Vorwort

Der Trend zu einer immer stärkeren Kopplung von Systemen bei gleichzeitiger Dezentralisierung durch Vernetzung hat dazu geführt, daß Computernutzern auf Wunsch enorme Datenmengen zur Verfügung stehen, die sich einer sinnvollen Bearbeitung durch den Nutzer allein völlig entziehen. Unterschiedliche Repräsentationsformalismen für Informationen, Mehrdeutigkeiten, Redundanz sowie eingeschränkte Verfügbarkeit sowohl von Informationen als auch von Rechenleistung machen konventionelle Suchverfahren unanwendbar. Stattdessen werden Suchverfahren und Programme benötigt, die sich intelligent an unterschiedliche Formalismen anpassen, ihre Handlungen ständig evaluieren und fähig sind, ihre Benutzer individuell zu unterstützen. Schlagwörter wie Knowbots, Search-Engines oder Data-Mining sind deshalb zur Zeit in aller Munde.

Ein umfassendes Buch, das die hinter diesen und ähnlichen Schlagwörtern verborgenen Ideen und Konzepte präsentiert, existiert jedoch zur Zeit noch nicht. Dies war für uns die Motivation, das Thema "Intelligente Suche im Internet mit Lernenden Systemen" in einem Seminar zu behandeln. Wir haben damit ein Forschungsgebiet aufgegriffen, das sowohl für alle am LSA beteiligten Gruppen von Interesse ist, aber darüber hinaus aktuell von vielen Seiten aufmerksam beobachtet wird. Daher haben wir uns entschlossen, die Ausarbeitungen, die im Rahmen dieses Seminars von den TeilmehmerInnen erstellt wurden, durch den vorliegenden Bericht einer breiteren Öffentlichkeit zugänglich zu machen.

Durch die individuellen Interessen und unterschiedlichen Sichtweisen der am LSA beteiligten Arbeitsgruppen ergab sich ein breites Themenspektrum. Die Palette reichte von theoretischen Grundlagen über grundsätzliche Suchkonzepte und persönliche Assistenten bis hin zu den bekannten "Search-Engines" und konkreten Anleitungen für den Bau eigener Agenten.

Dank des Enthusiasmuses aller beteiligten Studenten, die sich auch von schwer zugänglicher Literatur oder fehlenden Dokumentationen nicht behindern ließen, liegt mit den in diesem Bericht enthaltenen Ausarbeitungen für Einsteiger in die Problematik Suche und Agenten im Internet eine gute

Übersicht vor, die einen Startpunkt für eigene Experimente und Erfahrungen bilden kann.

Neben den beteiligten Studenten, deren Namen bei ihren Ausarbeitungen zu finden sind, möchten wir uns bei unseren Kollegen Matthias Fuchs, Dr. Klaus-Werner Jörg, Werner Stein und Wolfgang Wilke für die gute Zusammenarbeit und ihre Bereitschaft, sich in für sie nicht alltäglich betrachtete Gebiete einzuarbeiten, bedanken.

Kaiserslautern, im August 1996
J. Denzinger & R. Bergmann

Inhaltsverzeichnis

Kapitel 1

Die Geschichte des Internet und seiner Dienste

Christian Schreiber

Übersicht

Das Internet wird im September 27 Jahre alt. 1969 wurden die ersten 4 Hosts zu ARPANET, dem Vorgänger des Internet, verbunden.

In den 60er Jahren, noch zu Zeiten des kalten Krieges, stand das amerikanische Verteidigungsminiterium vor dem Problem, daß ihr Informationssystem verhältnismäßig einfach (z.B. durch einen gezielten Nuklearanschlag) lahmgelegt werden konnte. Es wollte deswegen ein dezentralisiertes Computernetzwerk einführen. Dieses Vorhaben, unter Leitung der Defense Advanced Research Projects Agency wurde bekannt unter dem Namen ARPANET. 1969 dann wurde der erste Interface Message Processor, ein Vorgänger heutiger Router, an der Universität von Kalifornien, Los Angeles eingesetzt. Ende des Jahres markierte die Verbindung von 4 zu militärischen Zwecken verbundenen Hosts die Geburtsstunde des heutigen Internet. Zwischen 1969 und 1983 führten Arbeiten auf dem Gebiet der Netzwerkprotokolle zu den heute bekannten Protokollen TCP und IP. Das Netz an sich wuchs unaufhaltsam.

Die Vielseitigkeit von TCP/IP führte zu ihrer Anwendung auch außerhalb ARPANET: die National Science Foundation (NSF) gründete 1987 das NSFNET um ihre Großrechner mit

Universitäten zu verbinden. Es war öffentlich zugänglich für Amerikaner und ihre Alliierten. Die Verbindung von NFSNET und hinzugekommenen regionalen Netzwerken wurde erstmals unter dem Namen Internet bekannt.

Heute ist das Internet und seine Dienste e-mail, FTP, Telnet, WWW in aller Munde und nahezu für jedermann zugänglich. Es sind bereits 11 Millionen Hosts miteinander verbunden, und man schätzt, daß ihre Zahl jeden Monat um 6 % weiter wächst.

1.1 Die Geschichte des Internet

"It is not proper to think of networks as connecting computers. Rather, they connect people using computers to mediate. The great success of the internet is not technical, but in human impact. Electronic mail may not be a wonderful advance in Computer Science, but it is a whole new way for people to communicate. The continued growth of the Internet is a technical challenge to all of us, but we must never loose sight of where we came from, the great change we have worked on the larger computer community, and the great potential we have for future change.“

David Clark, senior research scientist at MIT's Laboratory for Computer Science

1.1.1 "Prähistorie“

Bereits im Jahre **1858** mit der Verlegung des ersten transkontinentalen Kabels auf dem Grund des Atlantischen Ozeans wurde der erste Schritt zur vernetzten Gesellschaft gemacht. Es sollte eine schnelle Kommunikation zwischen den USA und Europa ermöglichen. Das Kabel wurde als Meilenstein in der Geschichte der Menschheit angesehen, es blieb jedoch nur wenige Tage in Betrieb: das Kabel war ein technischer Flop. 1866 wurden weitere Kabel verlegt; sie waren ein voller Erfolg und blieben fast hundert Jahre in Betrieb.

Als **1957** die UdSSR den ersten Satelliten in die Erdumlaufbahn schoß, fühlte sich die USA in ihrer nationalen Ehre erheblich gekränkt: Ausgerechnet vom kommunistischen Feind lief die selbsternannte "leading nation“Gefahr technologisch und vor allem militärisch abgehängt zu werden. In der Zeit des Kalten Krieges von den Ereignissen äußerst beunruhigt, sah sich US-Präsident Dwight D. Eisenhower gezwungen, innerhalb des *Department of Defense (DoD)* die *Advanced Research Projects Agency (ARPA)* zu gründen. Sie sollte die amerikanische Vorherrschaft auf dem Gebiet der Kommunikations- und Überwachungstechnik garantieren.

Die Organisation vereinigte einige der fähigsten Leute Amerikas, die in nur 18 Monaten den ersten funktionsfähigen amerikanischen Satelliten zusammenbauten. Die Forschungsaktivitäten der ARPA verlagerten sich jedoch in den Folgejahren auf die Vernetzung von Computern, da diese zunehmend für sicherheitsrelevante Aufgaben im Verteidigungsbereich eingesetzt wurden.

1.1.2 Die 60er: Von Routen und Päckchen...

1962 übernahm Dr. J.C.R. Licklider die Forschung der ARPA auf dem Gebiet der militärischen Nutzung der Computertechnologie, insbesondere deren Vernetzung. Um schnellere Erfolge zu erzielen, bezog er nationale Universitäten in die Forschung mit ein.

Den Hauptfeind immer vor Augen, war das strategische Problem schnell formuliert: wie könnten die amerikanischen Verantwortungsträger nach einem Atomkrieg noch miteinander kommunizieren ?

Die ARPA kam zu dem Schluß, daß Amerika ein flächendeckendes Kommunikationsnetz benötigte. Es sollte Staaten, Städte und vor allem militärische Basen miteinander verbinden. Doch dessen Schaltstationen und Leitungen würden immer einem gezielten Nuklearanschlag ausgesetzt sein, egal wie stark diese geschützt oder abgeschirmt wären. Ein herkömmliches Netzwerk konnte diese Zuverlässigkeitsbedingungen nicht erfüllen.

