ist auch $(A|B)$ ein regulärer Ausdruck. $(A|B)$ beschreibt die Menge aller Zeichenfolgen, die alle Zeichenfolgen aus A und B enthält (Vereinigung). Eine Zeichenfolge gilt als durch den Ausdruck $(A|B)$ beschrieben, wenn es sich um eine Zeichenfolge aus A oder aus B handelt (so wird z.B. durch "(Maier|Meier)" die Zeichenfolge "Maier" aber auch "Meier" erkannt).
3. "Hüllenbildung". Wenn A ein regulärer Ausdruck ist, dann ist auch $(A)^*$ ein regulärer Ausdruck. $(A)^*$ beschreibt die Menge aller Zeichenfolgen, die sich durch n -faches Hintereinanderschreiben der durch A beschriebenen Zeichenfolgen ergeben. Zu beachten ist, daß auch eine Zeichenfolge der Länge Null durch $(A)^*$ beschrieben wird ("1(0)*" beschreibt die Menge aller Zeichenfolgen, die mit "1" beginnen, gefolgt von einer beliebigen Anzahl von "0"-Zeichen; aufgrund des zuvor gesagten ist aber auch die Zeichenfolge "1" in dieser Menge enthalten).

Die Umwandlung eines Musters mit Jokerzeichen in einen regulären Ausdruck läßt sich recht einfach vornehmen.

Das Fragezeichen wird durch einen Ausdruck ersetzt, der alle Zeichen der Zeichenmenge, aus der die zu untersuchenden Zeichenfolgen gebildet werden, durch die "Oder"-Operation verknüpft. Nennen wir diesen Ausdruck den Alle-Zeichen-Ausdruck. Dieser würde in etwa folgende Gestalt besitzen "...(((A | B) | C) | D) ...".

Das Jokerzeichen Stern könnte dann durch einen Ausdruck ersetzt werden, der sich durch Anwendung der Hüllenbildung auf den Alle-Zeichen-Ausdruck ergibt.

Damit ist klar, daß reguläre Ausdrücke das mächtigere Werkzeug sind, da sie alle Möglichkeiten der Verwendung von Jokerzeichen abdecken, zusätzlich aber weitere Möglichkeiten bieten (Alternativen durch den "Oder"-Operator, Hintereinanderschreiben durch die "Hüllenbildung").

Verfahren zum Patternmatching

Das Erkennen von Mustern in Texten ist eine Aufgabenstellung, zu der inzwischen eine Reihe erprobter und gut erforschter Algorithmen entdeckt wurden. Einige dieser Verfahren befassen sich nur mit einem Sonderfall der Mustererkennung, der Erkennung von einfachen Zeichenfolgen. Muster beschreiben dagegen, wie schon erwähnt, Mengen von Zeichenfolgen. Ein Muster wird also erkannt, wenn eine Zeichenfolge aus der von dem Muster beschriebenen Menge erkannt werden konnte. Trotzdem sollen diese Verfahren im Anschluß kurz betrachtet werden, da sie möglicherweise einfach erweitert werden könnten, um zumindest Muster mit Jokerzeichen erkennen zu können.

Ausgangspunkt der Betrachtungen¹⁷ ist ein Text der Länge n und eine Zeichenfolge der Länge m , die in dem Text gesucht werden soll.

Ein naiver Ansatz würde vom ersten Zeichen des Textes ausgehend Abschnitte der Länge m mit der zu suchenden Zeichenfolge vergleichen. Bei einer Nichtübereinstimmung würde die Position, ab der verglichen wird um ein Zeichen weiter nach rechts verschoben. Dieses Verfahren hat im ungünstigsten Fall eine Laufzeit von $O(m \cdot n)$, in normalerweise auftretenden Texten ist aber nur mit einer durchschnittlichen Laufzeit von $O(m+n)$ zu rechnen.

Die obere Schranke des ungünstigsten Falles läßt sich auf $O(m+n)$ begrenzen, wenn das Verfahren von Knuth, Morris und Pratt (kurz KMP) eingesetzt wird. Diese entwickelten einen Algorithmus, der beim Auftreten einer Nichtübereinstimmung zwischen Muster und Text die Informationen über die bisher verglichenen Zeichen ausnutzt. Dabei besitzt das KMP-Verfahren den Vorteil, daß die Position im Text, ab der verglichen wird, immer nur inkrementiert wird (besonders günstig ist dies für die Suche in sequentiellen Dateien, da ein Rücksetzen hier nur durch komplizierte Mechanismen möglich wäre).

Ein weiteres Verfahren wurde von Boyer und Moore (kurz BM) entwickelt. Dieses nutzt ebenfalls die Informationen über die Zeichen aus, die bis zum Auftreten einer Nichtübereinstimmung untersucht wurden. Allerdings wird in diesem Fall das Muster von rechts nach links mit dem Text verglichen. Durch die Verwendung der Informationen über die bisher verglichenen Zeichen des Musters und des Zeichens im Text, das die Nichtübereinstimmung verursachte, gelang es Boyer und Moore einen Algorithmus zu entwerfen, der im ungünstigsten Fall eine Laufzeit von $O(n+m)$, im Durchschnitt aber nur eine Laufzeit von $O(n/m)$ benötigt.

Das Suchen nach Mustern, die Jokerzeichen enthalten, ließe sich mit jedem der drei vorgestellten Verfahren implementieren.

Am einfachsten könnte diese Aufgabe für den naiven Ansatz gelöst werden. Bei diesem wäre es möglich, die Berücksichtigung der Muster in den Algorithmus einzubetten. Wird ein Fragezeichen im Muster gefunden, so findet kein Vergleich mit dem Text statt, sondern es wird nur die Vergleichsposition im Text um ein Zeichen nach rechts verschoben. Bei einem Stern im Muster wird

¹⁷ Eine genauere Herleitung der Verfahren und Überlegungen zum Laufzeitverhalten findet sich in [SW92], S. 325ff. Sämtliche Informationen zum Laufzeitverhalten der Verfahren wurden dort entnommen.

Der Vollständigkeit halber sei auch die Originalliteratur genannt:

- | | |
|---------|--|
| [BM77] | Boyer, R.S; Moore, J.S.: <i>A fast string searching algorithm</i>
Communications of the ACM, 20, S. 762ff |
| [KMP77] | Knuth, D.E; Morris, J.H.; Pratt, V.R.: <i>Fast pattern matching in strings</i>
SIAM Journal on Computing, 6, S. 323ff |

die Position im Text solange verschoben, bis das dem Stern im Muster folgenden Zeichen gefunden wurde¹⁸ bzw. der Text zu Ende ist.

Bei den Verfahren von KMP oder BM erscheint es dem Verfasser nicht möglich, die Berücksichtigung der Jokerzeichen in den Algorithmus zu integrieren. Der Verfasser würde daher einen anderen Ansatz vorschlagen, der kurz skizziert werden soll.

Durch die Jokerzeichen wird ein Muster in mehrere einfache Zeichenfolgen zerlegt. Diese Zeichenfolgen können mit den Verfahren von KMP oder BM wie bisher gefunden werden. Folgt einer Zeichenfolge im Muster ein Fragezeichen, so wird zuerst diese Zeichenfolge gesucht, anschließend wird die Position im Text, ab der verglichen wird auf ein Zeichen hinter dem Ende der gefundenen Zeichenfolge gesetzt (es wird also ein Zeichen übersprungen). Dann werden die möglicherweise vorhandenen weiteren Zeichenfolgen des Musters analog behandelt. Wird eine Zeichenfolge im Muster von einem Stern gefolgt, so wird nach dieser Zeichenfolge gesucht und daraufhin nach der dem Stern folgenden Zeichenfolge des Musters (falls das Muster mit dem Stern endet, ist die Suche erfolgreich beendet). Der folgende Suchvorgang darf natürlich erst direkt hinter dem Ende der gefundenen Zeichenfolge im Text beginnen.

Schlußfolgerungen

Alle drei Verfahren besitzen jedoch einen entscheidenden Nachteil, wenn man sie auf die Eignung zur Auswertung von Suchausdrücken betrachtet. In diesen Suchausdrücken soll in der Regel das Auftreten *mehrerer* Muster in einem Text in eine logische Beziehung gesetzt werden ("Suche UND Usenet-News UND (Agent ODER Suchmaschine)"). Die zuvor betrachteten Verfahren sind jedoch nur in der Lage, jeweils nach *einem* Muster in einem Text zu suchen. So muß der Text für jedes in einem Suchausdruck erscheinende Muster einzeln untersucht werden.

Die Auswertung regulärer Ausdrücke läßt sich mit einer Laufzeit von $O(n)$ (s.S. 128[ASU86]) bewerkstelligen (dies geschieht durch Umwandlung eines regulären Ausdrucks in einen deterministischen Automaten). Außerdem kann durch reguläre Ausdrücke ein Text nach mehreren Mustern in einem Suchvorgang untersucht werden (dazu wird einfach ein neuer Ausdruck gebildet, der die Alternative der verschiedenen Muster darstellt). Daher hat sich der Verfasser dazu entschieden, die Muster als reguläre Ausdrücke zu realisieren. Damit wird nicht nur das mächtigere Werkzeug eingesetzt, sondern seine Anwendung kann in der Regel auch effizienter gestaltet werden. Der erhöhte Aufwand einer Implementierung muß dabei in Kauf genommen werden.

¹⁸ Natürlich müssen dabei auch alle Sonderfälle berücksichtigt werden. So kann ein Muster mit einem Stern enden. Dann kann die Suche sofort beendet werden, wenn alle Zeichen vor dem Stern mit dem gerade untersuchten Textausschnitt übereinstimmen.

Darüber hinaus sollte der Algorithmus Fälle, wie das Auftreten mehrerer Jokerzeichen hintereinander, robust verarbeiten können. Alternativ könnte die Eingabe solcher Muster auch dem Anwender "verboten" werden.

Spezifikation regulärer Ausdrücke

Die folgende Tabelle stellt eine Grammatik dar, die den exakten Aufbau regulärer Ausdrücke definiert, wie sie in dieser Studienarbeit für eine Implementierung eingesetzt werden sollen. Dabei werden gegenüber dem zuvor vorgestellten Grundkonzept einige Erweiterungen vorgenommen, die im wesentlichen der Erleichterung beim Formulieren eines Ausdrucks dienen. Diese werden im Anschluß an die folgende Tabelle erläutert.

Grammatik: Regulärer Ausdruck	
Expression	= Serie Expression " " Serie
Serie	= Singleton Serie Singleton
Singleton	= "." "[" FullCCL "]" "(" Expression ")" "\"" String "\"" EscapedChar Singleton "*" Singleton "+" Singleton "?"
FullCCL	= ["^"] CCL
CCL	= RangeOrChar CCL RangeOrChar
RangeOrChar	= EscapedChar EscapedChar "-" EscapedChar
String	= *EscapedChar
EscapedChar	= <jedes Zeichen, das keine besondere Bedeutung für den Aufbau regulärer Ausdrücke besitzt> "/" <jedes Zeichen>

Analog des Fragezeichens beim Einsatz von Jokerzeichen in Mustern wird hier der Punkt "." verwendet. Er dient als Abkürzung für den erwähnten Alle-Zeichen-Ausdruck.

Außer dem Stern, als Operator der Hüllenbildung, gibt es noch zwei weitere Operatoren. Das Zeichen "?" bedeutet, daß der vorhergehende reguläre Ausdruck einmal oder keinmal vorkommen darf. Das Zeichen "+" dagegen, daß der vorhergehende Ausdruck n-mal vorkommen darf, mindestens aber einmal vorkommen muß.

Statt eines einfachen Zeichens darf in einem regulären Ausdruck auch eine Zeichenklasse verwendet werden. Eine Zeichenklasse ist eine Teilmenge der für Zeichenfolgen verwendeten Zeichenmenge. Die Regel FullCCL definiert den Aufbau einer solchen Zeichenklasse. Diese wird immer durch eckige Klammern begrenzt. Innerhalb der Klammer kann dann eine Folge von Zeichen angegeben werden, die zu dieser Zeichenklasse gehören sollen. Als Abkürzung kann auch ein Bereich von Zeichen angegeben werden. Ein Bereich wird durch zwei Zeichen definiert, zwischen denen sich ein "-" Zeichen befindet (z.B. beschreibt "[0-9]" alle Ziffern). Ein der einleitenden Klammer folgendes "^"-

Zeichen bedeutet, daß die Zeichenklasse, das Komplement der nachfolgend angegebenen Zeichen darstellen soll ("^[0-9] steht also für alle Zeichen, außer den Ziffern).

Die Regel `EscapedChar` wurde eingeführt, damit auch Zeichen, deren Bedeutung schon für den Aufbau von regulären Ausdrücken besetzt wurde (z.B. die eckigen Klammern) als zu erkennende Zeichen definiert werden können ("Escape-Mechanismus"). Diese Zeichen müssen von einem "\"-Zeichen angeführt werden.

2.2.3 Suchausdrücke für Gruppennamen

Wir befassen uns zuerst mit der Definition von Ausdrücken, die Gruppennamen qualifizieren sollen. Ein solcher Ausdruck soll eine Menge von Zeichenfolgen angeben. Ein Gruppenname gilt als gefunden, wenn er exakt einer der angegebenen Zeichenfolgen entspricht.

Grammatik: Qualifizierender Ausdruck für Gruppennamen

Expression	=	SimpleExpr SimpleExpr Expression
SimpleExpr	=	String QuotedString DollarQuotedPattern

Ein Ausdruck, der Gruppennamen qualifizieren soll, besteht aus einer Folge von `Strings`, `QuotedStrings` oder `DollarQuotedPatterns`. Diese drei Terminale stehen für drei unterschiedlich definierte Zeichenfolgen. Damit wird die Idee verfolgt, innerhalb eines Ausdruck sowohl einfache Zeichenfolgen, die exakt erkannt werden müssen, als auch spezielle Muster angeben zu können.

Als Unterscheidungsmerkmal zwischen normalen Zeichenfolgen, den `Strings` und speziellen Mustern, den `DollarQuotedPatterns` wird festgelegt, daß diese Muster durch eine Zeichenfolge definiert werden müssen, die mit einem Dollarzeichen beginnt und endet. Dies bedeutet auch, daß keine Dollarzeichen innerhalb einer Zeichenfolge, die einen `String` definiert bzw. in einer von Dollarzeichen umrahmten Zeichenfolge, die ein `DollarQuotedPattern` definiert, verwendet werden dürfen.

Um diese Einschränkung zu umgehen wurde der `QuotedString` eingeführt. Ein `QuotedString` ist analog zum `DollarQuotedPattern` eine Zeichenfolge, die von doppelten Anführungszeichen statt Dollarzeichen umrahmt wird. Zwischen den Anführungszeichen dürfen jetzt bei einem `QuotedString` die Dollarzeichen verwendet werden. Analog dürfen bei einem `DollarQuotedPattern` zwischen den Dollarzeichen die doppelten Anführungszeichen verwendet werden.

Damit auch Dollarzeichen in einem `DollarQuotedPattern` bzw. Anführungszeichen in einem `QuotedString` verwendet werden dürfen, wird definiert, daß eine Folge aus dem "\"-Zeichen gefolgt von einem beliebigen zweiten Zeichen (insbesondere das Anführungs- bzw. Dollarzeichen) so interpretiert wird, als wäre nur dieses zweite Zeichen aufgetreten und dieses zweite Zeichen wird nicht als Endebegrenzung der Zeichenfolge erkannt ("Escaping-Mechanismus").

Die folgende Tabelle stellt beispielhaft jeweils eine wohlgeformte Zeichenfolge dar, gefolgt vom Namen des Terminals, dem sie zugeordnet wurde und der Zeichenfolge, als die sie intern weiterverarbeitet wird.

Beispiele		
Zeichenfolge	Terminal	Intern
EinTest	String	EinTest
"String"	QuotedString	String
"String mit escape\""	QuotedString	String mit escape"
\$Muster\$	DollarQuotedPattern	Muster
"String mit \$"	QuotedString	String mit \$

2.2.4 Suchausdrücke für Artikel

Zur Qualifizierung von Artikeln sollen Ausdrücke eingesetzt werden, mit denen sich logische Aussagen über das Enthaltensein von Zeichenfolgen in der Subject-Zeile eines Artikels machen lassen. Im folgenden sollen einige Beispiele zeigen, wie solche Ausdrücke aussehen könnten (die Operatoren wurden der Übersichtlichkeit halber fett dargestellt):

News **and** (Agent **or** (Intelligente **near** Agenten) **or** Suchmaschine)

Thomas **near** Wieger **and** Studienarbeit

angeln **and** (Würmer **or** Fliegen)

Natürlich sollen auch, statt der in den Beispielen verwendeten konkreten Worten, Muster eingesetzt werden können.

Grammatik: Qualifizierender Ausdruck für Artikel		
Expression	=	OrExp
OrExp	=	AndExpr AndExpr "or" AndExpr
AndExpr	=	NotExpr NotExpr "and" NotExpr
NotExpr	=	NearExpr "not" NearExpr
NearExpr	=	SimpleExpr SimpleExpr "near" SimpleExpr "(" Expression ")"
SimpleExpr	=	String QuotedString DollarQuotedPattern

Die Auswertung der Ausdrücke kann induktiv definiert werden. Eine `SimpleExpr` wird als wahr ausgewertet, wenn die durch diese definierte Zeichenfolge in der Subject-Zeile eines Artikels auftritt. Die Operatoren "and", "or" und "not" werden wie in der Aussagenlogik üblich ausgewertet. Eine Besonderheit stellt der "near"-Operator dar. Dieser wird als wahr ausgewertet, wenn die beiden `SimpleExpr`, die er verbindet, innerhalb der Subject-Zeile eines Artikels auftreten und zusätzlich nur eine bestimmte Anzahl von Zeichen voneinander entfernt sind. Ist diese Distanz überschritten, wird der Ausdruck als falsch ausgewertet, obwohl die Zeichenfolgen möglicherweise gefunden wurden. Typischerweise wird der "near"-Operator eingesetzt, wenn zusammengesetzte Begriffe gesucht werden, bei denen nicht klar ist in welcher Reihenfolge ihre Bestandteile innerhalb eines Textes auftreten. Der "near"-Operator toleriert auch eine begrenzte Anzahl beliebiger Zeichen, die zwischen den beiden Begriffen auftreten können. So umgeht er die Probleme, die andernfalls durch Trennzeichen oder eine unbestimmbare Anzahl von Leerzeichen zwischen den Teilbegriffen, auftreten könnten.