Wie aber sollte dieses neuartige Netzwerk gesteuert werden? Jede zentrale Autorität wäre ein offensichtliches Ziel für einen gegnerischen Angriff. Die auch auf diesem Gebiet tätige *RAND Corporation* behandelte dieses verzwickte Problem unter strengster Geheimhaltung.

1962 machte Paul Baran von der RAND Corporation in seinen Arbeiten erste Vorschläge für ein packetvermittelndes Netzwerk.

Baran's Vorschlag wurde **1964** publik: Basiskonzept war, daß es im Netzwerk keine zentrale Autorität gäbe, weiter sollte es schon im unvollständigen Anfangsstadium zuverlässig arbeiten.

Die Prinzipien waren einfach. Das Netzwerk selbst wurde als jederzeit unzuverlässig angenommen. Es sollte von Anfang an seine eigene Unzuverlässigkeit überwinden. Alle Knoten im Netzwerk sollten den gleichen Status besitzen, jeder Knoten die Berechtigung haben, Nachrichten zu erzeugen, zu übertragen und zu empfangen. Die Nachrichten selbst sollten in Pakete aufgeteilt sein, wovon jedes einzeln addressiert würde. Jedes einzelne Paket hätte einen bestimmten Start- und einen bestimmten Zielknoten. Es sollte sich dann eigenständig seinen Weg durchs Netzwerk bahnen.

Die genaue Route, die das Paket einschlägt, sollte unrelevant sein. Nur die angekommenen Daten sollten zählen. Also würde das Paket wie eine heiße

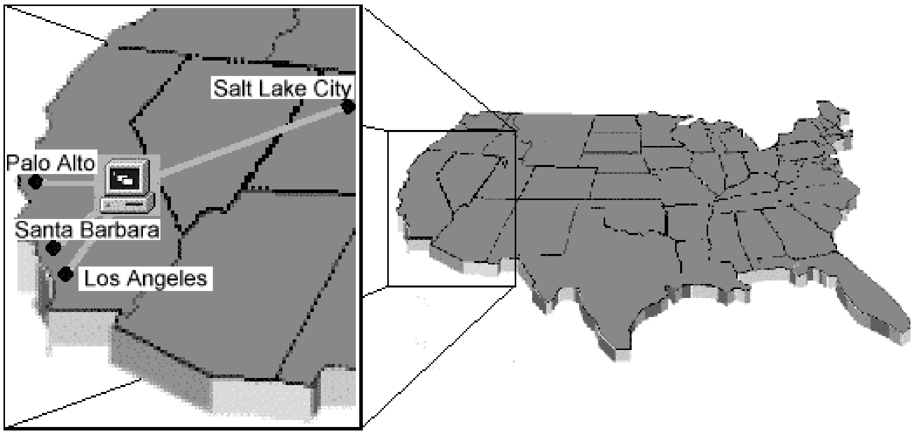


Abbildung 1.1: *die ersten 4 Knoten des ARPANET: UCLA in Los Angeles, UCSB in Santa Barbara, Stanford Research Institute in Palo Alto und die Universität von Utah in Salt Lake City*

Kartoffel von einem Knoten zum anderen weitergereicht werden, mehr oder weniger in Richtung des Zielknotens, bis es dann dort ankäme. Wären große Teile des Netzwerkes beschädigt, so wäre das nicht schlimm; das Paket würde sich seinen Weg über die noch intakten Knoten bahnen. Dieses auf den ersten Blick chaotisch anmutende System, mochte nicht das effizienteste sein (verglichen z.B. mit dem Telefonsystem) – doch es wäre extrem zuverlässig.

In den 60er Jahren wurde dieses fesselnde Konzept eines "bombensicheren", dezentralisierten, paketorientierten Netzwerkes von der RAND, dem *Massachusetts Institute of Technology (MIT)* und der *University of California, Los Angeles (UCLA)* unter hohem Aufwand vorangetrieben. Das *National Physical Laboratory (NPL)* in Großbritannien installierte **1967** das erste Testnetzwerk basierend auf diesen Prinzipien. Kurz danach entschied die ARPA, ein größeres, ehrgeizigeres Konzept in den USA auf die Beine zu stellen. Die Knoten diese Netzwerkes sollten sehr schnelle Supercomputer sein. Zu der Zeit waren hohe Rechenleistungen selten und sehr teuer. Für nationale Forschungs- und Entwicklungsarbeiten wäre ein Zugriff auf diese Computer über ein leistungsfähiges Netzwerk von großem Interesse.

Im Herbst **1969** wurde der erste *Interface Message Processor (IMP)* an der UCLA eingerichtet. Im Dezember des gleichen Jahres bestand das Mini-Netzwerk dann aus vier Knoten (Abb. 1.1), das fortan, nach seinem Sponsor, ARPANET genannt wurde.

Die vier Supercomputer konnten Daten über geeignete High-Speed-Leitungen übertragen und sogar von den übrigen Knoten aus programmiert werden.

Dank ARPANET konnten Wissenschaftler und Forscher gegenseitig Rechnerleistung über große Entfernungen hinweg nutzen. Dies war ein sehr begrüßter Service, da in den frühen 70er Jahren die Rechenzeit sehr kostbar war.

1.1.3 Die 70er: Ein leistungsfähiges Protoll: TCP/IP

Das *Network Control Protocol (NCP)* wurde **1970** Standard im ARPANET und im Jahr darauf (**1971**) umfaßte es schon fünfzehn Knoten. Mit der Einführung eines `e-mail` Programmes wurde im zweiten Jahr der Inbetriebnahme von ARPANET ein besonderes Phänomen sichbar: die meisten Benutzer des ARPANET hatten das für die Kommunikation von Computern vorgesehene Netz mehr oder weniger in einen elektronischen Hochgeschwindigkeits-Postdienst umfunktioniert. Die Mehrzahl der Übertragungen im ARPANET waren nämlich keine Berechnungen an entfernten Rechnern, sondern News und persönliche Mitteilungen. Foscher, die das ARPANET benutzten, um über Projekte zu diskutieren und ihre Arbeiten zu koordinieren, ließen es sich nicht nehmen, auf diesem Weg auch mal Klatsch und Tratsch zu verbreiten. Persönliche Benutzer-Accounts mit eigenen Mail-Adressen fanden demnach rasch Verbreitung. Viele nutzten also die Verbindung von Computern hauptsächlich als neue Möglichkeit der Mensch-zu-Mensch-Kommunikation Die Erfindung der *mailing-list*, eine Rundsendetechnik des ARPANET, bei der eine einzige Nachricht an eine Vielzahl von Netzteilnehmern geschickt werden konnte, entstand in diesem Umfeld. Eine der ersten großen mailing-lists war die "SF-LOVERS"für Science-Fiction Fans. Diskussionen über dieses und ähnliche Themen standen nicht in direktem Zusammenhang mit einer wissenschaftlichen Arbeit und wurden von vielen ARPANET System-Administratoren mehr geduldet als geschätzt, was ihre Beliebtheit aber nicht schmälern konnte.

Das ARPANET wuchs in den 70er Jahren kontinuierlich. Die dezentralisierte Struktur des ARPANET unterstützte wesentlich seine Erweiterung. Im Gegensatz zu üblichen Computernetzen konnte das ARPANET viele verschiedene Rechnertypen miteinander verbinden. So lange diese Maschinen die Standardregeln der Paketübertragung beherrschten, waren Hersteller, Typ und Besitzer unwesentlich.

Im Zuge dieses Wachstums sorgte im Oktober **1972** eine öffentliche Vorführung auf der ersten *International Conference on Computer Communication* für Aufsehen. Auf dieser Konferenz, die im Washingtoner Hilton Hotel stattfand, konnte die interessierte Öffentlichkeit dieses neue Kommunikationsmedium hautnah erleben. Mit Hilfe eines eilig dort installierten *Terminal Interface Processors (TIP)* konnte jeder das ARPANET benutzen und Anwendungen verteilt in den ganzen USA starten. Bei dieser spektakulären Vorführung wurden viele Skeptiker der ersten Stunde von der Antwortzeit und der Robustheit des Systems eines Besseren gelehrt. Aus diesem Treffen, wo auch das gesamte Who-Is-Who der damaligen Netzwerkszene anwesend war, ging die *International Network Working Group (INWG)* hervor. Hauptzweck dieser Vereinigung von Fachleuten war es, Einigkeit auf dem Gebiet der Protokolle zu erzielen.