3 Implementierung

3.1 Vorbetrachtungen

Struktur des Systems

Die konzeptionelle Struktur der im Rahmen dieser Studienarbeit erstellten Suchmaschine ergibt sich aus den im Entwurf genannten Teilaufgaben. Diese waren:

- Spezifikation und Auswertung der Suchmuster
- Realisierung der Newserver-Dienste
- Entwurf des Suchverfahrens
- Entwurf der Benutzerschnittstelle

Die Abbildung dieser konzeptionellen Struktur auf die Implementierung eines Systems hängt von den zur Verfügung gestellten Konstrukten der verwendeten Programmiersprache ab. Aufgrund gründlicher Vergleiche wählte der Entwickler des Mobile-Agenten-Systems Mole die objekt-orientierte Sprache Java für dessen prototypische Implementierung (s.S. 26ff [HOHL95])¹⁹. Grundlegendes Strukturierungsmerkmal dieser Sprache sind Klassen und sog. Packages. Durch Packages²⁰ wird in Java ein Modulkonzept realisiert (s.S. 113ff [GJS96]). Packages stellen eine übergeordnete Struktur dar, zu der Klassen zusammengefaßt werden können.

Zur Lösung der zu Beginn genannten Teilaufgaben wurde, aufbauend auf dem objekt-orientierten Ansatz der Sprache Java, eine Reihe interagierender Klassen geschaffen. Die entworfenen Klassen wurden dabei entsprechend ihrer Funktionalität in entsprechende Packages zusammengefaßt. Dadurch ließ sich die konzeptionelle Struktur des Systems sehr gut in der Implementierung abbilden.

Suchmuster wurden durch zwei Packages realisiert. Das Package `mole.TEW.Lexing` stellt Klassen zur Definition und Auswertung regulärer Ausdrücke bereit. Darauf aufbauend kann durch die in `mole.TEW.StringMatching` zusammengefaßten Klassen die Definition und Auswertung der Suchausdrücke vorgenommen werden.

In `mole.TEW.NewsAdaptor` wurden, als Teilaspekt bei der Realisierung der Newserver-Dienste, Gruppen und Artikel durch Klassen abgebildet. Weiterhin enthält dieses Package die Definition einer Schnittstelle zum Zugriff auf Gruppen und Artikel und eine Klasse, die diese Schnittstelle implementiert.

Grundlegende Funktionalität (z.B. Ausgabe von Debug-Informationen, Containerklassen, Sortierverfahren), die während der Implementierung des Systems benötigt wurde, ist durch Klassen in einem eigenen Package namens `mole.TEW.Foundation` zusammengestellt worden.

Alle weiteren Klassen wurden in das schon existierende Package `mole` integriert, in dem alle Klassen enthalten sind, die den Prototypen des Agentensystems Mole ausmachen. Insbesondere der Gateway-Agent zur Bereitstellung von Usenet-News, ein Systemagent der die Benutzerschnittstelle realisiert

¹⁹ Es sollte allerdings erwähnt werden, daß die vorgenommenen Vergleiche nicht unter dem hier betrachteten Aspekt der Strukturierungsmöglichkeiten durchgeführt wurden, sondern andere Gesichtspunkte wie Systemunabhängigkeit oder Sicherheit im Mittelpunkt standen.

²⁰ In einem Package werden dabei Klassen und sog. Interfaces zusammengefaßt. Interfaces stellen eine rein deklarative Komponente dar, mit der Konstanten und abstrakte Methoden definiert werden können. Sie beschreiben damit nur eine Schnittstelle, deren Implementierung jedoch durch eine Klasse erfolgen muß. Im Falle der Klassen wird bei der Codierung nicht - wie in anderen objekt-orientierten Programmiersprachen (z.B. C++ oder Objective C) - zwischen Deklaration und Implementierung getrennt (diese Trennung erfolgt sonst typischerweise durch sog. "Header"-Dateien).

und Agenten, welche die Suchverfahren realisieren wurden vom Verfasser dieser Studienarbeit dem Package `mole` hinzugefügt.

Gliederung

Im weiteren Verlauf sollen die implementierten Packages, teilweise aber auch einzelne Klassen vorgestellt werden, die in ihrer Gesamtheit eine Suchmaschine realisieren.

Die Darstellung orientiert sich dabei an der konzeptionellen Struktur des Systems, indem versucht wird der Reihenfolge der zu lösenden Teilaufgaben zu folgen, als auch an den Notwendigkeiten, die sich während der Phase der Implementierung ergaben. So werden zum besseren Verständnis Packages und Klassen, auf denen andere Teile des Systems aufbauen, zuerst vorgestellt.

Im weiteren Verlauf werden folgende Themen betrachtet:

- Grundlegende Funktionalität
- Reguläre Ausdrücke
- Suchausdrücke
- News-Anbindung
- Vereinfachter Aufruf von RPCs
- Der Gateway-Agent für Newsserver-Dienste
- Ein einfacher Trading-Systemagent
- Sequentielle und Parallele Suche
- Die Benutzerschnittstelle

3.2 Grundlegende Funktionalität

Während der Implementierung stieß der Verfasser auf Probleme bzw. benötigte er Hilfsfunktionalität, die zur Entwicklung von Klassen führten, die im Package `mole.TEW.Foundation` zusammengefaßt wurden.

Einführung der Klasse `XtBitSet`

So stellte sich heraus, daß die Implementierung der Klasse `BitSet` aus dem Package `java.util`²¹ von Sun für die vorhergesehene Aufgabe nicht geeignet war. Die Klasse `BitSet` stellt eine Menge von Bits dar. Jedes Bit kann durch eine Zahl vom Typ `int` indiziert werden. Es gibt Methoden um Bits zu setzen, zu löschen, abzufragen und zwei `BitSets` über eine logische Operation ("und", "oder", "exklusiv-oder") miteinander zu verknüpfen.

Ein neu erzeugtes `BitSet` stellt dabei Platz bereit, um den Zustand von Bits bis zu einem gewissen Maximalindex zu speichern. Soll ein Bit verändert werden, dessen Index oberhalb dieses Maximalindex liegt, so wird durch die Implementierung von Sun Platz geschaffen, um Bits bis zu diesem neuen Index behandeln zu können²².

Leider wird dies bei der logischen Verknüpfung von `BitSets` nicht berücksichtigt. Wird ein `BitSet` mit Platz für 100 Bits mit einem `BitSet` mit Platz für 1000 Bits verknüpft (`bitSet100.or(bitSet1000)`), so wird das kleine `BitSet` nicht auf die Größe des größeren erweitert. Dies hat zur Folge, daß die Verknüpfung von `BitSets` das Ergebnis in vielen Fällen auf zu-

²¹ Dieses Package gehört zu den von Sun im JDK 1.0.2 mitgelieferten Basis-Packages (s. [JDK97]).

²² Die Implementierung von Sun stellt den Platz für die Bits in einem `BitSet` in Vielfachen von 64 bereit. Daher wird in der Regel auch Platz für eine Reihe von Bits mit einem Index größer als dem gewünschten geschaffen.

wenig Bits beschränkt. Problematisch erwies sich weiterhin, daß BitSets keine Methode zur einfachen Invertierung bereitstellten²³.

Eine Modifikation der Klasse BitSet durch Erben, Überladen und Modifizieren schied aus, da die Klasse als final (s.S. 133 [GJS96]) definiert wurde. So war der Verfasser gezwungen eine eigene Klasse namens XtBitSet zu entwickeln, welche die vorhandene Funktionalität von BitSet zur Verfügung stellt und zudem die erwähnten Probleme beseitigt.

Sortierfunktionalität für Vector

Die Klasse Vector des Packages java.util stellt eine sog. Containerklasse dar. Eine Instanz dieser Klasse kann andere Objekte verwalten. Man sagt auch, ein Vector enthält diese Objekte ("contains"). Daher rührt der Name Containerklasse. Der Zugriff auf die in einem Vector enthaltenen Objekte erfolgt wie beim BitSet über einen Index, eine Zahl vom Typ int.

Im Rahmen dieser Studienarbeit erwies es sich als notwendig, die in einem Vector enthaltenen Objekte sortieren zu können. Dazu wurde die Klasse XtVector eingeführt. Diese erbt von Vector (der in diesem Falle nicht als final deklariert wurde) und implementiert eine Methode namens sort(). Objekte in einem Vector können jedoch nur sortiert werden, wenn sie verglichen werden können. Dazu müssen sie das folgende Interface implementieren:

```
public interface Comparable
{
    /**   Vergleicht die beiden Objekte.

        @return      Wert == 0      => this == that
                   Wert < 0 => this < that
                   Wert > 0 => this > that
    */
    int compareTo( Object that);
}
```

Die Funktion sort() basiert intern auf dem Quicksort-Verfahren (s.S 145ff [SW92]), wobei eine teilrekursive Variante verwendet wird. Dabei wird jeweils der kleinere Teil des während des Sortiervorgangs gebildeten Intervalls rekursiv, der größere iterativ weitersortiert, um Rekursionsprobleme (Überlauf des Stapels) bei Vektoren mit sehr vielen Elementen zu vermeiden.

Debug-Funktionen

Zur komfortablen Ausgabe von Status- und Fehlermeldungen wurde die Klasse Logging implementiert. Sie stellt eine einfache Infrastruktur bereit, um Ausgaben in bestimmte Klassen zusammenzufassen. Eine gemeinsame Klasse von Ausgaben wird dabei intern durch eine Zahl zusammengefaßt. Die Ausgabe läßt sich dann mit speziellen Methoden für eine Klasse aktivieren bzw. deaktivieren.

Über die Methode logLineForNum(String s, int num) kann ein String ausgegeben werden. Die Ausgabe erfolgt jedoch nur, wenn sie für die angegeben Zahl num "freigeschaltet" wurde. Dies geschieht durch die Methode turnOnLoggingForNum(int num). Die Methode turnOffLoggingForNum(int num) deaktiviert analog die Freischaltung der Ausgabe wieder.

²³ Dies läßt sich zwar durch "exklusiv-oder" mit einem BitSet gleicher Größe, dessen Bits alle gesetzt sind erreichen, stellt aber in der Praxis einen uneleganten Ansatz dar (erhöhter Codierungsaufwand; der Code muß entsprechend kommentiert werden, da die eigentlich gewünschte Grundoperation nicht mehr direkt nachvollziehbar ist; Laufzeiteffizienz wird beeinträchtigt, wenn bei jeder Invertierung das zweite erforderliche BitSet neu erzeugt werden muß).

Des Weiteren besitzt die Klasse `Logging` Methoden, um eine Freischaltung auch über Namen zu realisieren. Die Methode `logNumForName(String nameS)` liefert eine während der Laufzeit eindeutige Zahl für den übergebenen Namen. Sämtliche Aufrufe von `logLineForNum(String s, int num)` können mit der für diesen Namen ermittelten Zahl erfolgen. An einer völlig anderen Stelle im Code kann dann die Ausgabe für diesen Namen durch Aufruf von `turnOnLoggingForNum(int num)` aktiviert werden.

Eine Vector-Klasse für Integer-Zahlen

Die Sprache Java stellt das Konzept der Arrays (s.S. 193ff [GJS96]) bereit, um eine Menge von Variablen desselben Typs zusammenfassen zu können. Ähnlich dem `Vector` werden die Variablen eines Arrays über einen Index angesprochen. Ein Array wird einmalig angelegt mit der Kapazität, Variablen bis zu einem bestimmten Maximalindex zu verwalten.

Die dynamische Struktur vieler Algorithmen läßt es jedoch nicht zu, im voraus einen solchen Maximalindex zu ermitteln (z.B. ist bei Verwendung eines Stacks selten vorab bekannt, wieviel Elemente auf diesem abgelegt werden sollen). So wäre es oftmals wünschenswert eine Art dynamischer Arrays verwenden zu können. Diese sollten ohne eine Beschränkung der maximalen Kapazität angelegt werden und diese dann dynamisch den aktuellen Anforderungen anpassen können.

In dem von Sun im JDK 1.0.2 mitgelieferten Package `java.util` wurde zur Verwaltung von Variablen des Typs `Object` (und davon abgeleiteter Klassen) die Containerklasse `Vector` zur Verfügung gestellt, die das Gewünschte leistet.

Java ist eine stark getypte Sprache ("strongly-typed"). Es wird zwischen einfachen und Referenztypen unterschieden ("primitive types" und "reference types"). Einfache Typen sind z.B. `int`, `float`, `boolean`. Variablen vom Referenztyp können Verweise auf Objekte enthalten²⁴. Die Klasse `Vector` kann jedoch nur Variablen vom Referenztyp verwalten. Es erwies sich aber bei der Implementierung der Suchmaschine als wünschenswert, eine Funktionalität analog zur Klasse `Vector` zur dynamischen Verwaltung von Variablen des Typs `int` verwenden zu können.

Zu diesem Zweck wurde die Klasse `IntegerVector` geschaffen. Diese implementiert die Funktionalität der Klasse `Vector` zur Verwaltung von Variablen vom Typ `int`. Zusätzlich wurde eine Methode geschaffen, die einen `IntegerVector` aufsteigend sortieren kann²⁵.

3.3 Reguläre Ausdrücke

3.3.1 Verwendung des Packages `mole.TEW.Lexing`

Die Definition regulärer Ausdrücke und die Analyse von Texten nach diesen Ausdrücken wird durch die im Package `mole.TEW.Lexing` enthaltenen Klassen ermöglicht.

²⁴ Die ausführliche Darstellung des Sachverhaltes findet sich ab S. 29 in [GJS96].

²⁵ Nach Ansicht des Verfassers rührt die beschriebene Problematik aus einigen Schwachpunkten in der Konzeption der Sprache Java her. In einer vollständig objekt-orientierten Sprache (z.B. Smalltalk) wären auch die in Java verwendeten einfachen Typen als Objekte ausgeführt worden. Damit wäre es möglich gewesen, eine einzige allgemeine Containerklasse einzuführen, die Variablen beliebigen Typs verwalten kann, da letztendlich alle Typen auf die Basisklasse `Objekt` zurückgeführt werden. Die in Java vorgenommene Unterscheidung zwischen einfachen und Referenztypen ist vermutlich aus Überlegungen zur Laufzeiteffizienz vorgenommen worden.

Für die Problematik der Containerklassen bedeutet das aber, daß zu jedem der einfachen Typen eine eigene Implementierung vorgenommen werden muß. In der Codierung unterscheidet sie sich nur durch die verwendeten Variablentypen. Dies widerspricht dem Wunsch nach möglichst hoher Wiederverwendbarkeit. Eine Lösung scheint dem Verfasser nur durch Einführung generischer Konzepte in Java möglich (analog den Templates in C++).

Dabei kann in der Definitionsphase eine beliebige Menge regulärer Ausdrücke angegeben werden. Jeweils ein regulärer Ausdruck wird mit einem sog. Token verknüpft. Jedes Token wird durch eine unterschiedliche Zahl repräsentiert.

Die Analyse erfolgt dann dadurch, das ein zu untersuchender Text sequentiell in Token zerlegt wird. Für jede erkannte Teilfolge, die im Text erkannt wurde und die einem der angegebenen regulären Ausdrücke entspricht, wird als Ergebnis das korrespondierende Token geliefert.

Die Funktionalität des Packages wird nach außen durch die Klassen `Lexer`, `Lexmat` und `LexContext` bereitgestellt. Die Klasse `Lexer` dient der Definition regulärer Ausdrücke, die Klassen `Lexmat` und `LexContext` werden für die Analyse derselben eingesetzt.

In der Definitionsphase können einer Instanz der Klasse `Lexer` reguläre Ausdrücke übergeben werden, die später in der Auswertung erkannt werden sollen. Durch Aufruf der Methode `addExpressionWithToken(String expS, int acceptToken)` wird ein `String` übergeben, der einen regulären Ausdruck beschreibt und eine Zahl die das Token darstellt.

Der abschließende Aufruf der Methode `createLexmat()` erzeugt dann eine Instanz der Klasse `Lexmat`. Sie beschreibt intern einen Deterministischen Endlichen Automaten, der einen Text sequentiell analysieren und dabei Teilfolgen erkennen kann, die einem der definierten regulären Ausdrücke entsprechen.

Die Analyse von Texten erfolgt jedoch nicht direkt über den `Lexmat` sondern über eine Instanz der Klasse `LexContext`. Ein `LexContext` stellt den Zusammenhang zwischen einem speziellen Text und einem `Lexmat` her, der diesen Text in Token zerlegen soll. Dabei speichert der `Lexmat` alle notwendigen Informationen über den aktuellen Zustand der Analyse (die aktuelle Position im Text, die erkannte Teilzeichenfolge und das erkannte Token). Durch die Auslagerung des Analysezustandes in den `LexContext` konnte erreicht werden, daß mit Hilfe eines `Lexmat` gleichzeitig mehrere unterschiedliche Texte analysiert werden können.

Beispiel

Der folgende Programmauszug soll die Verwendung der erwähnten Klassen illustrieren. Dabei wird angenommen, daß eine Variable `aText` vom Typ `String` schon mit einem zu untersuchenden Text belegt ist.

Zu Beginn wird eine Instanz der Klasse `Lexer` erzeugt.

```
Lexer    theLexer = new Lexer();
```

Die Klasse `Lexer` stellt eine Konstante bereit, die als Wert des ersten Tokens verwendet werden sollte (`Lexmat.T_FIRST_TOKEN`).

```
int      token    = Lexmat.T_FIRST_TOKEN;
Lexmat   theLexmat;
LexContext lexCX;
```

Dem `Lexer` werden vier einfache Ausdrücke übergeben. Durch diese Ausdrücke sollen später die Zeichenfolgen "NOT", "AND", "OR" und beliebige vorzeichenbehaftete ganze Zahlen in einem Text erkannt werden können.

```
theLexer.addExpressionWithToken( "NOT", token++);
theLexer.addExpressionWithToken( "AND", token++);
theLexer.addExpressionWithToken( "OR", token++);
theLexer.addExpressionWithToken( "( (-[1-9] | [1-9]) [0-9]* ) | 0)",
                                token++);
```

Im Anschluß wird eine Instanz der Klasse `Lexmat` erzeugt.

```
theLexmat = theLexer.createLexmat();
```

Zur Analyse des Textes wird ein `LexContext` erzeugt, dem der erstellte `Lexmat` übergeben wird. Der `LexContext` soll mit Hilfe dieses `Lexmat` den Text `aText` untersuchen.

```
lexCX = new LexContext( theLexmat );
lexCX.setLexString( aText );
```

Der String `aText` wird im Anschluß sequentiell in Token zerlegt. Dies geschieht durch Aufruf der Methode `nextToken()` der Instanz `lexCX` des `LexContexts`. Die Klasse `Lexmat` definiert dabei einige ausgezeichnete Werte, die als Token zurückgeliefert werden können. `Lexmat.T_ERROR_TOKEN` wird geliefert, wenn eine Teilfolge im Text gefunden wurde, der kein regulärer Ausdruck zugeordnet werden konnte. `Lexmat.T_STOP_TOKEN` wird zurückgegeben, wenn das Ende des Textes erreicht wurde, die Analyse also abgeschlossen ist.

```
boolean    loopF = true;
do
{
    token = lexCX.nextToken();
    switch ( token )
    {
        case Lexmat.T_ERROR_TOKEN:
            System.out.println( "lexing:ERROR" );
            loopF = false;
            break;
        case Lexmat.T_STOP_TOKEN:
            System.out.println( "lexing:STOP" );
            loopF = false;
            break;
```

Hier wird das erkannte Token ausgegeben. Die Instanzvariable `ivLexemS` des `LexContexts` `lexCX` stellt dabei die tatsächlich erkannte Teilfolge im Text dar.

```
        default:
            System.out.println( "token=" + token + ",
                lexemS=" + lexCX.ivLexemS );//TST
    }
}
while ( loopF );
```

3.3.2 Intern verwendete Algorithmen und Datenstrukturen

Die Algorithmen lehnen sich weitestgehend an die in [ASU86] vorgestellten Verfahren an. Durch Top-Down-Parsing des an den `Lexer` übergebenen regulären Ausdrucks wird dieser in die Darstellung eines Nichtdeterministischen endlichen Automaten (NFA) umgewandelt (s.S. 121ff, [ASU86]). Die Zustände dieses Automaten werden durch Instanzen der Klasse `NFAState` repräsentiert. Instanzen der Klasse `NFA` fassen die `NFAStates` zu während der Umwandlung erzeugten Teilautomaten zusammen.