Die INWG brachte dann *X.25*, einen internationalen Standard für die Paketvermittlung über ein Netzwerk hervor. Außerdem brauchte man Standards für die kommerziellen Netzwerke in den USA, Kanada, Frankreich und Großbritannien wie z.B. Telenet, Datapac, das Experimental Packet Switching System, Transpac und das Réseau Communication par Paquet (RCP).

1973 wurde ARPANET dann international: das *University College of London* und das *Royal Radar Establishment* (Norwegen) wurden vernetzt.

Das `File Transfer Protocol (FTP)`wurde spezifiziert und erste Vorschläge für *Ethernet* geäußert.

Die Verbindung von heterogenen Netzwerken stellte noch immer ein schwieriges Problem dar. Die ARPA, allen voran Bob Kahn und Vinton Cerf[1] , startete ein Forschungsprogram, das die Lösung dieses Problem zum Ziel hatte: *Gateways* sollten diese Verbindungen übernehmen.

Erste Resultate waren schon **1974** zu vermelden: das bisherige NCP sollte nach Bob Kahn und Vinton Cerf's Publikation "A Protocol for Packet Network Intercommunication“durch das neue *Transmission Control Protocol (TCP)* ersetzt werden.

Mitte **1975** wurde das Management des ARPANET von der DARPA[2] zu einer eigenen Organisation, der *Defense Communication Agency (DCA, heute DISA)* übertragen.

TCP und das komplemantäre *Internet Protocol (IP)* erwiesen sich als äußerst leistungsfähig. **1977** gelang es, vier verschiedenartige Netzwerke, nämlich das ARPANET, das für Satellitenübertragung eingeführte *SATNET*, Ethernet und das funkorientierte *PRNET* miteinander zu verbinden..

Um das als zukunftsweisend angesehene TCP/IP zu perfektionieren, wurde **1979** das *Internet Configuration Control Board (ICCB)* gegründet. Im gleichen Jahr wurde `USENET` mit den ersten newsgroups eingeführt.

1.1.4 Die 80er: größer, schneller und für jeden: das NSFNET

In den 80er Jahren wurden leistungsfähige Computer für die verschiedensten sozialen Gruppen erschwinglich und auch die Anbindung an ein Netz erforderte keinen außergewöhnlichen Aufwand . Die Verbreitung von TCP/IP

[1] Vinton Cerf ist heute Senior Vice President bei der Data Service Division von *MCI*

[2] Aus ARPA wurde mittlerweile *DARPA, Defense Advanced Research Projects Agency*

führte dazu, daß ganze Netzwerke sich mehr oder weniger organisiert ins ARPANET einhängten. Da TCP/IP frei verteilt wurde und die Basistechnologie des Netzes dezentral war, wurde es sehr schwer zu kontrollieren, wer sich wo und warum Zugang zum Netz verschaffte.

Gegen **1980** wurde TCP/IP zum Standard im militärischen Bereich der Datenübertragung. Nun wollte man dafür sorgen, daß auch alle Computer im ARPANET das TCP/IP benutzten.

Das *BITNET* wurde **1981** hauptsächlich eingeführt, um IBM-Mainframes aus Universitäts-Rechenzentren miteinander zu verbinden.

Nach einigen Testphasen wurde am 1. Januar **1983** definitiv im ARPANET von NCP zu TCP/IP umgeschaltet. Um die Evolution der TCP/IP Protokollfamilie zu kontrollieren wurde im gleichen Jahr das *Internet Activities Board (IAB)* gegründet. Gleichzeitig entwickelte man an der Universität von Wisconsin den ersten *Name Server*. Er ermöglichte es den Netzbenutzern, entfernte Hosts zu adressieren, ohne den genauen Pfad zu kennen. Mit Berkeley UNIX wurde IP-Netzwerksoftware jetzt auch für Desktop-Workstations verfügbar.

Ebenfalls 1983 machte sich der militärische Teil des ARPANET selbständig und wurde zu *MILNET (zuerst Defense Data Network, DDN)*. Dank TCP/-IP blieben beide jedoch miteinander in Verbindung. Obwohl ARPANET selbst noch wuchs, verblaßte es doch im Vergleich zu den explosionsartig wachsenden, hinzugelinkten neuen Netzwerken.

Während DARPA und DCA ihre Verbündeten dazu bewogen, auf TCP/IP umzustellen, griff **1984** die *National Science Foundation (NSF)* über ihr Office of Advanced Scientific Computing ins Geschehen ein. Das neu gegründete, auf TCP/IP basierende NSFNET (zuerst Computer Science Network, CSNET) war Maßstab für den technologischen Fortschritt indem es neuere, schnellere Supercomputer mit dickeren, schnelleren Leitungen verband. Mit NSFNET konnte damals als Novum ein mail-Dienst über Telefonleitungen angeboten werden. Erstmals konnte jetzt die breite Öffentlichkeit, die bislang keinen Zugang zum ARPANET hatte, mit dem NSFNET die Dienste eines überregionalen Computernetzes benutzen. Besonders Universitäten nutzten dies, um ihre Ausbildung zu verbessern.

Während 1984 auch die ersten *Domain Name Server (DNS)* eingeführt wurden, wuchs die Zahl der teilnehmenden Hosts auf über 1.000.

Das NFSNET wurde konsequent weiterentwickelt: zuerst **1986** und besonders **1988**, als *MERIT* an der Universität von Michigan zusammen mit MCI und IBM ein 1,5 Mbps Backbone[3] entwickelten. Das NFSNET wurde damit rund 30mal schneller als mit dem vorherigen ARPANET-Standard von 56

[3]Das 1,5 Mbps Backbone wurde auch als Evolutionsstufe T1 bezeichnet

Kbps. Auch andere Regierungsstellen klinkten sich jetzt ein: die NASA, das National Institute of Health, das Department of Energy... Das NSFNET mit seinen mittels TCP/IP verbundenen lokalen Netzwerken wurde fortan in der Öffentlichkeit als "Internet"bekannt. Die Zahl der Host war auf über 10.000 gewachsen.

Das NSFNET dehnte sich über die ganze Welt aus: 1988 wurden Dänemark, Finland, Frankreich, Kanada, Island, Norwegen und Schweden aufgenommen.

ARPANET selbst hörte **1989** auf zu existieren. Es konnte die wachsenden Anforderungen an Geschwindigkeit und Übertragungsmengen nicht mehr erfüllen und wurde somit ein Opfer seines überwältigenden Erfolges. Seine Benutzer merkten kaum etwas davon, da die Funktionen des ARPANET durch das NSFNET nicht nur weiter angeboten wurden, sondern sich kontinuierlich weiterentwickelten. Der TCP/IP Standard für Computer-Netzwerke war mittlerweile weltweit etabliert.

Die Zahl der Hosts des Netzes stieg auf über 100.000 und es kamen weitere Teilnehmer dazu: Australien, Deutschland, Großbritannien, Israel, Italien, Japan, Mexiko, die Niederlande, Neuseeland und Puerto-Rico.

Die 80er Jahre waren aber auch durch das Aufkommen von kommerziellen Internet-Anwendungen gekennzeichnet: das Interesse am Internet in der Forschung, der Lehre, der Regierung und auch der Bevölkerung garantierten ein weites Kundenspektrum für engagierte Firmen. Dieser Trend zur Kommerzialisierung sollte sich in den 90er Jahren weiter fortsetzen.

1.1.5 Die 90er: Der Siegeszug des World Wide Web

Die Wachstumsrate des Internet in den frühen 90er Jahren entwickelte sich exponentiell in bezug auf Zahl der Netzwerke, Zahl der Hosts und Übetragungsvolumen. Das Netz wuchs sogar schneller als der Verbreitungsgrad von Mobiltelefonen oder Faxgeräten.

1990 traten Argentinien, Belgien, Brasilien, Chile, Griechenland, Indien, Irland, Österreich, Süd-Korea, Spanien und die Schweiz dem NSFNET bei.

Die Übertragungskapazität des NSFNET wuchs **1991** auf über eine Billion Bytes pro Monat. Währenddessen wurde das NSFNET-Backbone auf 44 Mbps[4] hochgefahren. Neu im Netz waren Hong-Kong, Kroatien, Polen, Portugal, Singapur, Süd-Afrika, Taiwan, Ungarn und die Tschechien.

1992 wurde die *Internet Society (ISOC)* gegründet um die weltweiten Aktivitäten im und ums Internet zu koordinieren. Anschluß ans NSFNET be-

[4] Dies war die Evolutionsstufe T2

kamen im gleichen Jahr: Ekuador, Estland, Kamerun, Kuwait, Litauen, Luxemburg, Malaysia, die Slowakei, Thailand, Venezuela und Zypern.

Ein Grund für dieses spektakuläre Wachstum des Internet ist in einer Reihe von neuen Diensten zu sehen, z.B. Archivierungs- und Suchdienste, die es dem User ermöglichten, sich in den riesigen Datenmengen des Netzes zu orientieren. Viele dieser Dienste machten den Weg von einem Forschungsprojekt einer Universität zu einer kommerziellen Anwendung. Beispiele hierfür sind: der *Wide Area Information Service (WAIS)*, *Archie* (wird mittlerweile von der kanadischen Firma Bunyip betrieben), *LYCOS* von Carnegie Mellon und *YAHOO* aus Stanford. Eine wahre Killeranwendung für das Netz und diese Dienste war (und ist) das Anfang der 90er aufgekommene `World Wide Web (WWW)`.

Am *European Center for Particle Research (CERN)* entwickelt, wurde das WWW in experimenteller Form schon **1989** erprobt. Offiziell wurde es aber erst 1991 zur Benutzung freigegeben. Es erregte eigentlich am Anfang kein größeres Aufsehen, bis **1992** ein junges Programmierteam am *National Center for Supercomputing Applications (NCSA)* an der Universität von Illinois auf das Web aufmerksam wurde. Es entwickelte *Mosaic*, einen graphischen Webbrowser. Mosaic wurde übers Internet gratis zum download angeboten. Es wurde ein riesiger Erfolg. Die Möglichkeit, übers Netz Bilder, Töne, Video-Clips und Texte in einem Hypertext-System anzubinden faszinierte viele Software-Provider. Zwischen 1992 und 1995 wurde eine Menge von kommerziellen Webbrowsern und -servern angeboten, darunter auch die Produkte von *Netscape Communications*. Das Internet wurde mit diesen neuen Hilfsmitteln plötzlich kinderleicht zu bedienen. Man mußte eigentlich nur eine Maus betätigen können (point and click). Die Attraktivität von Hyperlinks und graphischem Interface verhalf Netscape Communications zur Position des Marktführers, ja fast zum Monopolisten.

Auch das Weiße Haus in Washington ging jetzt online (http://www.whitehouse.gov): Präsident Bill Clinton (president@whitehouse.gov), Vize Al Gore (vice-president@whitehouse.gov) und First Lady Hillary Clinton (root@whitehouse.gov). Die Vereinten Nationen (UN) waren ebenfalls elektronisch erreichbar.

NSFNET nimmt **1993** noch einige Nachzügler auf: Ägypten, Bulgarien, Costa Rica, die Fiji-Inseln, Ghana, Guam, Indonesien, Kasachstan, Kenia, Liechtenstein, Peru, Rumänien, Rußland, Türkei, Ukraine, die Vereinigten Arabische Emirate und die Jungfern-Inseln.

Der Datenfluß übers NSFNET überstieg **1994** zehn Billionen Bytes pro Monat und das WWW wurde der zweitbeliebteste Netzdienst, hinter dem FTP, aber jetzt vor Telnet.

Im April **1995** wurde in einer aufwendigen Aktion das NSFNET-Backbone abgeschaltet: Für die Netzbenutzer erfolgte dies fast unbemerkt, was aber nur durch Initiativen der mittlerweile selbständig agierenden Internet-Serviceprovidern möglich war. Anstelle eines ehemals vollständig von der Regierung gesponsorten Systems existierte nun ein System von kommerziellen Backbones. Das NFSNET selbst besann sich auf seine Ursprünge und wurde wieder zu einem Forschungsnetzwerk. Das WWW wurde 1995 zum wichtigsten Netzdienst, was die Übertragungsmengen anging. Lag es daran daß der Vatikan online ging (http://www.vatican.va) ?

1.2 Die Dienste des Internet

Das Internet bietet hauptsächlich fünf Dienste an: elektronische Post, Diskussionsgruppen, Arbeiten an entfernten Rechnern, Dateiübertragung und WWW. Die Aufkommen dieser Dienste in der Chronologie des Internet wurde im vorigen Kapitel schon aufgezeigt, deshalb folgen hier noch einige allgemeine Informationen.

1.2.1 e-mail

Die elektronische Post im Internet, die e-mail, ist vergleichbar dem Fax, wird aber über ein elektronisches Medium übertragen und ist vor allem kostenlos. e-mail ist weltweit verfügbar und kann außer Texten auch Software und kodierte Bilder übertragen, wobei letztere (Binärdateien) nur nach Kodierung in Standard-ASCII-Zeichensatz mittels *uuencode* übertragen werden können. Zur Dekodierung steht auf der Empfängerseite dann *uudecode* zu Verfügung.

1.2.2 news

Die Diskussionsgruppen oder "newsgroups“sind eine Welt für sich. Diese Welt aus neuesten Nachrichten, Debatten und Argumentationen ist bekannt unter dem Namen "USENET“. USENET ist weniger ein physisches Netzwerk als vielmehr eine Ansammlung von sozialen Konventionen. Es gibt ungefähr 2500 verschiede newsgroups und jeden Tag werden ungefähr sieben Millionen Wörter eingetippt und zur Diskussion beigetragen.

e-mail und netnews sind sehr weit verbreitet. Für diese Dienste ist kein direkter Internet Anschluß nötig. Eine Telefonverbindung zu Internet-Randbereichen wie BITNET, UUCP oder FIDONET genügt. Die letzten drei Dienste, Arbeiten an entfernten Rechnern, Dateiübertragung und WWW benötigen einen sogenannten "direkten Internet-Zugang“mittels TCP/IP.

1.2.3 remote-login

Das Arbeiten an nicht-lokalen Rechnern[5] , seinerzeit eines der Hauptgründe für die Einführung des ARPANET, ist auch heute noch ein nützlicher Dienst. Programmierer können Accounts an weitentfernten, leistungsfähigen Computern nutzen und dort Programme ausführen oder selbst schreiben. Wissenschaftler können die Rechenleistung von Supercomputern auf einem anderen Kontinent nutzen.

1.2.4 FTP

Die Dateiübertragung mittels File Transfer Protocol (FTP) ermöglicht den Internetbenutzern, anonym Dateien von einem dafür vorgesehenen Datei-Server zu kopieren und auf ihrem eigenen Rechner kostenlos zu benutzen. Immer mehr solcher Server führten zu einem immer unübersichtlicheren Angebot an frei verfügbarer Software. Um diese enormen Datenmengen zu organisieren wurden neue Internet-Programme wie z.B. archie, gopher und WAIS entwickelt.

1.2.5 WWW

Die Attraktivität des WWW ist insbesondere auf die Leistungsfähigkeit seines Protokolls *http* und seiner Formatierungsprache *HTML* zurückzuführen: Das sich noch im Stadium der Weiterentwicklung befindliche HTML unterstützt nämlich die Einbindung von Textformatierungen, Verweisen, Tabellen, Bildern und Animationen in ein Dokument. Dadurch können Informationen sehr attraktiv mit einem Webbrowser dargestellt werden. Durch dieses angenehme User-Interface und das sehr einfache Navigieren mittels eingebetteten *Links* konnten sich viele Netzbenutzer für das WWW begeistern.