Der Aufruf der Methode `createLexmat` des `Lexers` führt dann zur Konstruktion eines deterministischen endlichen Automaten. Die Zustände eines solchen Automaten werden durch Instanzen der Klasse `DFAState` gekapselt. Eine Instanz der Klasse `Lexmat` faßt diese Zustände als Darstellung eines Automaten zusammen. Die Umwandlung des Nichtdeterministischen Endlichen Automaten geschieht dabei anhand der in [ASU86] skizzierten Verfahren (s.S. 177ff).

3.4 Suchausdrücke

3.4.1 Verwendung

Die im Package `mole.TEW.StringMatching` zusammengefaßten Klassen implementieren die Auswertung von Suchausdrücken. Sie bauen dabei auf der im vorigen Kapitel beschriebenen Funktionalität zur Auswertung regulärer Ausdrücke auf. Die Funktionalität des Packages wird nach außen durch die Klassen `EqualsStringMatcher` und `ContainsStringMatcher` bereitgestellt.

Ein `EqualsStringMatcher` wertet dabei einen Suchausdruck aus, wie er in Kapitel 2.2.3 "Suchausdrücke für Gruppennamen" beschrieben wurde. Ein solcher Suchausdruck beschreibt eine Menge alternativer Zeichenfolgen. Ein Text gilt als "passend", wenn er exakt einer dieser Zeichenfolgen entspricht (daher das Prefix `Equals` des Klassennamens).

Ein `ContainsStringMatcher` wertet Ausdrücke aus, wie sie in Kapitel 2.2.4 "Suchausdrücke für Artikel" definiert wurden. Ein Text wird dabei als "passend" bewertet, wenn die Auswertung des logischen Ausdrucks, der über das Enthaltensein (daher das Prefix `Contains` des Klassennamens) von Zeichenfolgen im Text gebildet wurde, "wahr" ergibt.

Wird eine Instanz einer der beiden Klassen erzeugt, so wird dem Konstruktor ein `String` übergeben, der den von dieser Instanz auszuwertenden Ausdruck beschreibt.

Beide Klassen wurden von der abstrakten Klasse `StringMatcher` abgeleitet. Diese Klasse definiert die Methode

```
abstract public boolean matchString( String theString );
```

die für Instanzen der erbdenden Klassen implementiert werden muß. Dieser Methode wird als Parameter ein `String` übergeben, der den zu untersuchenden Text darstellt. Die Methode wertet daraufhin den Suchausdruck aus, für den sie erzeugt wurde und liefert einen Wahrheitswert, der angibt, ob der untersuchte Text "passend" war.

3.4.2 Interne Realisierung

Beide `StringMatcher` interpretieren bei Aufruf ihres Konstruktor den diesem übergebenen Ausdruck und wandeln ihn in eine interne Darstellung um, die später beim Aufruf der `matchString`-Methode besonders effizient ausgewertet werden kann. Die Ausdrücke wurden in den zuvor genannten Kapiteln des Entwurfsabschnittes dieser Studienarbeit durch Grammatiken spezifiziert. Interpretation und Umwandlung der Ausdrücke werden folglich durch Implementierung eines "Top-Down-Parsing"-Verfahrens realisiert. Die Zerlegung des Suchausdrucks in entsprechende Token kann, dank der zur Auswertung regulärer Ausdrücke geleisteten Vorarbeit, sehr elegant durch Erzeugen entsprechender Instanzen der Klassen `Lexer`, `Lexmat` und `LexContext` geleistet werden.

Der `EqualsStringMatcher`

Im Falle des `EqualsStringMatchers` muß zur Auswertung des Suchausdrucks nur ein `Lexmat` erzeugt werden. Die von einem `EqualsStringMatcher` beschriebenen Suchausdrücke bestehen aus einer Liste von Teilausdrücken, die Zeichenfolgen definieren. Diese Ausdrücke müssen zum Teil noch in reguläre Ausdrücke umgewandelt werden (SimpleStrings z.B. müssen von Anführungszeichen umschlossen werden) bevor sie einem `Lexer` übergeben werden können. Nachdem dies mit dem letzten Teilausdruck eines Suchausdrucks geschehen ist, wird von diesem `Lexer` ein `Lexmat` erzeugt.

Die Auswertung eines Textes durch die `matchString`-Methode erfordert dann nur den Test, ob der `Lexmat` eine durch einen regulären Ausdruck definierte Zeichenfolge im Text erkannt hat und ob diese den gesamten Text einschließt (der `Lexmat` könnte schließlich eine gültige Teilfolge zu Beginn

des Textes erkannt haben, die von ungültigen Zeichen gefolgt wird). Die Implementation der matchString-Methode besteht deshalb nur aus wenigen Zeilen.

```
public boolean matchString( String s )
{
    LexContext    lexCX = new LexContext( ivTokenLexmat, s );

    /* Eines der in Expression definierten Token muß gefunden
       worden sein und den gesamten String qualifizieren.
    */
    return lexCX.nextToken()>=T_FIRST_TOKEN &&
           lexCX.nextToken()<=Lexmat.T_STOP_TOKEN;
}
```

Der ContainsStringMatcher

Ein ContainsStringMatcher erzeugt ebenfalls einen Lexmaten, der die in der Grammatik als SimpleExpr bezeichneten Ausdrücke, die Teilzeichenfolgen repräsentieren, in einem Text erkennen kann. Der eigentliche Suchausdruck wird für die Auswertung in eine Baumstruktur umgewandelt, die durch Instanzen verschiedener Klassen dargestellt wird. Diese Baumstruktur entspricht dabei der implizit durch die Grammatik des Ausdrucks definierten ("Parse-Tree"). Die folgende Abbildung stellt die Vererbungshierarchie dieser Klassen dar.

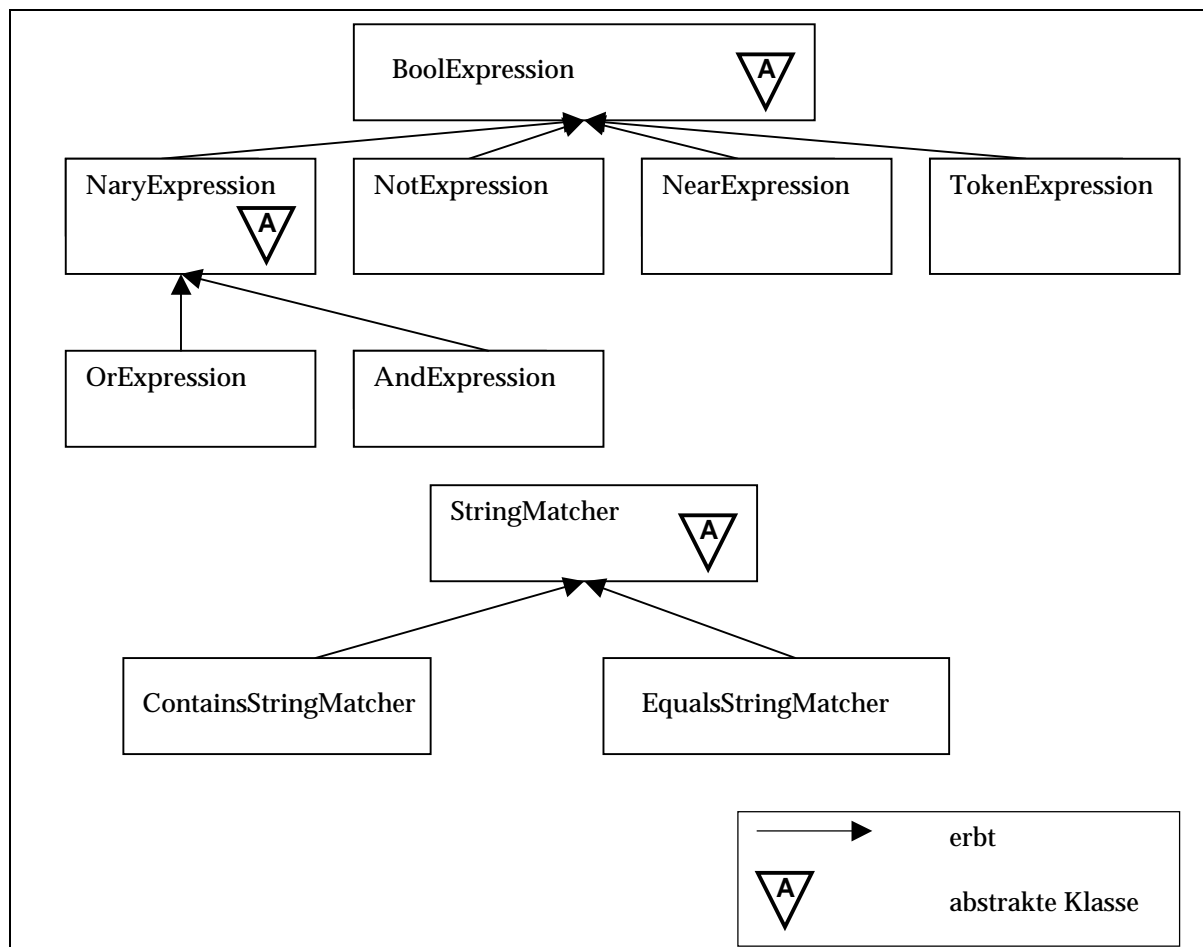


Abb. 7: Vererbungshierarchie der Expression- und StringMatching-Klassen

Alle Klassen werden von der abstrakten Basisklasse `BoolExpression` abgeleitet. Diese definiert zwei Methoden:

- `abstract public void reset()`

Wird vor der Auswertung eines Textes vom `StringMatcher` aufgerufen und setzt den internen Zustand einer Instanz zurück.

- `abstract public boolean evaluateWithContext(Object context)`

Mit den durch das Objekt `context` zur Verfügung stehenden Informationen wird die Methode durch eine Instanz ausgewertet.

Die Auswertung der Methode `matchString` wird nun folgendermaßen durchgeführt. Mit Hilfe des erzeugten Lexmaten wird der zu untersuchende Text in Token zerlegt. Für jede der durch `SimpleExpr` im Suchausdruck definierten Zeichenfolgen, die im Text erkannt wurde, wird die Tatsache vermerkt, daß sie gefunden wurde, sowie die Position, an der dies im Text geschah.

Anschließend wird die Methode `evaluateWithContext` des Wurzelobjekts der erzeugten Baumstruktur von `Expression`-Instanzen mit der Instanz des `ContainsStringMatchers` als Argument aufgerufen. Diese Instanz stellt also den Kontext dar, der alle zur Auswertung einer `Expression` zusätzlich benötigten Informationen bereitstellt. Dabei handelt es sich um die erwähnten Informationen über Auftreten und Position einer erkannten Zeichenfolge, die vom `ContainsStringMatcher` vermerkt wurden. Die Auswertung der `evaluateWithContext`-Methode hängt nun von der jeweiligen `Expression`-Klasse ab.

- `TokenExpression` liefert `true`, wenn der `ContainsStringMatcher` die mit dieser Instanz korrespondierende Teilfolge im Text gefunden hat.
- `NotExpression` liefert `true`, wenn die von ihr referenzierte `Expression`-Instanz, welche die Wurzel eines Teilbaums darstellt, bei Aufruf von `evaluateWithContext` `false` liefert. Andernfalls liefert eine `NotExpression` `true`.
- `NearExpression` liefert `true`, wenn die beiden von ihr referenzierten `TokenExpressions` `true` liefern und die mit diesen korrespondierend Zeichenfolgen im Text weniger als zehn Zeichen voneinander entfernt sind.
- `AndExpression` liefert `true`, wenn alle von ihr referenzierten `Expression`-Instanzen bei Aufruf von `evaluateWithContext` `true` liefern.
- `OrExpression` schließlich liefert `true`, wenn eine der von ihr referenzierten `Expression`-Instanzen bei Aufruf von `evaluateWithContext` `true` liefert.

Das Ergebnis des Aufrufs der `evaluateWithContext`-Methode des Wurzelobjektes wird als Ergebnis der `matchString`-Methode zurückgegeben.

3.5 News-Anbindung

Die Klassen im Package `mole.NewsAdaptor` implementieren den Zugriff auf News, dargestellt durch Gruppen und Artikel. Diese werden dabei auf die Klassen `NewsGroup` und `NewsArticle` abgebildet. Vom Zugriff auf Gruppen und Artikel wird durch ein spezielles `NewsAdaptor`-Interface abstrahiert. Die durch dieses Interface definierten Methoden können dann auf beliebige Weise von verschiedenen Klassen implementiert werden. Dank des `NewsAdaptor`-Interfaces werden die Details des Zugriffs auf Gruppen und Artikel vor den "Nutzern" einer dieses Interface implementierenden Klasse verborgen. So könnte eine neu entwickelte `NewsAdaptor`-Klasse den Zugriff auf Artikel und Gruppen per NNTP lösen und problemlos in eine existierende Architektur eingebunden werden.

Im Detail werden durch das `NewsAdaptor`-Interface folgende Methoden definiert:

```
int    numberOfGroups()
```

Liefert die Anzahl der verfügbaren Gruppen.

```
String[]    groupNames();
```

Liefert ein `String`-Array mit den Namen aller verfügbaren Gruppen.

```
NewsGroup    groupNamed(String groupName);
```

Liefert eine `NewsGroup` mit Namen `groupName` oder `null`

```
String[]    articleIDsForGroupNamed( String groupName );
```

Liefert ein `String`-Array mit den IDs aller verfügbaren Artikel für die Gruppe mit Namen `groupName`.

```
byte[]    rawDataForArticleWithIDFromGroup
           ( String articleID, NewsGroup theGroup );
```

Liefert ein `byte`-Array mit dem unverarbeiteten Inhalt des Artikels mit ID `artikelID` aus der `NewsGroup theGroup`.

```
byte[] rawHeaderDataForArticleWithIDFromGroup
           ( String articleID, NewsGroup theGroup );
```

Liefert die Daten eines Artikel-Headers mit ID `articleID` aus der `NewsGroup theGroup` oder `null`.

In Rahmen dieser Arbeit wurde eine erste Implementierung dieses Interfaces durch eine Klasse namens `FileNewsAdaptor` geschaffen. Die Voraussetzungen für den Einsatz eines `FileNewsAdaptors` wurden in Kapitel 2.1.2 "Realisierung der Newsserver-Dienste" genau spezifiziert. Durch die erwähnten Voraussetzungen ist der folgende Sachverhalt gegeben:

- Die Information über die verfügbaren Gruppen liegen in einer Datei mit den Namen "active" vor.
- Die Artikel werden in einem Verzeichnisbaum abgelegt, der den Hierarchiecharakter der Gruppenbezeichnungen durch eine entsprechende Folge von ineinander verschachtelten Verzeichnisses widerspiegelt.
- Jeder Artikel besitzt eine Nummer als Dateinamen, die später als eindeutige ID (eindeutig innerhalb der jeweiligen Gruppe) verwendet werden kann.

Die Information über das Verzeichnis, das die "active"-Datei enthält sowie die Angaben des Basisverzeichnisses unter dem die Artikel abgelegt wurden muß bei Erzeugung einer `FileNewsAdaptor`-Instanz deren Konstruktor übergeben werden. Die folgende Zeile zeigt die Definition des Konstruktors.

```
public FileNewsAdaptor( String confPath, String basePath )
```

Primäre Aufgabe des `NewsAdaptor`-Interfaces ist es über die Methode `groupNames()` eine Liste mit den Namen der verfügbaren Gruppen zu liefern und über die Methode `groupNamed(String groupName)` ein `NewsGroup`-Objekt für eine Gruppe mit dem übergebenen Namen `groupName` zurückzugeben.

Der weitere Zugriff auf Artikel erfolgt dann über die entsprechende `NewsGroup`-Instanz. Diese kann eine Liste mit IDs (Methode `articleIDs()`) verfügbarer Artikel liefern und zusätzlich zu einer ID die entsprechende Instanz eines `NewsArticles` (Methode `articleWithID(articleID)`).

Eine Instanz eines `NewsArticles` soll hauptsächlich den Header des Artikels bereitstellen. Dieser soll dabei auf eine Weise vorverarbeitet werden, die über die Bezeichnung eines Header-Feldes (Field-Name) einen einfachen Zugriff auf den Inhalt des Feldes (Field-Body) ermöglicht. Darüber hinaus soll der Field-Body so umgewandelt werden, daß sein Text durch den Unicode kodiert wird, er also als `Java-String` repräsentiert werden kann. Dazu ist es erforderlich, die entsprechenden MIME-Kodierungen erkennen und umwandeln zu können. In der vom Verfasser realisierten Implementierung wird dabei nur das im MIME-Standard spezifizierte "Q"-Encoding beachtet. Die Vorverarbeitung des Header geschieht durch einen einfachen Parsing-Vorgang, gemäß den in [RFC822] genannten Regeln für das sog. Simple-Header-Parsing.

Die Methode `headerTable()` eines `NewsArticles` liefert ein nach dem erwähnten Verfahren der Vorverarbeitung erstelltes `Hashtable`, dessen Einträge aus den Field-Bodies der Header-Felder eines Artikels besteht. Die Einträge wurden unter den Field-Names der Header-Felder vorgenommen.