1.3 Fazit

Die Geschichte des Internet könnte sehr gut mit dem Untertitel "Eine amerikanische Erfolgsstory“versehen werden. Ebensogut würde passen: "Vom Kriegsspiel zur vernetzten Gesellschaft“, denn obwohl es müßig ist, bei geschichtlichen Entwicklungen die was-wäre-wenn-Frage zu stellen, drängt diese sich einem hier fast auf: Wie weit wären wir mit unserer vernetzten Gesellschaft, ohne die Forschungsaktivitäten des amerikanischen Verteidigungsministeriums in den 60er Jahren?

[5] heute meist möglich mit Programmen wie telnet oder rlogin

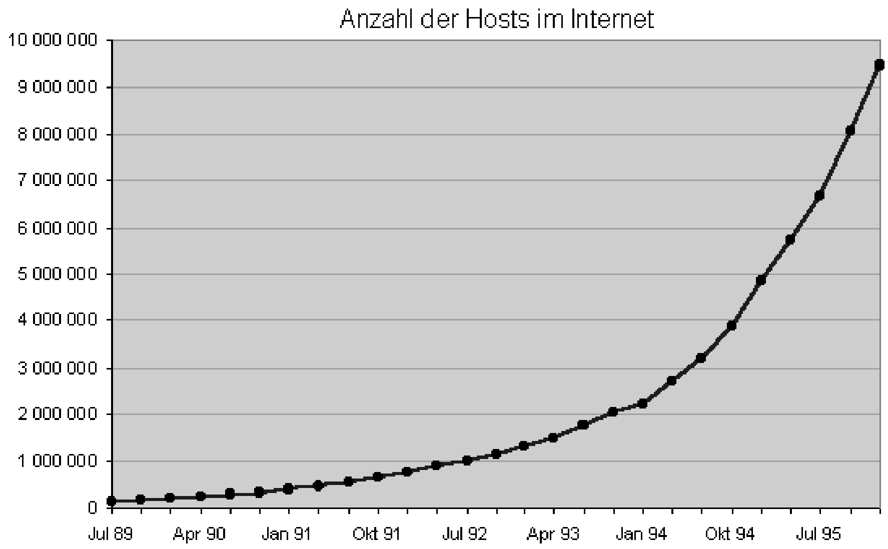


Abbildung 1.2: *Anzahl der Hosts im Internet*

Wie geht es weiter? In der Zukunft wird das Internet für die Öffentlichkeit wohl nur aus dem World Wide Web bestehen, das sich weiter kommerzialisieren wird, und gewissermaßen mit der Entwicklung zum Privatfernsehen verglichen werden kann. Echtzeit-Videoübertragung gibts ja schon...

1.4 Statistiken

Wie aus den Abb. 1.2 und 1.3 hervorgeht, ist im Internet bis zum Stand Januar '96 immer noch ein exponentielles Wachstum, sowohl bei der Anzahl de Hosts, als auch bei der Anzahl der Netwerke und Domains festzustellen.

Interessant ist auch Abb. 1.4. Sie zeigt den weltweiten Verbreitungsgrad der wichtigsten Computernetze. Dabei ist Unix-to-Unix-Copy in den meisten Ländern verfügbar, aber auch hier zeigt das Internet das stärkste Wachstum.

1.5 Abkürzungen

ARPA	Advanced Research Projects Agency
CERN	European Center for Particle Research
DARPA	Defense Advanced Research Projects Agency
DCA	Defense Communication Agency
DDN	Defense Data Network
DISA	Defense Information System Agency

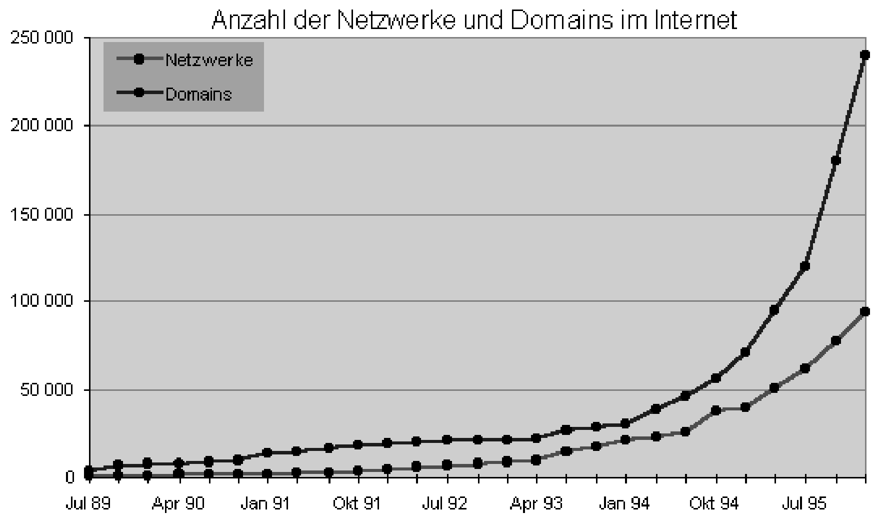


Abbildung 1.3: *Anzahl der Netzwerke und Domains im Internet*

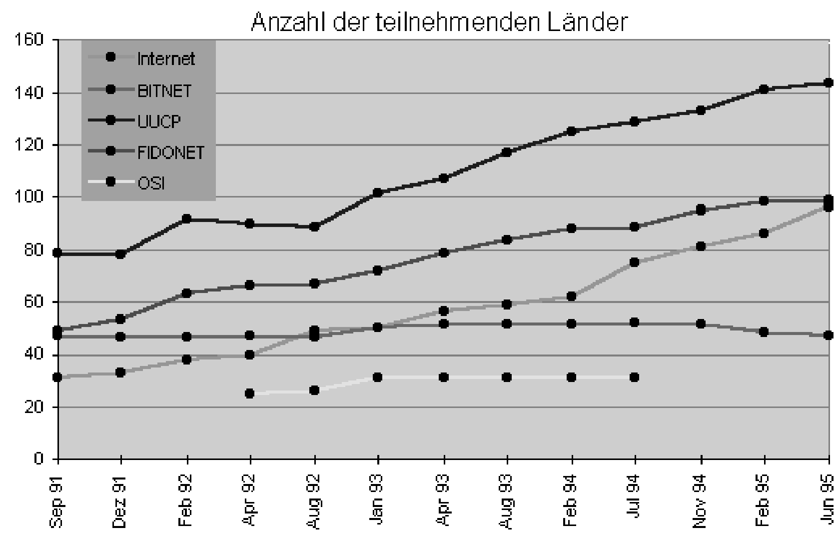


Abbildung 1.4: *Anzahl der teilnehmenden Länder der wichtigsten Computernetze*

DNS	Domain Name Server
DoD	Department of Defense
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
http	hypertext transfer protocol
IAB	Internet ativities Board
ICCB	Internet Configuration Control Board
ICCC	International Conference on Computer Communication
IMP	Interface Message Processor
INWG	International Network Working Group
IP	Internet Protocol
ISOC	Internet Society
MIT	Massachuasetts Institute of Technology
NCP	Network Control Protocol
NCSA	National Center for Supercomputing Applications
NPL	National Physical Laboratory
NSF	National Science Foundation
RCP	Reseau Communication par Paquet
TCP	Transmission Control Protocol
TIP	Terminal Interface Processor
UCLA	University of California, Los Angeles
UN	United Nations
UUCP	Unix to Unix Copy
WAIS	Wide Area Information Service
WWW	World Wide Web

1.6 URL's

ARPA	http://www.arpa.mil
CERN	http://www.cern.ch
DARPA	http://www.darpa.mil
DoD	http://dtic.dla.mil
ISOC	http://www.isoc.org
MCI	http://www.gov.mci.net
MERIT	http://nic.merit.edu
MIT	http://web.mit.edu
NCSA	http://www.ncsa.edu
NPL	http://www.npl.co.uk
NSF	http://www.nsf.gov
UCLA	http://www.ucla.edu

Kapitel 2

Information Retrieval

Christian Voigtländer

Übersicht

Ein Information Retrieval System (IRS) hat die Speicherung, die Repräsentation und den Zugriff auf Informationen (Dokumente) zum Gegenstand. Man richtet an ein IRS ein semantisch beschriebenes Suchkriterium (Anfrage), die das System mit einer Liste von Dokumenten beantwortet. In dieser Arbeit werden die Verfahren der Listenerstellung und der Bewertung der Effizienz dieser Verfahren einführend behandelt. Es werden stellvertretend ein Text Retrieval und ein Probalistisches Retrievalverfahren untersucht. Abschließend werden Kriterien zur Bewertung von IRS wie Recall/Precision und weitere Kennziffern vorgestellt.