Der Zugriff auf die Subject-Zeile eines durch eine `NewsArticle`-Instanz repräsentierten Artikels läßt sich also durch eine einzige Zeile Code realisieren.

```
System.out.println( theArticle.headerTable().get( "subject" ) )
```

3.6 Vereinfachter Aufruf von RPCs

Problematik

In Kapitel 1.2.3 "Die Architektur von Mole" wurde ein spezieller Mechanismus erwähnt, über den in Mole der Aufruf von Methoden, die Agenten bereitstellen, realisiert wurde. Ein Agent, der eine Methode eines anderen Agenten aufrufen will, muß dazu eine Instanz der Klasse `RPCMessage` erzeugen. Beim Aufruf des entsprechenden Konstruktors ist eine Vielzahl von Parametern zu übergeben. Jeweils Name und Ort des Senders und Empfängers der Nachricht, ein Wert, der angibt, was im Fehlerfall zu tun ist sowie der Name der vom Empfänger auszuführenden Methode (im folgenden zur besseren Unterscheidung auch RPC-Methode genannt) und die an diese zu übergebenden Parameter. Diese `RPCMessage` wird daraufhin durch die Methode `call` der `Location`, an der sich der sendende Agent befindet an den Empfänger geschickt und dort ausgewertet. Zum Schluß wird das Ergebnis zurückgeliefert.

Das folgende Programmfragment zeigt eine typische Realisierung eines Methodenaufrufs. Dabei wird angenommen, daß das Fragment innerhalb einer Methode eines Agenten aufgerufen wird. Die Variablen `myname` und `actuallocation` sind Instanzvariablen des Agenten und mit gültigen Werten vorbelegt. Ferner wird vorausgesetzt, daß die Variablen `receiverName` und `receiverLocationName` ebenfalls mit entsprechenden gültigen Werten vorbelegt sind.

```
Object[]    argV;
Object      result;
RPCMessage  rpcMsg;

argV = new Object[2];
argV[0] = "Test";
argV[1] = new Integer( 42 );
rpcMsg = new RPCMessage
(
    myname,
    actuallocation.locationName(),
    receiverName,
    receiverLocationName,
    1,
    "testWithStringAndInteger",
    argV
);
result = actuallocation.call( rpcMsg );
```

Der Aufruf einer RPC-Methode eines anderen Agenten mit zwei Argumenten erstreckt sich über mehrere Zeilen. Dies bedeutet, verglichen mit dem herkömmlichen Aufwand bei der Formulierung eines Methodenaufrufs, einen deutlichen erhöhten Codierungsaufwand. Auch die Lesbarkeit des Programms leidet darunter.

Lösung

Der Verfasser hat aufgrund dieser Überlegungen eine neue von der Klasse `RPCMessage` abgeleitete Klasse namens `XtRPCMessage` geschaffen, die Klassenmethoden für einen vereinfachten Aufruf von RPC-Methoden²⁶.

Dabei wird zwischen dem Aufruf von RPC-Methoden eines Agenten, der sich an derselben `Location` wie der aufrufende Agent und dem Aufruf von RPC-Methoden, bei denen sich Sender und Empfänger an verschiedenen `Locations` befinden, unterschieden. Ersteres wird durch eine Reihe von Klassenmethoden namens `localCall`, letzteres über Klassenmethoden mit den Namen `call` realisiert.

Von jeder dieser Methoden existieren Varianten gleichen Namens, die sich allein durch die Anzahl der zu übergebenden Parameter unterscheiden (Stichwort "Überladen"). Dahinter verbirgt sich der Grundgedanke, das "Verpacken" der zu übergebenden Parameter in ein Array nicht mehr selbst vorzunehmen, sondern sie den neuen Methoden als Parameter zu übergeben und von diesen "verpacken" zu lassen.

Zur Illustration sei die Deklaration zweier `localCall`-Methoden dargestellt.

```
public static Object localCall( Agent callingAgent,
                               AgentName receiverAgent,
                               String procedureName )

public static Object localCall( Agent callingAgent,
                               AgentName receiverAgent,
                               String procedureName, Object arg0 )
```

Die beiden unterscheiden sich dadurch, daß die zweite Methode einen weiteren Parameter erhält. Über die erste Methode wird der Aufruf einer RPC-Methode ohne Argumente, über die zweite Methode der Aufruf mit einem Argument realisiert. Analog zu diesem Schema wurden Methoden für den Aufruf von RPC-Methoden mit bis zu drei Argumenten realisiert (für mehr Argumente ergab sich in dieser Studienarbeit kein Bedarf, eine Erweiterung nach diesem Schema ist aber trivial).

Der gegenüber dem anfangs vorgestellten Programmfragment vorgestellte Codierungsaufwand reduziert sich auf eine Zeile.

```
Object      result;

result     = XtRPCMessage.localCall( this, receiverName,
                                     "testWithStringAndInteger",
                                     "Test", new Integer( 42 ) );
```

Statt den `AgentName` und `LocationName` des aufrufenden Agenten wird eine Referenz auf den Agenten übergeben (`this`). Die Methode `call` kann anhand dieser Referenz die beiden erstgenannten Parameter ermitteln (dies geschieht mit Hilfe von Instanzvariablen des Agenten). Weiterhin wird ausgenutzt, daß sich beide Agenten an derselben `Location` befinden. Es ist daher nur notwendig, den `AgentName` des Agenten anzugeben, der die RPC-Methode ausführen soll.

Die neu geschaffenen Klassenmethoden erzeugen automatisch eine `RPCMessage`, die alle notwendigen Informationen erhält und lösen den RPC-Mechanismus durch Aufruf der Methode `call` der `Location` des aufrufenden Agenten automatisch aus.

²⁶ Dem Verfasser steht zwar der Quellcode des Prototypen von Mole zur Verfügung, so daß auch eine direkte Modifikation von Klassen möglich gewesen wäre. Da dadurch eine Zusammenführung aller auf dem System Mole aufbauenden Entwicklungen unnötig erschwert worden wäre, hat der Verfasser davon abgesehen, diesen Weg einzuschlagen.

Zum Aufruf einer RPC-Methode eines Agenten, der sich an einer anderen Location als der aufrufende Agent befindet wurde analog den `localCall`-Methoden eine Reihe von `call`-Methoden geschaffen. Diese erhalten als weiteren Parameter den `LocationName` des Agenten, der die RPC-Methode ausführen soll. Zur Verdeutlichung soll abschließend die Deklaration einer solchen Methode gezeigt werden, die genau ein Argument als Parameter erhält.

```
public static Object call( Agent callingAgent,
                        AgentName receiverAgent,
                        LocationName receiverLocation,
                        String procedureName,
                        Object arg0 )
```

3.7 Der Gateway-Agent für Newsserver-Dienste

Aufbauend auf der zur Anbindung von News implementierten Infrastruktur läßt sich der Gateway-Agent für Newsserver-Dienste, der in das Mole-System integriert werden muß, relativ einfach realisieren.

Der Agent wird durch Instanzen der Klasse `NewsService` repräsentiert. Diese wurde direkt von `SystemAgent` abgeleitet. Ein `NewsService` meldet sich, nachdem er von einer `Location` gestartet wurde, bei dieser als Anbieter des Dienstes "News" an. Beim Erzeugen eines `NewsService` müssen dem Konstruktor der Pfad des Verzeichnisses in dem sich die Datei mit den Namen aller verfügbaren Gruppen befindet sowie der Pfad der Wurzel des Verzeichnisbaums, in dem alle Artikel abgelegt sind, übergeben werden. Dadurch kann ein `NewsService` einen `FileNewsAdaptor` erzeugen, über welchen der gesamte Zugriff auf News abgewickelt wird.

Als Anbieter des Dienstes "News" stellt ein `NewsService` anderen Agenten über den RPC-Mechanismus insgesamt 4 RPC-Methoden zur Verfügung.

- `groupNames`

Wird ohne Parameter aufgerufen und liefert einen `Vector`, der die Namen aller verfügbaren Gruppen als `Strings` enthält

- `articleIDsForGroupNamed`

Erhält als Parameter den Namen einer Gruppe. Liefert als Ergebnis einen `Vector`, der die IDs aller verfügbaren Artikel dieser Gruppe enthält.

- `headerTableForArticleWithIDfromGroupNamed`

Wird mit der ID eines Artikels und dem Namen einer Gruppe als Parametern aufgerufen. Die Methode gibt ein `Hashtable` zurück, das den im Kapitel 3.5 "News-Anbindung" beschriebenen Aufbau besitzt.

- `rawDataForArticleWithIDfromGroupNamed`

Wird mit der ID eines Artikels und dem Namen einer Gruppe als Parametern aufgerufen. Als Ergebnis des Aufrufs wird ein `Byte-Array` geliefert, das die unbearbeiteten Daten des Artikels enthält.

Intern werden alle diese Methoden auf die in Kapitel 3.5 "News-Anbindung" genannten Mechanismen abgebildet.

3.8 Ein einfacher Trading-Systemagent

Für die Vermittlung von Diensten zwischen verschiedenen Locations wird ein sog. Trading-Agent eingesetzt. Im Rahmen dieser Studienarbeit wurde eine sehr einfache Implementierung eines solchen Trading-Agenten geschaffen. Im wesentlichen verwaltet dieser Agent eine Liste von Orten, die jeweils einen bestimmten Dienst zur Verfügung stellen.

Der Agent wird durch Instanzen der Klasse `SimpleTrader` realisiert, die von `SystemAgent` abgeleitet wurden. Der `SimpleTrader` meldet sich bei der Location, von der er gestartet wurde als Anbieter des Dienstes "Trading" an. Mobile Agenten, die sich ebenfalls an dieser Location aufhalten, können über den RPC-Mechanismus von Mole die Methode `locationsForService` aufrufen. Dabei wird als Argument ein String übergeben, der den Namen eines Dienstes darstellt. Als Ergebnis des Aufrufs wird eine Liste mit `LocationNames` von Locations geliefert, die diesen Dienst anbieten.

Konfiguration

Ein `SimpleTrader` kann durch zwei Methoden konfiguriert werden. Der Aufruf der Methode

```
registerLocationForService( LocationName locationName,  
                           String serviceNameS)
```

erstellt einen Eintrag, in dem vermerkt wird, daß die Location mit Namen `LocationName` den durch `serviceNameS` bezeichneten Dienst bereitstellt. Analog kann durch Aufruf der Methode

```
unregisterLocationForService( LocationName locationName,  
                              String serviceNameS)
```

dieser Eintrag wieder gelöscht werden.

Darüber hinausgehende Mechanismen (automatischer Austausch von Konfigurationen) wurden nicht implementiert.

Integration in das System

Zum Betrieb der Suchmaschine ist es erforderlich, daß sich an der Location, an der ein Suchvorgang initiiert wird, eine vollständig konfigurierte Instanz eines `SimpleTraders` befindet. Der in Kapitel 1.2.3 "Die Architektur von Mole" beschriebene Mechanismus zur initialen Konfiguration einer Location mit Agenten sieht dafür das Anlegen einer Datei vor, die persistente Abbilder der gewünschten Agenten enthält. Ein Programm, das eine solche Datei generiert, muß daher um ein paar Zeilen erweitert werden, die eine Instanz der Klasse `SimpleTrader` erzeugen und diese korrekt konfigurieren. Das folgende Programmfragment soll diesen Vorgang illustrieren.

```
/* Trader erzeugen und NewsServer anderer Locations registrieren  
*/  
SimpleTrader    myTrader = new SimpleTrader();  
  
myTrader.registerLocationForService( new LocationName  
    ( "location2.mole.informatik.uni-stuttgart.de" ), "News" );  
  
myTrader.registerLocationForService( new LocationName  
    ( "location3.mole.informatik.uni-stuttgart.de" ), "News" );
```

3.9 Sequentielle und Parallele Suche

3.9.1 Allgemeines

Die in Kapitel 2.1.3 "Entwurf des Suchverfahrens" beschriebenen Verfahren wurden durch Implementierung mehrerer Klassen realisiert.

Es stellte sich heraus, daß sich die beiden Suchverfahren auf einen gemeinsamen Algorithmus zurückführen lassen, der in einer äußeren Schleife die "passenden" Gruppennamen und in einer inneren Schleife die "passenden" Artikel ermittelt. Sequentielle und parallele Suche unterscheiden sich in ihrer darauf aufbauenden Implementierung nur in den Verfahren zur Auswertung der Terminierungskriterien und bei der Speicherung der Ergebnisse.

Als Basisklasse wurde daher die Klasse `NewsSearchAgent` geschaffen. Diese definiert zwei abstrakte Methoden, welche von den von ihr abgeleiteten Klassen `ParNewsSearchAgent` (für die parallele Suche) und `SeqNewsSearchAgent` (für die sequentielle Suche) zu implementieren sind. Der in `NewsSearchAgent` realisierte Suchalgorithmus baut auf diesen Methoden auf.

Die Koordination der Suche paralleler Agenten wird von der Klasse `SearchController` übernommen. Sie stellt also den im Entwurf genannten Koordinator dar. Weiterhin wird über die Klasse `SearchController` der Start sowohl der sequentiellen als auch der parallelen Suche durch Erzeugen entsprechender Suchagenten ausgelöst.

Der Verfasser zeigt im folgenden, im Gegensatz zu den bisher dargestellten Code-Fragmenten, zum besseren Verständnis der Suchverfahren, die wesentlichen Ausschnitte des Quellcodes der Klassen `NewsSearchAgent`, `ParNewsSearchAgent` und `SearchController`, die zusammen das parallele Suchverfahren realisieren. Das sequentielle Verfahren wird, aufgrund seiner einfachen Terminierungskriterien, die autonom vom suchenden Agenten ausgewertet werden, nicht näher dargestellt.

3.9.2 Kommentierter Quellcode

Initialisierung der Suche durch den SearchController

Im folgenden wird die Methode des `SearchControllers` gezeigt, mit Hilfe derer die `ParNewsSearchAgents` an die verschiedenen verfügbaren `Locations`, die Usenet-News anbieten, ausgesendet werden.

Die Instanzvariable `ivHitLimit` enthält bei Eintritt in die Methode den aktuellen Wert der maximal gewünschten Treffer, die Variable `ivTimeLimitMillis` die Zeitspanne in Millisekunden, die von den Suchagenten für die Suche aufgewendet werden darf. Über diese Werte wird die Begrenzung des Suchvorgangs vorgenommen.

```
void launchParallelSearch( EqualsStringMatcher groupNameEqualsSM,
                          ContainsStringMatcher subjectContainsSM )
{
    Vector          locationNameV;
    LocationName   homeName = actuallocation.locationName();
    Enumeration    enum;
```

Zu Beginn werden zwei `Hashtables` initialisiert. Diese dienen der Koordination der Suche und der Speicherung des Suchergebnisses. In das `ivGroupNameTable` wird jede Gruppe eingetragen, die von

einem `ParNewsSearchAgents` bearbeitet wird. Das `ivResultTable` nimmt Referenzen auf die gefundenen Artikel auf.

```
ivGroupNameTable = new Hashtable();
ivResultTable    = new Hashtable();
```

Weiterhin wird die Instanzvariable `ivHitCount` auf 0 gesetzt. Diese zählt die von den Suchagenten gefundenen Treffer und hilft dadurch bei der Auswertung der Suchbegrenzung.

```
ivHitCount = 0;
```

Im Anschluß werden durch Kommunikation mit dem lokalen Trading-Agenten die Orte ermittelt, die den Dienst "News" anbieten.

```
locationNameV = getNewsServiceLocationNames();
```

Dies wurde in einer eigenen Methode (`getNewsServiceLocationNames()`) des `SearchControllers` implementiert, da er sowohl die parallele als auch die sequentielle Suche auslöst und er diese Funktionalität in beiden Fällen benötigt. Die Methode wird hier als Einschub gezeigt.

```
private Vector getNewsServiceLocationNames()
{
    AgentName[]    agentNameA;
    AgentName      traderName;
    Vector         locationNameV;

    /* Trading-Agent ermitteln
    */
    if( (agentNameA = actuallocation.serviceProvidersOf
        ( "Trading" ))==null || agentNameA.length==0 )
    {
        return null;
    }
    traderName = agentNameA[0];
    /* Orte mit "News"-Diensten ermitteln
    */
    try
    {
        locationNameV =
            (Vector)XtRPCMessage.localCall
            ( this, traderName, "locationsForService", "News" );
    }
    catch (Exception e)
    {
        //Ignore
        locationNameV = null;
    }
    return locationNameV;
}
```

Die folgenden Auszüge stellen die von obigem Einschub unterbrochene Fortsetzung der `launchParallelSearch`-Methode dar. In einer Schleife wird für jede ermittelte `Location` ein `ParNewsSearchAgent` erzeugt und bei der `Location` angemeldet.

```

enum = locationNameV.elements();
while ( enum.hasMoreElements() )
{
    LocationName      targetName;
    ParNewsSearchAgent agent;

    targetName = (LocationName)enum.nextElement();
    agent      = new ParNewsSearchAgent
        (   groupNameEqualsSM, subjectContainsSM,
            myname, homeName, targetName, ivTimeLimitMillis );
    actuallocation.createAgent( agent );
}
}

```

Start und Migration des `ParNewsSearchAgent`

Der `ParNewsSearchAgent` "merkt" sich über die Instanzvariable `ivState`, in welchem Zustand er sich befindet. Je nach Zustand löst er bei Aktivierung durch das Agentensystem dann die entsprechende Aktion aus. Die Aktivierung geschieht durch Aufruf der Methode `start()`. Wie schon im Grundlagenteil dieser Arbeit dargelegt, handelt es sich dabei um ein Callback-Methode. Diese wird aufgerufen, wenn ein neu erzeugter bzw. migrierter Agent bei einer `Location` registriert wurde. Ein neu erzeugter `ParNewsSearchAgent` besitzt direkt nach der Erzeugung durch Aufruf seines Konstruktors den Wert `cStartState`. Im ersten Schritt migriert er daher, zustandsgesteuert zu seiner Ziel-`Location`. Dort führt er die Suche aus und kehrt im Anschluß daran wieder an seinen Ausgangspunkt zurück, an dem er sich beim `SearchController` abmeldet.

```

public void start()
{
    switch ( ivState )
    {
        case cStartState:
            ivState = cAtTargetState;
            actuallocation.goTo( this, ivTarget );
            break;
        case cAtTargetState:
            searchForArticles();
            ivState = cMissionCompleteState;
            actuallocation.goTo( this, ivHome );
            break;
        case cMissionCompleteState:
            try
            {
                XtRPCMessage.call( this, ivSearchController, ivHome,
                                   "searchDone" );
            }
            catch( Exception e )
            {
                //Ignore
            }
            actuallocation.die( this );
            break;
    }
}
}

```

Das Suchverfahren des NewsSearchAgenten

An der Ziel-Location wird die Suche durchgeführt. Der einer Suche zugrundeliegende Algorithmus ist dabei, wie eingangs erwähnt, bei beiden Suchverfahren identisch und wird deshalb durch die Methode `searchForArticles` in der Basisklasse `NewsSearchAgent` realisiert, von der dann jeweils die Klasse für das konkrete Suchverfahren abgeleitet werden muß.

```
protected void searchForArticles()
{
    AgentName[]    agentNameA;
    AgentName      newsServiceName;
    String[]       groupNameA;
    int            i;
    long           startMillis;
```

Zu Beginn des Suchvorgangs wird die aktuelle Systemzeit ermittelt. Über diesen Zeitpunkt wird später festgestellt, ob das durch den Wert der Instanzvariablen `ivTimeLimitMillis` definierte Zeitlimit überschritten wurde, die Suche also abubrechen ist.

```
startMillis      = System.currentTimeMillis();
```

Im folgenden wird Kontakt zum Gateway-Agenten aufgenommen, der den Service "News" bereitstellt. Über den RPC-Methodenaufruf werden die verfügbaren Gruppen ermittelt.

```
agentNameA       = actuallocation.serviceProvidersOf( "News" );
if( agentNameA!=null && agentNameA.length>0 )
{
    newsServiceName = agentNameA[0];
    try
    {
        groupNameA = (String[])XtRPCMessage.localCall
            ( this, newsServiceName, "groupNames" );
```

In der äußeren Schleife werden alle Gruppennamen überprüft.

```
for ( i=0; i<groupNameA.length; i++ )
{
```

Wird ein "passender" Gruppenname ermittelt, muß getestet werden, ob die Gruppe durchsucht werden soll. Dies geschieht durch Aufruf der Methode `mustSearchInGroupNamed`. Diese wurde für die Klasse `NewsSearchAgent` abstrakt deklariert und wird von den Klassen `ParNewsSearchAgent` und `SeqNewsSearchAgent` jeweils unterschiedlich implementiert.

```
if ( ivGroupNameEqualsSM.matchString( groupNameA[i] ) &&
    mustSearchInGroupNamed( groupNameA[i] ) )
{
    int          j;
    String[]     articleIdA;
```


Implementierung der abstrakten Methoden des Suchagenten

Die Klasse `NewsSearchAgent` hat, wie bereits erwähnt, zwei abstrakte Methoden definiert, die während des Suchverfahrens verwendet wurden. In der Klasse `ParNewsSearchAgent` werden diese Methoden durch Aufruf der RPC-Methoden des `SearchControllers` realisiert.

```
protected boolean mustSearchInGroupNamed( String groupName )
{
    try
    {
        return ((Boolean)XtRPCMessage.call
            ( this, ivSearchController,
              ivHome, "examineGroupNamed",
              groupName )).booleanValue();
    }
    catch( Exception e )
    {
        //Ignore
        return false;
    }
}

protected boolean checkStopForFoundArticleWithIDFromGroupNamed
    ( String articleID, String groupName )
{
    try
    {
        return ((Boolean)XtRPCMessage.call
            ( this, ivSearchController, ivHome,
              "checkStopForFoundArticleWithIDFromGroupNamed",
              articleID, groupName )).booleanValue();
    }
    catch( Exception e )
    {
        //Ignore
        return true;
    }
}
```

Die RPC-Methoden des SearchControllers

In der `dispatch`-Methode des `SearchControllers` wird der Aufruf von Methoden über den für Mole spezifischen RPC-Mechanismus aufgelöst. Diese werden hier abschließend gezeigt. Durch Aufruf der Methode `examineGroupNamed()` übergibt ein `ParNewsSearchAgent` dem `SearchController` die Information über eine für die Recherche nach Artikeln "passende" Gruppe. Falls in dieser Gruppe noch nicht von einem anderen Agenten gesucht wird, gibt der `SearchController` den Wahrheitswert `true` zurück.

```
// -----
// Agent Methods (overridden)
// -----

synchronized public Object dispatch( RPCMessage rpcMsg)
    throws RPCParameterWrongException
{
    /* Für eine robuste Implementierung sollte man die Typen und
       Anzahl der Parameter prüfen. Vielleicht sollte dafür ein
       Konzept in Mole integriert werden. Andererseits müsste ja
```

```

    dank RMI in JDK 1.1 die Sache auch eleganter gelöst werden
    können.
    */
    if (rpcMsg.procedure.equals( "examineGroupNamed" ) )
    {

```

Hier wird das zu Beginn erwähnte `ivGroupNameTable` konsultiert, um dem Suchagenten mitzuteilen, ob er die entdeckte "passende" Gruppe nach Artikeln durchsuchen muß.

```

        if ( ivGroupNameTable.get( (String)rpcMsg.argv[0] )!=null )
        {
            return new Boolean( false );
        }
        else
        {
            ivGroupNameTable.put
                ( (String)rpcMsg.argv[0],(String)rpcMsg.argv[0] );

            return new Boolean( true );
        }
    }
}

```

Die RPC-Methode `checkStopForFoundArticleWithIDFromGroupNamed` wird von einem `ParNewsSearchAgent` aufgerufen, wenn er einen Artikel gefunden hat. Der `SearchController` speichert den Verweis auf Gruppenname und Artikel-ID und wertet das Kriterium der Trefferbegrenzung aus. Sobald die Zahl maximaler Treffer überschritten wurde, bekommt der Suchagent den Wahrheitswert `true` zurückgeliefert, als Information, daß die Suche sofort abubrechen ist.

```

    else if (rpcMsg.procedure.equals
        ("checkStopForFoundArticleWithIDFromGroupNamed" ) )
    {
        /* Wenn Ergebnislimit erreicht sollen die ParNewsSearchAgents
           nicht mehr weitersuchen
        */
        if (ivHitCount == ivHitLimit)
        {
            return new Boolean( true );
        }
        addArticleWithIDFromGroupNamedToResult( (String)rpcMsg.argv[0],
                                                (String)rpcMsg.argv[1] );
        ivHitCount++;
        return new Boolean( false );
    }
}

```

3.10 Die Benutzerschnittstelle

Aufbauend auf den GUI-Klassen²⁷, die in dem von Sun bereitgestellten Package `java.awt` zusammengefaßt sind, wurde das im Entwurf beschriebene GUI realisiert. Dabei wurde ein GUI-Builder der Firma Metrowerks (weitere Informationen dazu in [MW97]) namens *Constructor* verwendet. Dieser erlaubt das komfortable, interaktive Entwerfen einer Benutzeroberfläche. Zum Abschluß des Entwurfsvorgangs kann der GUI-Builder eine Datei mit Java-Sourcecode erzeugen, der von den in `java.awt` definierten Klassen abgeleitete Klassen definiert, welche nach ihrer Instanziierung

²⁷ GUI, die Abkürzung von Graphical User Interface, ist der englische Fachbegriff für eine grafische Benutzerschnittstelle.

durch Aufruf einer Methode einer von Metrowerks implementierte Klasse namens Reanimator die Benutzerschnittstelle realisieren.

Das GUI wird von der Klasse SearchController verwaltet. Alle Events, die vom Benutzer durch Interaktion mit dem GUI ausgelöst werden können, werden von einer Instanz des SearchController ausgewertet.

Da die Thematik der GUI-Anbindung, abgesehen von den erwähnten Details, keine Besonderheiten in Hinsicht auf die in dieser Studienarbeit untersuchte Thematik mit sich bringt, verweist der Verfasser den interessierten Leser auf den Quellcode der Klasse SearchController.

4 Tests und Vergleiche der Suchmaschinen

4.1 Test der Agenten-Suchmaschine

Testumfang und Bedingungen

Zur Erhebung von Daten über die Performanz der entwickelten Agenten-Suchmaschine nahm der Verfasser insgesamt drei verschiedene Tests vor. Zum einen wurden die beiden Suchverfahren, sequentiell und parallel getestet, zum anderen wurde der den beiden Suchverfahren zugrunde liegende Suchalgorithmus (des weiteren Sequentiell-Pur) außerhalb der Agenten-Umgebung als eigenständiges Programm getestet. Der letztgenannte Test hatte den Sinn, alle innerhalb des Agenten-Systems Mole vorhandenen spezifischen Mechanismen zu umgehen, um festzustellen, welchen Einfluß diese auf die Performanz der Suchmaschine besitzen. Insbesondere der Zugriff auf Funktionen der Gateway-Agenten (vor allem die Funktionalität des NewsServer-Agenten) über den RPC-Mechanismus konnte im letzten Test umgangen werden.

Sämtliche Tests wurden auf einem dem Verfasser zur Verfügung stehenden Rechner vorgenommen. Dabei handelte es sich um einen PC-kompatiblen Rechner mit Intel Pentium® 90 MHz und 72 MB Hauptspeicher. Als Java-Runtime-Umgebung wurde der Just-In-Time-Compiler der Firma Symantec (s. [SYMANTEC]) verwendet. Die ermittelten Ergebnisse werden in der nachfolgenden Tabelle angegeben und im Anschluß diskutiert. Die den Tests zugrundeliegende Konfiguration der Mole-Engine und der einzelnen Locations, sowie ein verkürztes Protokoll der parallelen Suche wird in Anhang B beschrieben.

Der Suchvorgang wurde mit dem Ausdruck

```
comp\.lang.c++.*
```

als Kriterium für Gruppennamen und

```
( ( ( program or code ) and retrieval ) or help )
```

als Kriterium für Artikel gestartet.

Insgesamt mußten 683 Gruppennamen untersucht werden, von denen sich drei als "passend" erweisen sollten. Innerhalb dieser drei Gruppen galt es insgesamt 576 Artikel zu untersuchen, von denen genau zwölf den Suchkriterien entsprachen.

Testergebnisse und Bewertung

Ergebnisse der Zeitnahmen		
Suchverfahren	Zeit in Millisekunden	Zeit in Minuten
Sequentiell	663.073	11 Minuten 3 Sekunden
Parallel	615.054	10 Minuten 15 Sekunden
Sequentiell-Pur	79.214	1 Minute 19 Sekunden

Am augenfälligsten ist der deutliche Unterschied zwischen den gemessenen Zeiten der Suchverfahren, die innerhalb des Agenten-Systems Mole erhoben wurden, gegenüber dem Sequentiell-Pur genannten reinen Suchalgorithmus, der ohne die Mechanismen des Agenten-Systems arbeitet. Innerhalb des

letztgenannten Tests wurde vom Verfasser außerdem die Zeit gemessen, die zur Auswertung der Suchkriterien benötigt wurde. Dabei wurden 62.300 Millisekunden (62,3 Sekunden) ermittelt.

Der Zeitbedarf des Suchverfahrens resultiert in der Variante Sequentiell-Pur also größtenteils aus der Auswertung der Suchkriterien. Der Zeitaufwand für das Lesen der Informationen von Gruppen und Artikeln aus dem Dateisystem ist dem gegenüber als gering einzuschätzen.

Der in etwa zehnfach höhere Zeitaufwand, der innerhalb der Umgebung des Agentensystems ermittelt wurde, kann im wesentlichen auf den RPC-Mechanismus von Mole zurückgeführt werden. Er ergibt sich aus der Notwendigkeit, den Header jedes Artikels einer "passenden" Gruppe über den Aufruf einer Methode des NewsService-Gateway-Agenten per RPC zu ermitteln. Dies geschieht insgesamt 576 mal.

Der leichte Vorsprung der parallelen Suche gegenüber der sequentiellen wäre durch Start der Locations auf verschiedenen Rechnern weiter ausbaubar gewesen.

Schlußfolgerungen

Die realisierte Agenten-Suchmaschine wertet eine Suchanfrage relativ langsam aus. Die starken Performanzeinbußen innerhalb der Ablaufumgebung des Mobile-Agenten-Systems Mole machen deutlich, daß auf diesem Gebiet noch Verbesserungen notwendig sind. Die Online-Dokumentation erwähnt diesen Sachverhalt²⁸ und gibt Hoffnung, daß mit Umstellung des RPC-Mechanismus auf das RMI-Verfahren von Sun (Remote Method Invovation, s. [RMI]) deutliche Verbesserungen (erwähnt wird Faktor 10) möglich sind.

4.2 Klassische Suchmaschinen

4.2.1 Vorbemerkungen

Wie schon in der Einführung erwähnt, entstand durch das ständig ansteigende Informationsvolumen der Usenet-News der Bedarf nach leistungsfähigen Werkzeugen, die eine Recherche nach den gewünschten Informationen unterstützen. Dies führte schließlich unter anderem zum Erscheinen sogenannter Suchmaschinen.

Die Hardware einer Suchmaschine wird in der Regel durch einen Rechner bzw. einen Verbund von Rechnern repräsentiert, der an das Internet angeschlossen ist. Der Zugriff auf eine Suchmaschine erfolgt durch Web-Browser und ist gewöhnlich für alle an das Internet angeschlossenen Anwender kostenlos. Viele Firmen, die Suchmaschinen bereitstellen, finanzieren sich über Werbung oder sehen ihr Angebot als technische Demonstration der Leistungsfähigkeit ihrer Hard- und Software an (s. [DJN97], [ALTAVISTA97]).

Das erstaunliche Wachstum des Internets in all seinen Ausprägungen (sei es beispielsweise die Zahl angeschlossener Nutzer oder die Menge des verfügbaren Informationsvolumens) hat natürlich nicht halt gemacht vor dem Phänomen der Suchmaschinen. So ist es schwierig, sich allein auf dem Gebiet der hier verfügbaren Dienste einen Überblick zu verschaffen. Monatlich werden neue Dienste angeboten, andere verschwinden. So wird z.B. das Gebiet der News-Recherche des Anbieters Excite (s. [EXCITE97]) nicht mehr direkt von diesem bedient, sondern dieser greift aufgrund einer strategischen Allianz auf die Ressourcen von DejaNews zu (s. [DJN_PR2]).

Der Verfasser hat sich im Verlauf seiner Recherchen dazu entschieden, zwei Suchmaschinen näher zu betrachten. Die Auswahl fiel zum einen auf AltaVista, da sich dieser Dienst durch den geballten Einsatz leistungsfähiger Hardware bemerkbar gemacht hat, zum anderen auf DejaNews, einen Dienst

²⁸ "We currently using our own implementation of an Java RPC which is not satisfying as it relies heavily on code written by the programmer of an agent. ... We're planning to use JavaSofts RMI mechanisms (they will be parts of Java 1.1) instead of our own mechanisms. This will be much (at least factor 10) faster as now. ... We expect to have RMI included in the next version of Mole." (s. [MOLE96])

der sich ausschließlich auf das Gebiet der News-Recherche spezialisiert hat und das wohl umfassendste Angebot besitzt.

4.2.2 AltaVista und DejaNews

Die gewählten Suchmaschinen besitzen eine Reihe von Gemeinsamkeiten und Unterschiede. Beide stehen als Dienste im World-Wide-Web zur Verfügung, beide sind kostenlos, beide bieten ähnliche Funktionen für eine Recherche.

Funktionalität

Grundlage jeder Recherche sind Worte bzw. Wortfragmente, die der Anwender eingeben muß. Aufgrund dieser Wortliste ermitteln die Suchmaschinen eine sogenannte Trefferliste. Treffer sind die Artikel, die alle Worte der Wortliste enthalten. Berücksichtigt werden dabei Worte, die in der Subject-Zeile oder im Body des Artikels auftreten. Durch Gewichtungsverfahren, die z.B. bewerten, wie häufig ein Wort der Wortliste in einem Artikel verwendet wurde oder an welcher Position es im Artikel auftrat (ein Wort in der Subject-Zeile eines Artikels erhält einen höheren Wert, als ein Wort im Body des Artikels), werden die Treffer für die Trefferliste vorsortiert. So werden die Treffer mit der höchsten Gewichtung auch als erste in der Liste präsentiert. Beide Suchmaschinen liefern die Trefferliste nicht komplett zurück, sondern nur einen Ausschnitt derselben, meist bis zu zehn Treffer. Ferner stellen sie Funktionen bereit, um folgende Ausschnitte der Trefferliste zu erhalten.

Die Suchmaschinen erlauben es, bei der Angabe von Worten Jokerzeichen einzusetzen und so eine Menge äquivalenter Worte zu beschreiben. Über die Grundfunktionalität hinaus bieten die Suchmaschinen die Möglichkeit komplexere Abfragen zu formulieren. Dies kann durch logische Ausdrücke in einer Form, wie sie schon in Abschnitt 2 "Entwurf" dieser Arbeit beschrieben wurden, geschehen. Weiterhin gibt es verschiedene andere Möglichkeiten, Suchanfragen zu formulieren (z.B. die Angabe, daß ein Artikel schon als Treffer gilt, wenn er ein Wort der Wortliste enthält), die sich aber letztendlich immer auch durch die erwähnten logischen Ausdrücke repräsentieren lassen (voriges Beispiel durch die Verknüpfung aller Worte einer Wortliste durch einen Oder-Operator).

DejaNews bietet gegenüber AltaVista einige weitergehende Funktionen (Erstellen von Filtern, die eine Vorauswahl von Artikeln ermöglichen, in denen dann gesucht werden soll), auf diese soll hier jedoch nicht weiter eingegangen werden, da sie im wesentlichen nur dem Komfort des Anwenders dienen, die Funktionalität der Suchmaschine aber nicht in signifikanter Weise erweitern.

Vergleich von Hard- und Software

Die Realisierung der beiden Suchmaschinen unterscheidet sich dabei sowohl was die eingesetzte Software betrifft, als auch die Hardware, mit der die Suchmaschinen betrieben werden. Über die eingesetzte Software kann nur wenig gesagt werden; es handelt sich in beiden Fällen um Eigenentwicklungen, die als Geschäftsgeheimnis behandelt werden. Beide können allgemein der Gattung der Volltext-Retrieval-Programme zugeordnet werden. Digital Equipment bietet Lösungen auf diesem Gebiet an, die auf der AltaVista-Technologie basieren ([ALTASOFT97]).

Über die Hardware sind dagegen einige Informationen auf den WWW-Seiten der jeweiligen Dienstleister verfügbar (s. [ALTAVISTA97], [DJN_ABOUT]). Die folgende Tabelle stellt einen Überblick dar.

Hardwareausstattung der Suchmaschinen	
AltaVista	DejaNews
<p>Web-Server: AlphaStation 500 256 MB Hauptspeicher, 6GB Festplatte</p> <p>Auf dieser Maschine läuft ein Web-Server, der die Anfragen an den News-Indexer bzw. den Web-Indexer verteilt.</p> <p>News-Indexer: AlphaServer 600 5/333, 896MB Hauptspeicher, 13 GB Festplatte .</p> <p>Diese Maschine hält einen Index des aktuellen News-Spools</p> <p>Newsserver: AlphaServer 600 5/333, 896MB Hauptspeicher, 24 GB RAID Festplatte</p> <p>Dient als normaler Newsserver, auf dem sich der aktuellen News-Spool befindet.</p>	<p>Web-Server: Dual Pentium 133 Rechner mit bis zu 256MB Hauptspeicher, mehrere 1-4GByte Festplatten.</p> <p>Datenbank Server: Dual Pentium 133 Rechner mit bis zu 256MB Hauptspeicher, mehrere 1-4GByte Festplatten.</p>

AltaVista setzt modernste Mainframe-Technologie ein, DejaNews dagegen verwendet leistungsstarke PC-Technologie, die heute (1997) allerdings nicht mehr dem aktuellen Stand der Technik entspricht. Dennoch liegen Größenordnungen zwischen der reinen Rechenleistung, die AltaVista und DejaNews zur Verfügung stehen.