2.1 Einleitung

Ein IRS besteht zunächst einmal aus einer ungeordneten Menge von Informationsobjekten. Nun wird eine Anfrage mittels einer Anfragesprache formuliert. Dann werden die Informationsobjekte mit einer Funktion bewertet, inwieweit sie der Anfrage genügen. Diese Funktion wird als *Retrieval Status Value (RSV)* bezeichnet. Sie liefert einen reelen Wert. Je höher der Wert ist, desto mehr erfüllt ein gespeichertes Objekt die Anfrage. Also können die vorhandenen Objekte anhand ihres RSV absteigend in einer Liste angeordnet werden, die dann dem Nutzer als Ergebnis präsentiert wird.

Das bei weitem am weitesten entwickelte Gebiet ist das Information Retrieval mit verschiedenen Texten. Dann heißt Information Retrieval auch *Text Retrieval*. Im Seminarzusammenhang gewinnen die Information Agents besondere Bedeutung. Ein Information Agent, also ein Programm, sammelt selbständig Informationen über ein gegebenes Thema. Dabei steht er vor der Aufgabe, aus einer auf einem Server vorhandenen Menge an HTML-Dokumenten diejenigen herauszufiltern, die am Besten zu seinem Thema passen, um die darin enthaltenen Referenzen und Verweise auf weitere Dokumente zur weiteren Informationsgewinnung zu verwenden.

Im Folgenden soll ein einfaches Textretrievalsystem vorgestellt und die grundlegende Arbeitsweise verdeutlicht werden.

2.2 Text Retrieval

2.2.1 Retrieval Status Value

Wir wollen zunächst einige grundlegende Probleme und Lösungsansätze am Beispiel eines einfachen Information Retrieval System (IRS) studieren. Als erstes wollen wir die Struktur eines IRS einfach näher charakterisieren. Es enthält $d_0, \ldots, d_{n-1}$ Informationsobjekte. Diese werden in der Objektsammlung $D = \{d_0, \ldots, d_{n-1}\}$ zusammengefaßt. Weiterhin existiert eine Anfrage q, die den Informationsbedarf des Nutzers darstellt. Die Menge aller möglichen Anfragen sei Q.

Wie schon eingangs erwähnt, besteht die Aufgabe eines IRS nun darin, aus der Menge D diejenigen Objekte zu isolieren, die die Anfrage q genügen. Dieses „Genügen“ wird mit einer reelen Zahl gemessen. Diese Zahl heißt *Retrieval Status Value*. Im weiteren wollen wir den Retrieval Status Value eines Objekts d_j unter der Anfrage q mit $RSV(q, d_j)$ bezeichnen. Wenn nun ein Dokument einer Anfrage genügt, wird d_j auch als relevantes Objekt, anderenfalls als irrelevant bezeichnet (Siehe auch 2.2.4).

Der RSV wird von der Retrieval Methode bestimmt. Diese besteht aus einer Indexierungsfunktion und einer Retrievalfunktion. Die Indexierungsfunktion weist jeder Anfrage $q \in Q$ einen Deskriptionsvektor $\vec{q} \in R^m$ und jedem gespeicherten Objekt $d_j \in D$ einen Deskriptionsvektor $\vec{d_j} \in R^m$ zu. Dabei ist mit m die Anzahl der im Retrievalvorgang zu bewertenden Objekteigenschaften gegeben, wie in Abbildung 2.1. Die Retrievalfunktion bildet formal dargestellt Paare von Deskriptionsvektoren auf reele Zahlen ab:

$$\rho : R^m \times R^m \to R, (\vec{x}, \vec{y}) \mapsto \rho(\vec{x}, \vec{y}). \tag{2.1}$$

Wenn also zwei Deskriptoren $\vec{q}, \vec{d_j}$ gegeben sind, dann wird der RSV be-

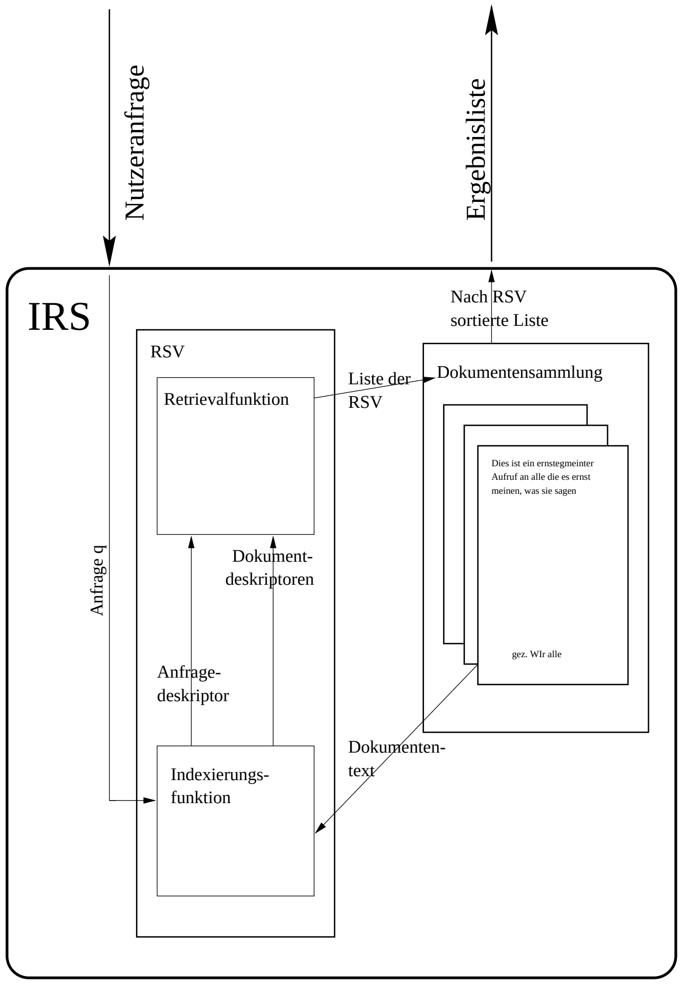


Abbildung 2.1: Ein einfaches Textretrievalsystem

stimmt durch

$$RSV(q, d_j) := \rho(\vec{q}, \vec{d_j}). \tag{2.2}$$

Die Schrittfolge, die bei einer Auswertung einer Anfrage q durchlaufen wird, läßt sich wie folgt beschreiben:

1. Berechnung der RSV's aller vorhandenen Objekte

2. Bildung einer absteigend sorierten Liste der Objekte

3. Präsentation eines genügend großen Listenteils

Diese Schrittfolge wird sich so nicht in Implementierungen von IRS wiederfinden, da es meist unsinnig ist, alle RSV zu errechnen oder die Hard/Software-

Plattformen Änderungen verlangen. Nichtsdestoweniger stellt diese Abfolge eine einfache und relativ wirklichtskeitsnahe Abbildung der Realitäten dar.

2.2.2 Berechnung des Retrieval Status Value

Nun wollen wir eine einfache Methode zur Berechnung eines RSV näher studieren. Dabei sei $\Phi = \{t_0, \ldots, t_{m-1}\}$ eine feste Menge von Eigenschaften, also *Terme* die in $d_0, \ldots, d_{n-1}$ auftreten können. Da wir ein einfaches Text Retrieval System als Beispiel gewählt haben, sind diese Eigenschaften hier Zeichenketten in Großbuchstaben: $t_i =< t_i[0], \ldots, t_i[k_i - 1] > \forall 0 \leq i < m$. Jedes Vorkommen eines Wortes in einem Objekt nennt man einen *Token*. Also enthält jedes Objekt mehr Token als Terms.