Menge und Aktualität der verfügbaren Daten

Vergleicht man die verfügbare Menge an Gruppen und Artikeln stellt man überraschenderweise fest, daß DejaNews hier über eine weitaus größere Datenbasis verfügt (s. [ALTAVISTA97], [DJN_ABOUT], [DJN_PR1], [DJN_PR2]).

Recherchierbare Datenbasis		
	AltaVista	DejaNews
Gruppen	14.000	15.000
Artikel	4 Millionen	100 Millionen
Zeitraum	k.a	22 Monate

Der Grund hierfür ist das erklärte Ziel der DejaNews-Betreiber, alle Artikel, die seit Entstehen der Usenet-News veröffentlicht wurden zu archivieren (s. [DJN_NET]). AltaVista legt dagegen den Schwerpunkt seines Angebots auf die Recherche von WWW-Seiten.

Die Bewertung der Aktualität, der von einem Newsserver bereitgestellten Daten muß den Verteilungsmechanismus von Artikeln zwischen verschiedenen Newsservern berücksichtigen. Wie in Kapitel 1.1 "Usenet-News" erläutert, wird ein Artikel nach der Erstellung vom Anwender an einen Newsserver übertragen. Dieser sendet den Artikel dann an benachbarte Newsserver, falls diese sich

für die Gruppe, in welcher der Artikel veröffentlicht wurde "interessieren". Die Zeit, die zwischen dem Veröffentlichen eines Artikels bis zum Eintreffen beim Newsserver der Suchmaschine vergeht, darf in die Bewertung der Aktualität der bereitgestellten Daten nicht eingehen, da die Suchmaschine darauf keinen Einfluß nehmen kann. Entscheidend ist vielmehr die Zeit, die zwischen dem Eintreffen eines Artikels beim Newsserver der Suchmaschine und der Verfügbarkeit des Artikels für eine Suchabfrage vergeht (Latenzzeit).

Um eine solche Messung vornehmen zu können, müßte die Möglichkeit bestehen, den Zeitpunkt des Eintreffens eines Artikels beim Newsserver der Suchmaschine zu ermitteln. Leider gibt es dazu keinen direkten Weg. Als einzige Möglichkeit stellt sich das direkte Senden eines Artikels an den Newsserver einer Suchmaschine zur Veröffentlichung dar. Dies ist jedoch leider nur bei DejaNews möglich.

Zum Veröffentlichen von Artikel mit Hilfe des DejaNews eigenen Newsservers wurde sogar ein eigenes WWW-basiertes Interface geschaffen. Durch Ausfüllen eines Formulars in einem Web-Browser wurden alle dazu notwendigen Angaben gemacht (z.B. die Gruppe, in welcher der Artikel erscheinen sollte, die Subject-Zeile und der eigentliche Text). Anschließend wurde das Formular an DejaNews abgesendet. Als Bestätigung erschien eine Meldung, die besagte, daß der Artikel innerhalb einer halben Stunde auf dem DejaNews eigenen Newsserver und innerhalb von 4 bis 8 Stunden in der Datenbank erscheinen würde²⁹. Der Artikel wurde in der Gruppe "alt.test" mit dem Subject "Mikroenzephalitische Autointoxikative Demenz" veröffentlicht. Die Wahl des ungewöhnlichen Subjects stellte dessen Eindeutigkeit sicher. Nach absenden des Artikel wurde in halbstündigem Abstand vom Autor eine Anfrage an DejaNews nach diesem Artikel gerichtet. Nach ca. Dreieinhalb Stunden wurde der Artikel zum ersten Mal gefunden. Der Test bestätigt also die von DejaNews gemachten Angaben.

Im Falle AltaVistas kann nur eine Schätzung aufgrund der hierzu veröffentlichten Daten vorgenommen werden. So wird behauptet, daß der WWW-Indexer von AltaVista ca. 1 GB pro Stunde indizieren kann. Ähnliche Leistungsdaten für die Indizierung der News-Daten vorausgesetzt, kann davon ausgegangen werden, daß ein Artikel nach Eintreffen beim Newsserver innerhalb von maximal einer Stunde der Recherche zur Verfügung steht (das gesamte Datenaufkommen in Usenet-News beträgt momentan noch etwas weniger als 1GB pro Tag; s. [CALC97]).

4.3 Performanz der klassischen Suchmaschinen

4.3.1 Kriterien

Ein Vergleich der Performanz von Suchmaschinen gestaltet sich aus mehreren Gründen als außerordentlich schwieriges Unterfangen. So besteht, wie zuvor ausführlich dargestellt, die Software der Suchmaschinen aus Eigenentwicklungen der jeweiligen Anbieter, die auf jeweils völlig unterschiedlicher Hardware abläuft. Es bleibt daher nur die Möglichkeit, die Suchmaschinen als Ganzes, zu betrachten, als eine "Black-Box", an die Anfragen gestellt werden. Diese werden intern auf unbekannte Weise verarbeitet und schließlich wird ein Ergebnis zurückgeliefert. Konkrete Aussagen über die Güte der verwendeten Algorithmen sind daher nicht möglich. Trotzdem soll in einem folgenden Kapitel der Versuch gemacht werden, einen groben Algorithmus zu skizzieren und zu bewerten, der Grundlage eines von einer Suchmaschine implementierten Verfahrens sein könnte.

Um vergleichbare Bedingungen zu erhalten, sollen die zu testenden Suchmaschinen gleiche Anfragen (jeweils in der Syntax der betreffenden Suchmaschine) über den gleichen Datenbestand (gleicher

²⁹ "Your Article Has Been Spooled For Posting! Your article has been put in our outgoing spool. It should be posted to our news server within 0.5 hours, and should appear in our database in 4-8 hours."

Zeitraum und gleiche Menge an Gruppen) bearbeiten. Dies kann durch entsprechende Formulierung der Suchanfragen an die Suchmaschinen gewährleistet werden.

Aufgrund der Anbindung der Suchmaschinen an das World-Wide-Web ist die Zeitnahme gewissen Einschränkungen unterworfen. So läßt es sich nicht vermeiden, daß die Zeit für das Übertragen von Anfrage und Ergebnis in die Messung eingeht. Diese Zeiten können, je nach der momentanen Belastung des Netzes, stark variieren. Weiterhin kann keine Aussage über die Last gemacht werden, der eine Suchmaschine ausgesetzt ist (diese arbeiten in der Regel parallel eine Reihe von Anfragen gleichzeitig ab). Der Verfasser wird daher keine exakten Zeitnahmen, sondern nur Größenordnungen angeben können.

4.3.2 Zeitnahme AltaVista - DejaNews

Um einen ungefähren Eindruck zu gewinnen, wie schnell AltaVista und DejaNews eine Suchanfrage auswerten, wurde vom Verfasser eine einfache Suchanfrage formuliert und dann für jede Suchmaschine ausgewertet.

Die Suchanfrage sollte Artikel der "comp.sys" Hierarchie, die im Zeitraum zwischen dem 1. und 7. April 1997 erschienen sind, berücksichtigen. Der Test wurde am 14. April 1997 vorgenommen, zu einem Zeitpunkt also, an dem die in dem angegebenen Zeitraum veröffentlichten Artikel auf beiden Newsservern angekommen sein mußten. Es wurden alle Artikel gesucht, die in der Subject-Zeile die Worte "program" oder "code" und zusätzlich das Wort "retrieval" enthielten. Des weiteren sollten alle Artikel geliefert werden, die das Wort "help" enthielten.

Die eigentliche Suchanfrage hatte daher folgende Gestalt:

```
((program or code)and retrieval)or help)
```

Die weitere Eingrenzung auf den vorgegebenen Zeitraum und die gewünschte Menge von Gruppen wurde jeweils durch Ausfüllen spezieller Formulare der Suchmaschinen bewerkstelligt.

Beide Suchmaschinen lieferten innerhalb von weniger als 30 Sekunden³⁰ die ersten Ergebnisse als Trefferliste. Dabei meldete AltaVista insgesamt ca. 2000, DejaNews exakt 2309 Dokumente als Treffer.

Um die ermittelten Zeiten abzusichern, wurde jede Suchanfrage zehnmal ausgeführt. Die ermittelten 30 Sekunden als Obergrenze konnten dabei aufrecht erhalten werden.

4.3.3 Zusammenfassung der Ergebnisse

Es läßt sich feststellen, daß die klassischen Suchmaschinen eine sehr hohe Performanz bei der Auswertung von Suchanfragen haben. Die verwendeten Indizierungsverfahren bedingen jedoch eine gewisse Latenzzeit zwischen einer und acht Stunden, bis Artikel einer Suche zur Verfügung stehen.

³⁰ Die Anfrage wurde vom Verfasser am frühen Morgen (MEZ) vorgenommen, ein Zeitraum mit relativ geringer Netzbelastung, da in den Vereinigten Staaten noch Nacht ist.

4.4 Theoretische Betrachtungen

4.4.1 Algorithmus der Agenten-Suchmaschine

Der Zeitaufwand des im Rahmen dieser Studienarbeit implementierten Suchverfahrens, läßt sich im Falle der sequentiellen Suche an einer `Location` wie folgt abschätzen:

Zum Ermitteln der passenden Gruppen muß die Liste der an der `Location` verfügbaren Artikel komplett analysiert werden. Sei n die Anzahl der verfügbaren Gruppen und m die Länge des längsten Gruppennamens. Dann kann der Suchaufwand nach oben mit $O(n*m)$ abgeschätzt werden, da die Auswertung des Gruppennamens qualifizierenden Ausdrucks durch einen DFA (ein Lexmat stellt nichts anderes dar) realisiert wurde.

Für jede ermittelte "passende" Gruppe müssen dann die Subject-Zeilen der in dieser Gruppe enthaltenen Artikel betrachtet werden. Bezeichne in diesem Falle n die Anzahl der Artikel einer Gruppe und m die Länge der längsten Subject-Zeile. Damit kann die Auswertung nach "passenden" Artikeln ebenfalls mit $O(n*m)$ abgeschätzt werden. Die Ermittlung der enthaltenen Teilfolgen erfolgt durch einen DFA, besitzt also pro Artikel einen Aufwand von $O(m)$. Die Auswertung der logischen Verknüpfungen des Ausdrucks steigt zwar mit der Komplexität des Ausdrucks, geht jedoch in die Abschätzung des Aufwands bei der Auswertung nach "passenden" Artikeln nur mit einem konstanten Wert ein, der in der O -Notation nicht berücksichtigt wird.

4.4.2 Ein möglicher Suchmaschinen-Algorithmus

Im folgenden soll ein Verfahren skizziert werden, mit dem eine besonders performante Auswertung von Suchausdrücken, die auf Jokerzeichen basieren, möglich ist. Nach Ansicht des Verfasser könnte dieses Verfahren, in abgewandelter Form, den Algorithmen der zuvor besprochenen klassischen Suchmaschinen zugrunde liegen.

Das skizzierte Suchverfahren baut auf der Vorverarbeitung aller Artikel, die einer Recherche zur Verfügung stehen sollen, auf. Die bei dieser Vorverarbeitung gewonnenen Informationen können dann zu einer besonders schnellen Auswertung der Suchausdrücke herangezogen werden.

Datenstruktur

Die Vorverarbeitung findet im wesentlichen durch eine sog. Indizierung der im System vorhandenen Artikel statt. Dazu wird eine Liste der in den Artikeln vorkommenden Worte (sowohl in der Subject-Zeile als auch im Body) geführt. Jeder Listeneintrag zu einem Wort verweist auf eine Liste mit Referenzen auf alle Artikel, die dieses Wort enthalten. Diese Liste wird beim Eintreffen neuer Artikel ständig aktualisiert. Die Namen der Gruppen werden dabei ebenfalls als Worteinträge behandelt. Dadurch wird für jeden Gruppennamen eine Liste der zugehörigen Artikel geführt. Für eine schnelle Auswertung von Suchausdrücken ist es erforderlich, die Liste der Worte aufsteigend sortiert zu halten. Trifft ein neuer Artikel ein, so wird er in Worte zerlegt. Diese Worte werden dann sortiert in die Liste eingefügt. Zu jedem Wort wird die unter diesem Eintrag geführte Liste von Artikelreferenzen um eine Referenz auf den neuen Artikel erweitert.

Die durch diese Vorverarbeitung realisierte Datenstruktur soll durch die folgende Grafik veranschaulicht werden.

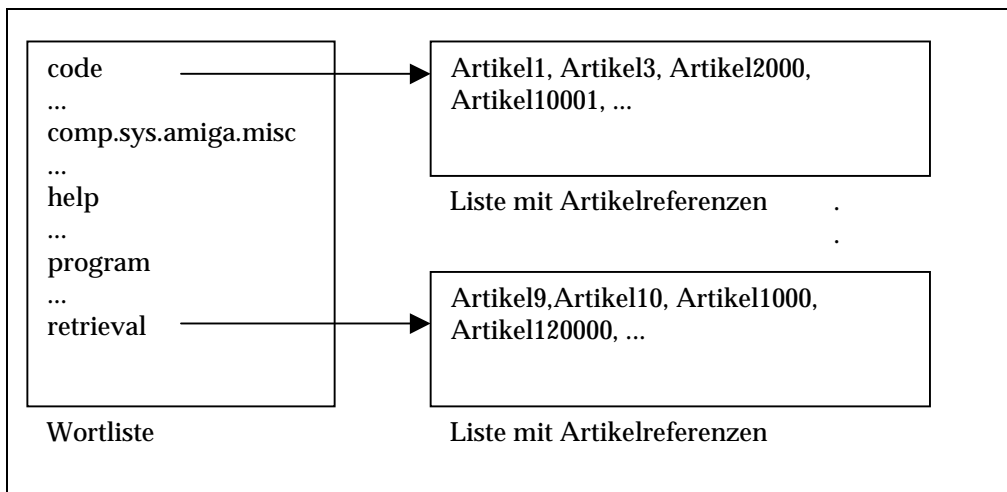


Abb. 8: Datenstruktur einer index-basierten Suchmaschine

Auswertung einer Suchanfrage

Die Auswertung einer Suchanfrage stellt sich dann als Mengenoperation dar. Jeder Suchausdruck besteht aus der Beschreibung von Worten, über deren Enthaltensein in einem Artikel er einen logischen Zusammenhang formuliert.

Zu jedem Wort kann, dank der Vorverarbeitung, die Menge der dieses Wort enthaltenden Artikel ermittelt werden (bei sortierter Liste ein Aufwand von $O(\log(n))$ mit n als Anzahl der in die Liste eingetragenen Worte). Die im Suchausdruck gemachte logische Verknüpfung wird dann als Operation auf die ermittelten Mengen realisiert. Der Suchausdruck

(program or code)

liefert als Ergebnis also die Vereinigungsmenge der Menge aller Artikel die das Wort "program" enthalten mit der Menge der Artikel die das Wort "code" enthalten. Da die Suchmaschinen als Trefferliste immer nur einen begrenzten Ausschnitt liefern, läßt sich diese Operation besonders effizient realisieren. Im Falle der Vereinigungsmenge werden z.B. einfach die ersten n Einträge der ersten ermittelten Wortliste geliefert.

Schlußfolgerung

Das vom Verfasser vorgeschlagenen Verfahren macht deutlich, wie die klassischen Suchmaschinen die hohe Performanz beim Auswerten von Suchanfragen erzielen könnten³¹.

³¹ Es ist nach Ansicht des Verfassers sehr wahrscheinlich, daß die besprochenen Suchmaschinen ihre Verfahren auf eine ähnliche Weise implementiert haben. Zentrale Fragen, wie z.B. das immens hohe zu indizierende Datenvolumen behandelt werden kann, wurden dabei nicht angesprochen. Auch die Problematik des sortierten Einfügens von Worten in eine außerordentlich große Indexliste wurde nicht weiter angesprochen. Die effiziente Implementierung von Verfahren zur Lösung dieser Probleme stellt, nach Ansicht des Verfassers, das eigentliche Geschäftsgeheimnis der Entwickler der genannten Suchmaschinen dar.

4.4 Klassische Suchmaschinen vs. parallel suchende Agenten

Die klassischen Suchmaschinen sind der vom Verfasser implementierten Suchmaschine bei der Suche in großen Datenmengen in der Performanz deutlich überlegen. Dies zeigen sowohl die zu Beginn dieses Abschnitts an der Agenten-Suchmaschine vorgenommenen Messungen, als auch die in weiteren Kapiteln mit den klassischen Suchmaschinen durchgeführten Messungen. Die theoretischen Überlegungen zum Zeitbedarf der Suchverfahren untermauern diese Ergebnisse.

Dadurch, daß die Agenten-Suchmaschine die zu untersuchenden Daten nicht vorverarbeiten muß, werden die in diesem Abschnitt bei den klassischen Suchmaschinen festgestellten Latenzzeiten vermieden, was der Aktualität der recherchierbaren Datenmenge zugute kommt.

Insgesamt kann festgestellt werden, daß das vom Verfasser implementierte Suchverfahren nur für relativ kleine Datenmengen geeignet ist, wie sie typischerweise in lokalen Newsgruppen eines Newsservers anfallen. Die Vorteile einer Suchmaschine auf Basis Mobiler Agenten, die eingangs genannt wurden, insbesondere eine sehr gute Unterstützung einer heterogenen Struktur von Newsservern als Datenquelle bleiben von den in diesem Abschnitt durchgeführten Betrachtungen unberührt. Eine Ergänzung der vom Verfasser implementierten Suchmaschine um Komponenten, die auch eine indexbasierte Suche ermöglichen, erscheint denkbar. Im folgenden Abschnitt 5 "Zusammenfassung und Ausblick" wird vom Verfasser jedoch ein Ansatz vorgeschlagen, der den Aufwand einer solchen Neuimplementierung umgeht. Dies erscheint insbesondere unter dem Aspekt sinnvoll, daß es schon bisher eine Reihe von Implementierungen solcher Verfahren gibt (s. z.B. [SWISH]).

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung und Bewertung

Die Infrastruktur eines Mobile-Agenten-Systems hat sich zur Implementierung einer Suchmaschine als gut geeignet erwiesen. Dabei stellten sich vor allem die Möglichkeiten zur einfachen Verteilung von Algorithmen (im Falle der parallelen Suche) und die unproblematische Unterstützung verteilter Datenquellen als vorteilhaft heraus (dies soll im Verlauf noch detaillierter dargelegt werden).

Der Verfasser konnte bei der Implementierung gegenüber herkömmlichen Ansätzen³² keine Nachteile entdecken. Es läßt sich vielmehr feststellen, daß im Rahmen eines Mobile-Agenten-Systems (insbesondere des verwendeten Prototyps Mole), auch alle herkömmlichen Ansätze zur Realisierung einer Suchmaschine geboten sind³³. In bezug auf die Problemstellung dieser Studienarbeit stellt das Paradigma der Mobilen Agenten daher eine wirkliche Erweiterung der bisher vorhandenen Entwicklungsmöglichkeiten dar.