Im **Beispielsatz (1)**

> In diesem Fall legt der Autor das Layout des Textes bei der interaktiven Eingabe fest.

kommen 15 Token vor, aber nur 14 Terme. Das Wort „der" erschien doppelt. Formaler betrachtet wird ein token durch ein Prädikat beschrieben.
Sei $< d_j[0], \ldots, d_j[l_j - 1] >$ ein Textstring. Dann zeigt das Prädikat $token_0$ (i, j, p, l)[1] an, ob der Substring $< d_j[p], \ldots, d_j[p + l - 1] >$ eine Instanz des Terms t_i ist. Weiterhin bezeichne $tf(t_i, d_j)$ die term frequency, d.h. die Anzahl des Auftretens von t_i im Dokument d_j.

$$tf(t_i, d_j) :=| \{(p, l) \mid token_0(i, j, p, l)\} | \tag{2.3}$$

Und $df(t_i)$, genannt item (document) frequency, gibt die Anzahl der Dokumente an, die t_i enthalten:

$$df(t_i) :=| \{d_j \in D \mid tf(t_i, d_j) > 0\} | \tag{2.4}$$

Beispiel:
d_1 =Beispielsatz(1), d_2 =<DER SPION DER AUS DER KÄLTE KAM>
$t_0 =< PAPA >$ und $t_1 =< DER >$.
Es ergeben sich:
$tf(t_0, d_{\{1,2\}}) = 0$, $tf(t_1, d_1) = 2$, $tf(t_1, d_2) = 3$
$df(t_0) = 0$, $df(t_1) = 2$.

Man nimmt allgemein an, daß ein Begriff bedeutsam für die Charakterisierung eines Dokumentes ist, wenn der Begriff im Dokument d_j selber häufig vorkommt aber in der Dokumentensammlung D selten. Also ist $tf(t_i, d_j)$ proportional (Gleichung 2.3) zum Vorkommen des Begriffs t_i definiert und

[1] indiziert, weil später noch abweichend ein weiterer *token*-Ausdruck definiert wird

die inverse item frequency mittels der document frequency umgekehrt proportional:

$$idf(t_i) := \log \frac{n+1}{df(t_i)+1}. \tag{2.5}$$

Nun erzeugt die Indizierungsfunktion für die Anfrage q und die gespeicherten Objekte d_j Deskriptoren, die die folgende Gestalt haben:

$$\begin{aligned} \vec{d_j} &:= (a_{o,j}, \ldots, a_{m-1,j})^T & (2.6)\\ \vec{q} &:= (b_0, \ldots, b_{m-1})^T & (2.7) \end{aligned}$$

Wie schon festgestellt, soll die Komponente eines solchen Vektors wertmäßig groß sein, wenn die entspechende feature frequency groß und die item frequency klein sind. Damit bietet sich folgende Definition an:

$$\begin{aligned} a_{i,j} &:= tf(t_i, d_j) + idf(t_i) & (2.8)\\ b_i &:= tf(t_i, q) + idf(t_i). & (2.9) \end{aligned}$$

Als sinnvolle Retrievalfunktion hat sich die cos - Funktion[2] erwiesen:

$$\rho_0(\vec{q}, \vec{d}j) := \frac{\vec{q}^T \vec{d_j}}{\sqrt{\vec{q}^T \vec{q}}\sqrt{\vec{d}_j^T \vec{d_j}}} \tag{2.10}$$

Die Indizierungsfunktion eingesetzt, ergibt dann den RSV:

$$RSV_0(q, d_j) := \frac{\sum_{i=0}^{m-1} a_{i,j} b_j}{\sqrt{\sum_{i=0}^{m-1} a_{i,j}^2}\sqrt{\sum_{i=0}^{m-1} b_i^2}} \tag{2.11}$$

Der RSV entsteht also, indem der cosinus des Winkels zwischen dem Deskriptors der Anfrage und dem Deskriptor des gerade bearbeiteten Dokuments errechnet wird. Ist der Winkel klein, also das Dokument ist für die Anfrage relevant, wird der RSV des Dokuments nahe an 1 liegen. Ist ein Dokument für die Anfrage irrelevant, werden sich die Deskriptoren stark unterscheiden und der Winkel zwischen den Deskriptionsvektoren wird groß sein und damit einen kleinen RSV erzeugen.

Diese Funktion ist natürlich ein sehr einfacher Vertreter ihrer Klasse. Sie wird um so komplexer werden, je mehr man versucht, sie auf den Bereich zu spezialisieren, aus dem die zu bewertenden Objekte stammen. Weitere Verbesserungen können durch Relevance Feedback (siehe 2.2.4). erreicht werden.

[2]cos-Funktion zwischen zwei beliebigen n-dimensionalen reelen Vektoren

2.2.3 Stopwords und Word Reduction

In unserem IRS besteht die Feature-Menge Φ aus einzelnen Worten. Da aber nicht alle Worte denselben Informationsgehalt haben, kann man z.B. Füll- und Bindewörter aus den Objekten entfernen, ohne daß sich ein Informationsverlust einstellt.
Im **Beispiel**

> Das Original ist in einer verständlichen Sprache geschrieben; ich habe mich bemüht, das auch für die deutschen Leser zu gewährleisten, ohne daß dadurch die Präzsion der Darstellung leiden sollte.

sind die unterstrichenen Wörter sicher ohne Informationsgehalt. Ein Verfahren, solche Wörter zu erkennen, bedient sich der sog. *collectionfrequency*:

$$cf(t_i) := \sum_{j=0}^{n-1} tf(t_i, d_j) \tag{2.12}$$

Die entfernbaren Worte erkennt man an einem hohen cf-Wert. Wenn man also diese Worte entfernt, verändert man des RSV des Objektes nicht, weil sich mit hoher collection frequency auch eine hohe item frequency einstellt. Damit verschwindet aber die inverse item frequency und so bleibt der RSV nahezu unverändert, wenn man die Worte löscht. Diese Worte werden in einer Liste gesammelt. Diese heißt auch *stoplist* (Beispiel in [2]). Das Löschen dieser Worte verringert also laut unserem theoretischeren Ansatz die zu betrachtende Vektorlänge m der Deskriptoren. Es hat sich gezeigt, daß zwischen 30 und 50% der vorkommenden Worte stop words sind (Grundlage: englischer Text).

Nun stellt sich aber noch ein weiteres Problem: alle verbleibenden Worte können trotzdem noch mehrfach auftreten; mit verschiedenen Endungen nämlich. Seien zur Illustration dieses Sachverhalts q und d_j die beiden folgenden Objekte :

Query Item q : Mich interressiert, ob ein Befehl eckige oder geschwungene Klammern haben kann.

Stored Item d_j : Manche Befehle haben Parameter, die zwischen geschwungenen Klammern angegeben werden müssen. Manche Befehle haben Parameter, die wegelassen oder zwischen eckigen Klammern angeben werden können. Manche Befehle haben Varianten, die durch das Hinzufügen eines Sterns an den Befehlsnamen unterschieden werden.

Bei diesem Beispiel werden einfache Algorithmen die Verwandtschaft von eckige - eckigen, geschwungene - geschwungenen und Befehl - Befehle - Befehlsnamen nicht erkennen. Um solche Beziehungen zu erkennen und zu nutzen hat man Reduzierungsmethoden (reduction methods) entwickelt. Diese reduzieren die eingegebenen Wörter auf einen Stamm, der im Allgemeinen nicht mit dem grammatikalischen Wortstamm übereinstimmen muß. Im Folgenden wollen wir uns einen solchen Allgorithmus näher ansehen. Dieser Algorithmus würde die o.g. Worte wie folgt umwandeln:

ECKIGE,ECKIGEN $\mapsto$ ECK
GESCHWUNGENEN, GESCHWUNGENE $\mapsto$ GESCHWUNGEN
BEFEHLE, BEFEHLSNAME $\mapsto$ BEFEHL.