Vor- und Nachteile

Die vom Verfasser implementierte Suchmaschine ist in der vorliegenden Version mit Vor- und Nachteilen behaftet. Gravierendster Nachteil des verwendeten Suchverfahrens sind die im Vergleich zu den klassischen Suchmaschinen sehr hohen Laufzeiten, die für eine Suche benötigt werden.

Andererseits ergeben sich aus dem gewählten Suchverfahren mehrere Vorteile, von denen die wichtigsten hier genannt werden. So werden keine Systemressourcen für die Speicherung von Indexdateien benötigt und die zum Erstellen solcher Indexdateien notwendige massive Vorverarbeitung des Datenbestandes ist ebenfalls nicht erforderlich. Der Verzicht auf die Vorverarbeitung bedingt ebenfalls eine höhere Aktualität der Suchergebnisse. So steht ein auf dem Newsserver eingetroffener Artikel direkt einer Suche zur Verfügung. Darüber hinaus können Suchanfragen durch Verwendung regulärer Ausdrücke mächtiger formuliert werden, als mit den Möglichkeiten, welche die klassischen Suchmaschinen durch Jokerzeichen bieten.

Mobile-Agenten-Systeme

Die in der Einführung genannten Gründe, die den Verfasser dazu bewogen haben, eine Suchmaschine auf Basis eines Mobile-Agenten-Systems zu realisieren, sollen hier noch einmal betont werden.

Die prognostizierte jährliche Verdoppelung des Datenaufkommens in Usenet-News (s. [CALC97]) hat schon in der Vergangenheit dazu geführt, daß die meisten Newsserver nicht mehr alle weltweit verfügbaren Gruppen führen. Nach Ansicht des Verfassers wird sich dieser Trend in naher Zukunft weiter verstärken. Dieser Entwicklung wird langfristig wohl nur durch das Einrichten verschiedener, spezialisierter Newsserver begegnet werden³⁴ können. Dies führt zu einer Verstärkung, der schon heute (u.a. auch durch die lokalen Gruppen) vorhandenen heterogenen Gesamtstruktur der als Datenquellen bereitstehenden Newsserver.

Mobile-Agenten-Systeme stellen insbesondere durch die Konzepte der Orte und migrationsfähigen Agenten eine tragfähige Infrastruktur bereit, um Suchmaschinen zu entwickeln, die auf einfache Weise mit dieser heterogenen Struktur umgehen können.

³² Darunter wäre die Realisierung eines Programms zu verstehen, das direkt auf dem Rechner, auf dem die Daten verfügbar sind, installiert wird.

³³ Insbesondere wäre eine Implementierung der von den sog. "klassischen" Suchmaschinen realisierten Verfahren auf Basis eines Mobile-Agenten-Systems vorstellbar.

³⁴ Z.B. Newsserver, die nur den "unterhaltsamen" Teil der Usenet-News (die "alt"- und "rec"-Hierarchien) oder ausschließlich technische Informationen ("comp"- oder "sci"-Hierarchie) führen.

Zentraler Bestandteil solcher Systeme muß nach Meinung des Autors dabei eine Trading-Komponente sein, die Informationen über verfügbare Newsserver als Datenquellen bereitstellt. Durch diese ließe sich auch eine sehr dynamische Gesamtstruktur bewältigen, in der Newsserver jederzeit in das Gesamtsystem eingebunden bzw. aus diesem herausgenommen werden können. Da dieser Aspekt in dem Prototypen des Agentensystems Mole noch nicht berücksichtigt wurde, erscheinen dem Verfasser weitere Forschungen auf diesem Gebiet als lohnend.

Erweiterungsmöglichkeiten

Um die Nachteile der vorgestellten Implementierung zu überwinden schlägt der Verfasser eine "Meta-Suchmaschine" auf Agenten-Basis vor, die eine Mischarchitektur realisiert. Diese "Meta-Suchmaschine" soll die Vorteile der gegenwärtigen Implementierung mit den Vorteilen der klassischen Suchmaschinen vereinen.

Für Gruppen mit geringem Volumen, die selten abgefragt werden und lokale Gruppen (für die diese Annahme in der Regel ebenfalls zutrifft) soll die "Meta-Suchmaschine" die vorhandene Implementierung verwenden. Eine Suche in diesen Gruppen wird dadurch ressourcen-schonend gewährleistet und garantiert eine hohe Aktualität.

Andernfalls soll auf die Möglichkeiten der klassischen Suchmaschinen zugegriffen werden, um den Vorteil der schnellen Auswertung von Suchausdrücken auszunutzen. Dies könnte durch Entwurf von Gateway-Agenten realisiert werden, die Suchanfragen an die erwähnten klassischen Suchmaschinen abschicken und deren Ergebnisse auswerten. Denkbar wäre auch, eine Anbindung an eine frei verfügbare Software zu realisieren, die eine indexbasierte Volltext-Recherche ermöglicht (z.B. [SWISH]). Als letzte Alternative wäre die Neuimplementierung eines solchen Verfahrens zu nennen.

Unabhängig vom gewählten Ansatz muß dafür gesorgt werden, daß Suchanfragen einheitlich formuliert werden. Insbesondere darf ihre Mächtigkeit die Fähigkeiten einer an das System angebotenen Suchmaschine nicht überschreiten. Da die bisher betrachteten klassischen Suchmaschinen zum überwiegenden Teil identisch formulierte Suchanfragen verarbeiten können, gestaltet sich diese Forderung jedoch als unproblematisch. In bezug auf die im Rahmen dieser Studienarbeit implementierte Suchmaschine müßte jedoch die Verwendung von regulären Ausdrücken verboten und ein Mechanismus geschaffen werden, der Suchmuster mit Jokerzeichen unterstützt (was durch Umformen in einen regulären Ausdruck geleistet werden könnte).

5.2 Ausblick

Beschränkte Konzepte

Nach ausführlicher Auseinandersetzung mit der Thematik kommt der Verfasser zu dem Schluß, daß Suchmaschinen, egal ob in der als klassisch bezeichneten Variante oder wie in dieser Studienarbeit basierend auf einem Mobile-Agenten-System implementiert, nur ein beschränktes Hilfsmittel darstellen, um das Problem der allgemeinen Informationsüberflutung zu bewältigen. Das im vorigen Kapitel vorgestellte Konzept einer Meta-Suchmaschine stellt, nach Ansicht des Verfassers, zwar einen guten Ansatz zur Recherche in verteilten Datenquellen dar, der in Zukunft sicher noch an Bedeutung gewinnen wird, letztendlich stellt sich jedoch das Konzept der Suchmaschinen als zu beschränkt heraus.

Aus eigenen Erfahrungen im Umgang mit Suchmaschinen bei der Recherche nach Informationen für diese Studienarbeit gewann der Autor die Erkenntnis, daß Suchmaschinen sicherlich eine gute Unterstützung beim Auffinden von Informationen darstellen, das Problem jedoch nicht auf zufriedenstellende Weise lösen können.

Die typische Arbeitsweise im Umgang mit einer Suchmaschine stellt sich als ein relativ zeitaufwendiger, mehrstufiger Prozeß dar. In der Regel wird, selbst nach Formulierung einer komplexeren Suchanfrage, von den Suchmaschinen noch eine Fülle von irrelevanten Dokumenten angegeben. Erst durch Betrachten einiger dieser Dokumente und daraus resultierende Verfeinerung der Abfrage kann die Ergebnismenge auf die tatsächlich relevanten Dokumente eingegrenzt werden.

Nimmt man als Maß für die Güte einer Recherche die Menge der gelieferten Treffer und setzt diese in Verhältnis zu den tatsächlich relevanten Dokumenten (precision³⁵), so ergibt sich häufig nach der ersten Suchanfrage ein relativ ungünstiges Verhältnis.

Die Notwendigkeit der Interaktion eines Anwenders mit der Suchmaschine resultiert dabei aus der Tatsache, daß die bisher gegebenen Möglichkeiten der Formulierung von Suchanfragen dem Problem nicht gerecht werden. Menschen sind gewohnt in Konzepten zu denken und einzuordnen. Suchmaschinen arbeiten dagegen mit Aussagen über das Enthaltensein von Worten (oder Zeichenfolgen) in Dokumenten. Es ist Aufgabe des Anwenders, das Konzept, dem die von ihm gesuchten Dokumente zugeordnet werden können, in einen entsprechenden Suchausdruck, der eine Aussage über das Enthaltensein von Teilworten darstellt, umzuformulieren, damit er die gewünschten Dokumente von einer Suchmaschine geliefert bekommt.

Diese Umformung führt zwangsweise zu Einschränkungen, da eine Abfrage, die sich eigentlich auf die Semantik eines Dokumentes bezieht, in eine Abfrage über rein syntaktische Zusammenhänge überführt wurde.

Intelligente Agenten

Tatsächlich gibt es eine Vielzahl von Anstrengungen, die Einschränkungen von Suchverfahren, wie sie im Falle der Suchmaschinen realisiert wurden, zu überwinden.

Ein Großteil der aktuellen Forschung³⁶ diesbezüglich wird dabei auf dem Gebiet sog. Intelligenter Agenten vorgenommen. Darunter ist nach [HOHL95] ein "Oberbegriff für alle Agentensysteme, die mit Mitteln der künstlichen Intelligenz arbeiten" (s.S. 3) zu verstehen. Eines der Einsatzgebiete, für die diese Intelligenzen Agenten vorgesehen werden, ist die Recherche nach Informationen ("information retrieval").

Nach Ansicht des Verfassers ist dabei die Realisierung einer Infrastruktur für Intelligente Agenten auf den Grundlagen eines Mobilen-Agenten-Systems besonders vorteilhaft. Die Eignung eines Mobilen-Agenten-Systems für verteilte Datenquellen wurde schon mehrfach betont. Die Fähigkeit von Mobilen Agenten, unabhängig vom Anwender an anderen Orten zu arbeiten, prädestinieren sie jedoch geradezu für ein Recherche-Verfahren, das Intelligente-Agenten-Technologie verwendet. So ist es für einen Anwender sicherlich bequemer, eine Anfrage für den Intelligenzen Agenten zu formulieren und diesen dann unabhängig vom Anwender recherchieren zu lassen, statt interaktiv mit einer Suchmaschine die gewünschten Informationen aufzuspüren.

Dabei ist es durchaus kein Nachteil, wenn der Agent eine gewisse Zeit für eine Recherche benötigt (einige Stunden, vielleicht sogar mehrere Tage). Denn in vielen Fällen ist es für den Anwender günstiger auf das Ergebnis einer automatischen agenten-basierten Recherche zu warten, wenn dafür eine so hohe Qualität zu erwarten ist, daß kaum Nacharbeit nötig wird, statt selbst eine Suchmaschine zu

³⁵ [JANSEN97] erwähnt in diesem Zusammenhang zwei Maße, die zum Leistungsvergleich von Suchmaschinen (allgemeiner von Retrieval-Systemen) verwendet werden können:

- Recall = Anzahl erhaltener relevanter Dokumente/Gesamtzahl relevanter Dokumente
- Precision = Anzahl erhaltener relevanter Dokumente/Gesamtzahl der erhaltenen Dokumente

³⁶ Sowohl [JANSEN97] als auch [IIR95] versuchen einen kleinen Überblick über die Thematik und aktuelle Forschungsprojekte zu verschaffen. Dem interessierten Leser seien diese Dokumente als Ausgangspunkt weiterer, eigenständiger Recherchen empfohlen.

bedienen, die zwar schnelle Ergebnisse liefert, aber meist auch sehr viele aufwendige Abfragen erfordert, um aus der Flut von Treffern die richtigen herauszufinden. Schließlich kann der Anwender diese Wartezeit für andere Tätigkeiten nutzen.

Weitere Forschungen

Nach Meinung des Verfassers sollte die auf einem Mobilen-Agenten-System basierende Entwicklung Intelligenter Agenten zur Lösung des Problems der Recherche nach Informationen in Zukunft verstärkt betrachtet werden.

Sobald die bisher entworfenen Agentensysteme das Prototypen-Stadium überschritten haben und erste Implementierungen im kommerziellen Bereich eingesetzt werden, werden nach Ansicht des Verfassers einige weitere Aspekte an Bedeutung gewinnen, die im Rahmen dieser Studienarbeit nicht betrachtet wurden. Dazu gehören vor allem die Themen Trading und Brokering. Die Vermittlung von Diensten und insbesondere die Bezahlung der in Anspruch genommenen Dienste ist für kommerzielle Anbieter, aber auch für deren Nutzer von entscheidender Bedeutung. So wird es in Zukunft nicht nur darauf ankommen, herauszufinden "wo" Informationen zur Verfügung stehen, sondern die Kosten, diese Information zu erhalten werden immer mehr in den Mittelpunkt rücken.

Aus Sicht des Anwenders sind dann Optimierungsalgorithmen gefragt, die nicht nur qualitativ hochwertige Informationen ermitteln, sondern diese auch möglichst kostengünstig beschaffen.

5.3 Fazit

Die in der Einführung erwähnte, ständig anwachsende Informationsflut hat sich als kennzeichnendes Merkmal des oft beschworenen Wandels unserer Gesellschaft hin zur sog. Informationsgesellschaft erwiesen. In Zukunft wird es dabei immer entscheidender, nicht über Information zu verfügen, sondern diese gezielt auffinden zu können. Vom Anwender unterstützt, können die in dieser Arbeit dargestellten Suchmaschinen dabei einen kleinen Beitrag leisten. Der vom Verfasser skizzierte Ansatz einer "Meta-Suchmaschine", welche die Vorteile der klassischen Suchmaschinen mit der Eignung der Mobilen-Agenten-Systeme für heterogene, dynamische, verteilte Systeme vereint, erscheint dabei in Hinblick auf eine Weiterentwicklung dieser Technologie als interessanter Ausgangspunkt für weitere Forschungen.

Insgesamt ist, nach Ansicht des Verfassers, jedoch der Ansatz der Suchmaschinen auf seine Eignung kritisch zu überprüfen. Die im Ausblick erwähnte Technologie der Intelligenen Agenten bietet tiefere Ansätze zur Lösung des Problems und wird in naher Zukunft sicher noch an Bedeutung gewinnen. Mobile-Agenten-Systeme könnten dabei eine gute Basis darstellen, auf der Intelligente Agenten aufbauen können.

Weitergehende Forschungen auf den erwähnten Gebieten wären daher wichtige Schritte, um der ständig anwachsenden Informationsflut zu begegnen und den Anwendern dementsprechend selektierte Informationen unter anwenderfreundlichen Bedingungen zu liefern.

Anhang

A Eine erweiterte BNF-Notation

A.1 Vorbetrachtungen

Im folgenden soll eine Notation vorgestellt werden, mit der eine Grammatik angegeben werden kann, die eine bestimmte Sprache definiert. Die Regeln einer Grammatik werden Produktionen genannt. Eine Produktion enthält syntaktische Variablen und sog. Terminalsymbole. Jede Produktion besitzt folgende Grundgestalt:

`<Linke Seite> = <Rechte Seite>`

Dabei sind `<Linke Seite>` und `<Rechte Seite>` Platzhalter für eine beliebige Kombination von Variablen und Terminalsymbolen. Eine solche Kombination wird auch Wort genannt.

Eine Produktion stellt eine Substitutionsregel dar, die auf Worte angewendet werden kann. Eine Produktion kann auf ein Wort angewendet werden, falls es das mit `<Linke Seite>` bezeichnete Teilwort der Produktion enthält. Dieses Teilwort kann dann durch das Wort `<Rechte Seite>` ersetzt werden.

Jede Grammatik besitzt eine ausgezeichnete Startvariable. Durch Anwendung der Produktionen der Grammatik können von dieser Startvariable ausgehend Worte gebildet werden.

Die von der Grammatik definierte Sprache ist die Menge aller Worte, die nur noch aus Terminalsymbolen bestehen und die ausgehend von der Startvariablen, durch beliebig häufige Anwendung der Produktionen gebildet wurden.

A.2 Die Notation

Die hier beschriebene erweiterte BNF-Notation folgt weitestgehend der Darstellung in [RFC822] (s.S. 2f). In dieser Notation werden syntaktische Variablen einfach durch ihren Namen dargestellt. Die Zeichen "`<`" und "`>`" werden nur verwendet, wenn dies die Kenntlichmachung einer syntaktischen Variablen erleichtert. Diese wird dann von den erwähnten Zeichen umschlossen. Terminalsymbole werden von Anführungszeichen `""` umschlossen.

Die Produktionen in der erweiterten BNF-Notation besitzen die schon erwähnte Grundgestalt mit der Einschränkung, daß auf der linken Seite nur eine einzelne Variable stehen darf (mit dieser Einschränkung lassen sich nur noch die kontextfreien Sprachen darstellen (s. [HU94])). Die rechte Seite einer Produktion besteht in der Grundgestalt aus einer Folge von Elementen (Variablen oder Terminalsymbolen).

Des weiteren wurden für die rechte Seite einer Produktion einige neue Konstrukte eingeführt. Diese dienen im wesentlichen einer kompakteren Darstellung der Produktionen.

Alternative

`"A | B"` bedeutet, das entweder das Element A oder das Element B in der Produktion verwendet werden kann (`"V = Dies | Das"` bedeutet also V darf durch Dies oder Das ersetzt werden).

Zusammenfassung

Mehrere Elemente können durch Klammern ("`(`", "`)`") zusammengefaßt werden. Sie werden dann wie ein Element behandelt.

Wiederholung

Ein Stern ("*") vor einem Element zeigt eine Wiederholung an. Vor und nach dem Stern kann durch Zahlen eine minimale und maximale Anzahl von Wiederholungen angegeben werden (vor dem Stern steht die minimale nach dem Stern die maximale Anzahl). Fehlt die minimale Anzahl, so wird 0 angenommen, fehlt die maximale Anzahl, wird unendlich angenommen. "1*Test" steht für das mindestens einmalige Auftreten des Elements Test, "1*2Test" erlaubt das Auftreten von Test genau ein- oder zweimal.

Optionalität

Wird ein Element von eckigen Klammern ("[" , "]") umschlossen, so gilt es als optional.

Grammatik 1: Simple Header Parsing

```

Message          =   *Field *(CRLF *text)
Field            =   Field-Name ":" [Field-Body] CRLF
Field-Name       =   1*<jedes Zeichen, außer Steuerungszeichen, SPACE und
                    ":">
Field-Body       =   *Text [CRLF LWSP-char Field-Body]

```

Grammatik 2: Encoded-Word

```

Encoded-Word     =   "=?" Charset "?" Encoding "?" Encoded-Text "=?"
Charset          =   Token;
Encoding         =   Token;
Token            =   1*<Jedes CHAR außer SPACE, CTRLs und Especials>
Especials       =   "(" / ")" | "<" | ">" | "@" | "/" | ";" | ":" |
                    "/" | "[" | "]" | "?" | "." | "="
Encoded-Text    =   1*<Jedes darstellbare ASCII-Zeichen außer ?" oder
                    SPACE>

```

Grammatik 3: Regulärer Ausdruck

Expression	=	Serie Expression " " Serie
Serie	=	Singleton Serie Singleton
Singleton	=	"." "[" FullCCL "]" "(" Expression ")" "\"" String "\"" EscapedChar Singleton "*" Singleton "+" Singleton "?"
FullCCL	=	["^"] CCL
CCL	=	RangeOrChar CCL RangeOrChar
RangeOrChar	=	EscapedChar EscapedChar "-" EscapedChar
String	=	*EscapedChar
EscapedChar	=	<jedes Zeichen, das keine besondere Bedeutung für den Aufbau regulärer Ausdrücke besitzt> "/" <jedes Zeichen>

Grammatik 4: Qualifizierender Ausdruck für Gruppennamen

Expression	=	SimpleExpr SimpleExpr Expression
SimpleExpr	=	String QuotedString DollarQuotedPattern

Grammatik 5: Qualifizierender Ausdruck für Artikel

```
Expression      = OrExp
OrExp           = AndExpr |
                 AndExpr "or" AndExpr
AndExpr        = NotExpr |
                 NotExpr "and" NotExpr
NotExpr        = NearExpr |
                 "not" NearExpr
NearExpr       = SimpleExpr |
                 SimpleExpr "near" SimpleExpr |
                 "(" Expression ")"
SimpleExpr     = String |
                 QuotedString |
                 DollarQuotedPattern
```

B Test der Agenten-Suchmaschine

B.1 Konfiguration

Engine und Locations

```
# engine.cfg
# config file for a Mole engine
# Author: Thomas E. Wieger
# Date: 1.3.97

# definition of the engine
ENGINEADDRESS 129.69.210.21
ENGINEPORT 7777
ENGINESYSTEMLIST systemclasses.dat
ENGINECLASSES /twieger/Developer/Java/Mole/Classes/
ENGINEHOME /twieger/Developer/Java/Mole/

# Location 1: "Stuttgart"
LOCATIONNAME Stuttgart
LOCDESCRIPTION Heimatbasis
STARTUPFILE location1.dat
NSSTART 0.0.0.0.0.0.0.1
NSEND 0.0.0.0.0.0.255.255

# Location 2: "London"
LOCATIONNAME London
LOCDESCRIPTION Nummer 2
STARTUPFILE location2.dat
NSSTART 0.0.0.0.0.1.0.0
NSEND 0.0.0.0.0.1.255.255

# Location 3: "NewYork"
LOCATIONNAME NewYork
LOCDESCRIPTION Nummer 3
STARTUPFILE location3.dat
NSSTART 0.0.0.0.0.2.0.0
NSEND 0.0.0.0.0.2.255.255

# Location 4: "Rom"
LOCATIONNAME Rom
LOCDESCRIPTION Nummer 4
STARTUPFILE location4.dat
NSSTART 0.0.0.0.0.2.0.0
NSEND 0.0.0.0.0.2.255.255
# last line
```

Agenten der Locations

Durch die folgende Methode wurden Dateien mit persistenten Agenten erzeugt, die jeweils eine der vier in der zuvor gezeigten Datei konfigurierten Locations initial belegen. Dabei wird die erste Location als Ausgangsbasis aller Suchvorgänge mit einem SimpleTrader und einem SearchController konfiguriert, die folgenden drei Locations erhalten jeweils einen NewsService.

```

static public void main(String argv[])
{
    try
    {
        FileOutputStream      fOS;
        ObjectOutputStream    oOS;
        String                moledir =
                                "/twieger/Developer/Java/Mole/";

        /* Populate "location1"
        */
        fOS = new FileOutputStream( moledir + "location1.dat");
        oOS = new ObjectOutputStream( fOS );
        {

            /* Trader erzeugen und NewsServer anderer Locations
            registrieren
            */
            SimpleTrader      myTrader = new SimpleTrader();

            myTrader.registerLocationForService
                ( new LocationName( "London" ),
                  "News" );
            myTrader.registerLocationForService
                ( new LocationName( "NewYork" ),
                  "News" );
            myTrader.registerLocationForService
                ( new LocationName( "Rom" ),
                  "News" );

            myTrader.myname = nextName();
            oOS.writeObject( myTrader);

            /* SearchController erzeugen
            */
            SearchController  mySearchController =
                                new SearchController();

            mySearchController.myname = nextName();
            oOS.writeObject( mySearchController);
        }
        oOS.flush();

        /* Populate "location2"
        */
        fOS = new FileOutputStream( moledir + "location2.dat");
        oOS = new ObjectOutputStream( fOS );
        {
            /* NewsService Gateway-Agenten erzeugen
            */
            NewsService      myNewsService = new NewsService
                ( "C:/Scratch/News", "C:/Scratch/News" );
        }
    }
}

```

```

        myNewsService.myname = nextName();
        oOS.writeObject( myNewsService);
    }
    oOS.flush();

    /* Populate "location3"
    */

```

Analog der Belegung von "Location2" wird eine NewsService-Instanz erzeugt.

```

    /* Populate "location4"
    */

```

Analog der Belegung von "Location2" wird eine NewsService-Instanz erzeugt.

```

    }
    catch (java.lang.Exception e)
    {
        System.out.println( "Makefiles.java::Error: " + e.toString());
    }
}

```

B.2 Parallele Suche – Protokoll

Das Protokoll einer parallelen Suche wird im folgenden verkürzt wiedergegeben.

Der Suchvorgang wird mit dem Ausdruck

```
comp\.lang.c++.*
```

für Gruppennamen und

```
( ( ( program or code ) and retrieval ) or help )
```

gestartet. Zeitlimit und Trefferbegrenzung wurden maximal gewählt, so daß eine erschöpfende Suche durchgeführt wurde.

Zu Beginn des Suchvorgangs fragt der SearchController den SimpleTrader nach Locations mit dem Service "News" ab. An diese werden dann ParNewsSearchAgents geschickt.

```

SearchController::getNewsServiceLocationNames() asking for trader
SearchController::getNewsServiceLocationNames() trader found
SimpleTrader::dispatch locationsForService

Location Stuttgart: arrive of 0.0.0.0.0.0.0.2
Location: Install Agent
Location: now 4 agents
starting agent 0.0.0.0.0.0.0.2
Location Stuttgart: arrive of 0.0.0.0.0.0.0.3
Location: Install Agent
Location: now 5 agents
starting agent 0.0.0.0.0.0.0.3
ParNewsSearchAgent::arrived at Location:Stuttgart( Heimatbasis)
Location 0.0.0.0.0.0.0.3 wants to go to NewYork

```

Erzeugung und Ankunft der weiteren ParNewsSearchAgents in Stuttgart gekürzt.

Ein ParNewsSearchAgent kommt in NewYork an.

```
Location NewYork: arrive of 0.0.0.0.0.0.0.3
Location: Install Agent
Location: now 3 agents
starting agent 0.0.0.0.0.0.0.3
ParNewsSearchAgent::arrived at Location:NewYork(Nummer 3)
NewsSearchAgent(0.0.0.0.0.0.0.3)::Trying to get groupnames
```

Ankunft weiterer ParNewsSearchAgents an ihren Zielorten wurde gekürzt. Nach der Ankunft fragen die Such-Agenten beim NewsService die verfügbaren Gruppen ab. Wird eine "passende" Gruppe gefunden, so wird der SearchController informiert. Der SearchController gibt dann zurück, ob der Such-Agent in dieser Gruppe suchen muß.

```
SearchController::dispatch for 0.0.0.0.0.0.0.3 method examineGroupNamed
NewsSearchAgent(0.0.0.0.0.0.0.3)::Matched Group:comp.lang.c++
SearchController::dispatch for 0.0.0.0.0.0.0.4 method examineGroupNamed
SearchController::dispatch for 0.0.0.0.0.0.0.4 method examineGroupNamed
NewsSearchAgent(0.0.0.0.0.0.0.4)::Matched Group:comp.lang.c++.leda
SearchController::dispatch for 0.0.0.0.0.0.0.2 method examineGroupNamed
SearchController::dispatch for 0.0.0.0.0.0.0.2 method examineGroupNamed
SearchController::dispatch for 0.0.0.0.0.0.0.2 method examineGroupNamed
NewsSearchAgent(0.0.0.0.0.0.0.2)::Matched
Group:comp.lang.c++.moderated
SearchController::dispatch for 0.0.0.0.0.0.0.4 method examineGroupNamed
```

Der erste Agent hat alle verfügbaren Gruppen untersucht. Da zum Teil schon andere Agenten in diesen Gruppen suchen, ist die Arbeit für ihn erledigt. Er kehrt daher an seine Heimatbasis zurück.

```
NewsSearchAgent(0.0.0.0.0.0.0.4)::Stopped all groups examined

Location 0.0.0.0.0.0.0.4 wants to go to Stuttgart
Location: now 2 agents
Location: MigrationMessage
Location Stuttgart: arrive of 0.0.0.0.0.0.0.4
Location: Install Agent
Location: now 4 agents
starting agent 0.0.0.0.0.0.0.4
ParNewsSearchAgent::arrived at Location:Stuttgart( Heimatbasis)
```

An der Heimatbasis angekommen, meldet sich der Agent beim SearchController ab.

```
SearchController::dispatch for 0.0.0.0.0.0.0.4 method searchDone
SearchController::Another SearchAgent is back, now at work:2
Location : Agent 0.0.0.0.0.0.0.4 wants to die
Location: now 3 agents
```

Ein anderer Agent hat in inzwischen einen "passenden" Artikel gefunden. Er meldet diesen dem SearchController.

```
NewsSearchAgent(0.0.0.0.0.0.0.3)::Matched Article:67294
NewsSearchAgent(0.0.0.0.0.0.0.3)::Subject: Formatted INPUT, Please
help!!

SearchController::dispatch for 0.0.0.0.0.0.0.3 method
checkStopForFoundArticleWithIDFromGroupNamed
SearchController::received result No.:1
```

Der zweite Agent ist ebenfalls mit der Suche fertig und kehrt zurück. Der letzte Agent (insgesamt wurden drei Agenten ausgesendet) findet in weiteren Verlauf noch zehn weitere Artikel, die den Kriterien entsprechen. Die Ausgaben dieser Vorgänge wurden gekürzt.

```
SearchController::dispatch for 0.0.0.0.0.0.0.3 method
checkStopForFoundArticleWithIDFromGroupNamed
SearchController::received result No.:12
SearchController::dispatch for 0.0.0.0.0.0.0.3 method examineGroupNamed
SearchController::dispatch for 0.0.0.0.0.0.0.3 method examineGroupNamed
NewsSearchAgent(0.0.0.0.0.0.0.3())::Stopped all groups examined
```

```
Location 0.0.0.0.0.0.0.3 wants to go to Stuttgart
Location: now 2 agents
Location: MigrationMessage
Location Stuttgart: arrive of 0.0.0.0.0.0.0.3
Location: Install Agent
Location: now 4 agents
starting agent 0.0.0.0.0.0.0.3
ParNewsSearchAgent::arrived at Location:Stuttgart( Heimatbasis)
SearchController::dispatch for 0.0.0.0.0.0.0.3 method searchDone
SearchController::Another SearchAgent is back, now at work:0
SearchController:Search done. Time needed:615054
Location : Agent 0.0.0.0.0.0.0.3 wants to die
Location: now 3 agents
```

Nachdem der letzte Agent seine Arbeit beendet hat und wieder an der Heimatbasis angekommen ist, ist die Suche beendet. Der SearchController hat für die Gesamtdauer der Suche eine Zeit von 615.054 Millisekunden, also 10 Minuten und 15 Sekunden ermittelt.

C Verzeichnis der Abbildungen

Abb. 1: Verteilung von News

Abb. 2: Ein Mobiles-Agenten-System im Überblick

Abb. 3: Systemarchitektur von Mole

Abb. 4: Sequentielle Suche

Abb. 5: Parallele Suche

Abb. 6: Dialogfenster der Benutzerschnittstelle

Abb. 7: Vererbungshierarchie der Expression-Klassen

Abb. 8: Datenstruktur einer index-basierten Suchmaschine

D Literaturverzeichnis

Vorbemerkung

Viele der im folgenden genannten Dokumente liegen nur im Hypertext-Format vor. Da in diesem Fall in der Regel nicht mit Seitenzahlen gearbeitet werden kann, führt der Verfasser die WWW-Adresse (URL) jeder einzelnen WWW-Seite als eigenen Literaturhinweis an.

Einige Angaben lassen sich nicht direkt als "Literatur"-Hinweise auffassen, sie geben vielmehr die WWW-Adresse an, über die auf Funktionen eines bestimmten Programms zugegriffen werden kann. So gibt [ALTAVISTA97] die Zugriffsadresse der Suchmaschine *AltaVista* der Digital Equipment Corp. an. Auf die durch die Bedienungshinweise des Programms gewonnenen Informationen wird ebenfalls durch Angabe der WWW-Adresse des Programms hingewiesen.

Literaturhinweise

- [ALTASOFT97] Unbekannt: *AltaVista MarketSpace*
Digital Equipment Corp., 1997
<http://altavista.software.digital.com>
- [ALTAVISTA97] Verweis auf die Suchmaschine *AltaVista*
Digital Equipment Corp., 1997
<http://www.altavista.digital.com>
- [ASU86] Aho, Alred. V; Sethi, Ravi; Ullman, Jeffrey D.: *Compilers - Principles, Techniques and Tools*.
Addison-Wesley, 1986
- [CALC97] Verweis auf den *Usenet Calculator*
Netpart Inc., 1997
<http://www.netpart.com/janus/usenet.html>
- [DJN97] Verweis auf die Suchmaschine *DejaNews*.
Deja News Inc.
<http://www.dejanews.com>
- [DJN_ABOUT] Unbekannt: *Deja News - Who are we?*.
Deja News Inc.,1997
<http://www.dejanews.com/pr/dnabout.html>
- [DJN_NET] Unbekannt: *Deja News - What is Usenet?*.
Deja News Inc.,1997
http://www.dejanews.com/help/dnusetnet_help.html
- [DJN_PR1] Unbekannt: *Deja News - Press Release: May 30, 1996*.
Deja News Inc.,1996
http://www.dejanews.com/pr/dnpr_960530.html

-
- [DJN_PR2] Unbekannt: *Deja News - Press Release: February 4, 1997.*
Deja News Inc., 1997
http://www.dejanews.com/pr/dnpr_970210.html
- [DJN_WHY] Unbekannt: *Why you need Deja News!.*
Deja News Inc., 1997
<http://www.dejanews.com/pr/dnwhy.html>
- [EXCITE97] Verweis auf die Suchdienste *Excite*
Excite Inc.
<http://www.excite.com>
- [GJS96] Gosling, J.; Joy, B.; Steele, G.: *The Java™ Language Specification.*
Addison-Wesley, 1996
- [HCK95] Harrison, Colin G.; Chess, M. David; Kershenbaum, Aaron: *Resarch Report: Mobile Agents: Are they a good idea?*
IBM Research Division, T.J.Watson Research Center, Yorktown Heights, 1995
- [HOHL95] Hohl, Fritz: *Konzeption eines einfachen Agentensystems und Implementation eines Prototyps.*
Diplomarbeit Nr. 1267, Institut für Parallele und Verteilte Höchstleistungsrechner, Universität Stuttgart, 1995
- [HU94] Hopcroft, John E.; Ullman, Jeffrey D.: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*
Addison-Wesley, 1994
- [IIR95] Unbekannt: *Intelligent agents and information retrieval.*
Information Interchange Report Vol 2 No. 8, 1995
- [INN96] Verschiedene: Dokumentation und Sourcecode der Software *Internet News.*
<ftp://ftp.isc.org/isc/inn/inn-1.5.1.tar.gz>
- [JANSEN97] Jansen, Major Jim: *Dissertation Proposal: Using an Intelligent Agent to Enhance Search Engine Performance.*
Department of Electrical Engineering and Computer Science,
United States Military Academy, West Point, New York, 1997
<http://www.eecs.usma.edu/usma/academic/eecs/instruct/jansen/PROPOUT.html>
- [JDK96] Verschiedene: *Java API Documentation*
Sun Microsystems Inc., 1996
<http://java.sun.com/Products/JDK/currentrelease>
- [JTUT97] Verschiedene: *The Java Tutorial - Object oriented programming for the internet*
Sun Microsystem Inc., 1997
<ftp://ftp.javasoft.com/docs/Tutorial.zip>
- [MOLE96] Hohl, Fritz: *Online Dokumentation zum Projekt Mole: - Alpha Release 1.0.*
<http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html>.
Universität Stuttgart, 1996.

- [MW97] Diverse: Dokumentation zur Entwicklungsumgebung *CodeWarrior Professional 9* für Apple Macintosh
Metrowerks Inc., 1997
- [RMI] Diverse: Dokumentation zum *Remote-Method-Invocation-Verfahren*
Sun Microsystems Inc., 1996
<http://chatsubo.javasoft.com/current/>
- [RFC1036] Horton, M.; Adams, R.: *Standard for Interchange of USENET Messages*.
RFC 1036, 1987
- [RFC1521] Borenstein, N.; Bellcore; Freed, N.: *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*.
RFC 1521, 1993
- [RFC1522] Moore, K.: *MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text*.
RFC 1522, 1993
- [RFC822] Crocker, David H.: *Standard for ARPA Internet Text Messages*.
RFC 822, 13. 1982
- [RFC977] Kantor, Brian; Lapsley, Phil: *Network News Transfer Protocol*
RFC 977, 1986
- [SEARCHBOTS] Unbekannt: *CIG Searchbots*.
University of Massachusetts, 1997
<http://dis.cs.umass.edu/research/searchbots.html>
- [SIGMA] Ferguson, Innes A.; Karakoulas, Grigoris J.: *SIGMA: Multiagent Learning and Adaptation in an Information Filtering Market*.
Institute for Information Technology, Ottawa ON
<http://ai.iit.nrc.ca/CALVIN/SIGMA/title.html>
- [SW92] Sedgewick, Robert: *Algorithmen in C*
Addison-Wesley, 1992
- [SWISH] Hughes, Kevin: Online-Dokumentation zur Software *SWISH* (Simple Web Indexing System for Humans)
<http://www.eit.com/software/swish/>
- [SYMANTEC] Diverse: Dokumentation zur Entwicklungsumgebung *Symantec Café* Version 1.5.1
Symantec Corp., 1996
- [UNIC] Unicode Consortium, The: *The Unicode Standard: Worldwide Character Encoding Version 1.0*
<http://www.unicode.org>

Erklärung

Ich versichere hiermit, daß ich diese Arbeit selbstständig verfaßt und nur die angegebenen Hilfsmittel verwendet habe.

Thomas Wieger