Sei nun $\Psi = \{s_0, \ldots, s_{N-1}\}$ die mit einem word-reduction- Algorithmus behandelte Feature-Menge Φ. Der Reduktionsalgorithmus wird also durch

$$r : \Phi \to \Psi, t_i \mapsto s_h \tag{2.13}$$

beschrieben. Wenn also die Feature-Menge Ψ anstelle von Φ verwendet werden soll, dann müssen wir unsere bisherigen Defintionen an diese Forderung anpassen:

$$token_1(h, j, p, l) := \exists i(token_0(i, j, p, l) \wedge (r(t_i) = s_h)) \tag{2.14}$$

Basierend auf dieser Definition können wir nun die feature frequency $tf(s_h, d_j)$, die item frequency $df(s_h)$ und die inverse item frequency $idf(s_h)$ bilden, indem wir die schon vorhandenen Definitionen von $tf(t_i, d_j)$, $df(t_i)$ und $idf(t_i)$ sinngemäß verändern. So erhalten wir andere Deskriptionsvektoren:

$$\vec{d_j} := (u_{o,j}, \ldots, u_{N-1,j})^T, \tag{2.15}$$
$$\vec{q} := (v_0, \ldots, v_{N-1})^T \tag{2.16}$$

mit wertmäßig anderen aber vom Rechenweg her bekannten Komponenten:

$$u_{h,j} := tf(s_h, d_j) + idf(s_h), \tag{2.17}$$
$$v_h := tf(s_h, q) + idf(s_h) \tag{2.18}$$

Dabei entsteht natürlich auch ein neuer RSV:

$$RSV_1(q, d_j) = \frac{\sum_{h=0}^{N-1} u_{h,j} v_h}{\sqrt{\sum_{h=0}^{N-1} u_h^2}\sqrt{\sum_{h=0}^{N-1} v_h^2}} \tag{2.19}$$

Als eine Schlußfolgerung läßt sich feststellen, daß die verteilten Gewichte durch Reduktionsalgorithmen vergrößert werden. Zusätzlich wird sich die Anzahl gemeinsamer features zwischen zwei beliebigen Objekten vergrößern. Als Nachteil eines Reduktionsalgorithmus ist festzuhalten, daß auch die RSV irrelevanter Objekte erhöht werden. Es wird also die Qualiät des Retrievalvorgangs verändert. Ob tatsächlich eine qualitative Effizienzverbesserung stattfindet, hängt von den behandelten Daten ab. Es sind spezielle Beispiele bekannt, in denen die Qualtität der erzeugten Listen verschlechtert wird. Es ist aber eine Verbesserung der Reaktionszeit, also einer quantitativen Größe zu erwarten, weil ähnlich zu stop-Listen ein Verkleinern des Suchraumes stattfindet.

Man unterscheidet zwei Vorgehensweisen bei word-reduction-Algorithmen: wörterbuchgestützte und Regelbasierte Reduktion. Beim wörterbuch gestützten Reduzieren wird mittels einer Liste (Wörterbuch) der zum Wort gehörige Stamm nachgesehen. Der im Folgenden behandelte Algorithmus von Porter (siehe [6]) ist regelbasiert und bezieht sich auf englischen Text. Auf deutschen Texten arbeitende Reduzierungsalgorithmen sind wesentlich komplizierter und seien hier nur am Rande erwähnt.

Sei also ein Konsonant durch c bezeichnet und eine nichtleere Sequenz $ccc\ldots$ durch C. Analog ist dann natürlich eine nichtleere Sequenz $vvv\ldots$ von Vokalen bezeichnet durch V. Jedes Wort hat dann eine der Formen:

$$CVCV\ldots C,$$
$$CVCV\ldots V,$$
$$VCVC\ldots C,$$
$$VCVC\ldots V.$$

Noch kompakter zusammengefaßt ergibt sich:

$$[C](VC)^m[V]$$

wobei eckige Klammern optionales Auftreten darstellen, während die runden Artgenossen m-maliges Erscheinen ($m \geq 0$) bedeuten. Im Falle $m = 0$ ist ein leerer String ϵ einzusetzen. Einige Beispiele:

m = 0	ϵ, TR, EE, TREE
m = 1	TROUBLE, OATS, TREES, IVY
m = 2	TROUBLES, PRIVATE, OATEN, ORRERY

Die Regeln für die Entfernung einer Endung haben folgende Struktur:

$$(condition)S_1 \rightarrow S_2$$

Diese Notation bedeutet, daß bei Erfüllung der Bedingung *condition* durch den Wortstamm vor S_1, S_1 durch S_2 ersetzt wird. So ist folgendes Beispiel denkbar:

$$(m > 0)EMENT \rightarrow \epsilon$$

Bei Anwendung der Regel auf REPLACEMENT würde REPLAC übrigbleiben, da m = 2 hier gilt. Die Menge der angewendeten Bedingungen wird mit jedem Schritt des Algorithmus ausgetauscht. In jedem Schritt des Algorithmus wird eine Regel angewendet. Wenn mehrere Regeln anwendbar sind, wird die angewandt, die die längere Endung erfaßt. Porters Algorithmus besteht aus 6 Schritten, die alle ihre jeweiligen Regeln verwenden. Es werden Reduktionsleistungen erreicht, die den Eingabetext um ein Drittel reduzieren und damit eine entscheidende Verkürzung der Reaktionszeiten bewirken.

2.2.4 Neuausrichten der Suche: Relevance Feedback

Relevance Feedback bezeichnet einen Mechanismus zur automatischen Umformulierung der Anfrage. Hierbei wird die Situation vorrausgesetzt, daß der Nutzer bereits auf eine erste Anfrage hin eine sortierte Liste präsentiert bekommen hat, in der er die ersten i Objekte $d_0, \ldots, d_{i-1}$ inspiziert hat. Dabei hat er sie bereits 2 Mengen zugeordnet: der Menge der relevanten Objekte D_i^{rel} oder der Menge der relevanten Objekte D_i^{non}. So wird ein neuer Deskriptor $\vec{q'}$ der mit Hilfe der Relevanzinformationen in q' veränderten Anfrage q errechnet:

$$\vec{q'} = \vec{q} + \sum_{d_j \in D_i^{rel}} \vec{d_j} - \vec{d_w} \tag{2.20}$$

wobei

$$w := min\{h \mid (0 \leq h < i) \wedge (d_{p(h)} \in D_i^{non})\} \tag{2.21}$$

Der Deskriptor der neuen Anfrage wird also dadurch errechnet, indem die Deskriptoren aller relevanten Objekte zum Anfragedeskriptor addiert der Deskriptor des höchstrangigsten irrelevanten Objekts abgezogen wird.
Beispiel:

	t_1	t_2	t_3	t_4	t_5
$\vec{q} = ($	5,	0,	3,	0,	1)
$\vec{d_1} = ($	2,	1,	2,	0,	0)
$\vec{d_2} = ($	1,	0,	0,	0,	2)
$\vec{d_3} = ($	0,	2,	3,	1,	3)

Sei $w = 3$, also d_3 das erste irrelevante Dokument. Dann ist

$$\vec{q'} = \vec{q} + ((2,1,2,0,0)^T + (1,0,0,0,2)^T) - (0,2,3,1,3)^T$$

Und damit die neue modifizierte Anfrage q':

$$\vec{\mathbf{q}}' = (5,0,3,0,1)^T + (3,-1,-1,-1,-1)^T = \mathbf{(8,-1,2,-1,0)^T}$$

So erheblich wir im Beispiel wird sich q' in der Praxis meist nicht verändern. Mittels einer in 2.4 erklärten Bewertungsmethode läßt sich ein Effektivitätsgewinn von 90% beim Nachweis der relevanten Dokumente erkennen, d.h. man bekommt die relevanten Dokumente schneller angezeigt. Vergleicht man die neue Anfrage $\vec{q'}$ mit der alten $\vec{q}$, so hat $\vec{q'}$ beträchtlich weniger von Null verschiedene Komponenten. Das beeinflußt deshalb die Effizienz, weil es bedeutet, daß die vorhandenen Deskriptoren besser ausgenutzt werden.

2.3 Probalistisches Retrieval

2.3.1 Das Probalistic Ranking Principle - PRP

Beim Probalistischen Retrieval werden die RSV durch eine wahrscheinlichkeitstheoretische Rechnung ermittelt. Grundsätzlich gilt also:

$$RSV(q, d_j) = f(P(R \mid q, d_j)) + c_q \tag{2.22}$$

Diese Tatsache führt uns zum sogenannten *Probalistic Ranking Principle* (auch PRP):

> Wenn ein IRS eine geordnete Liste von Dokumenten mittels der Schätzung des Nutzens über einen wahrscheinlichkeitstheoretische Ähnlichkeit zwischen Anfrage und Dokumentensammlung erstellt, wird diese die bestmögliche Schätzung der Ähnlichkeit auf der Grundlage der zur Verfügung gestellten Daten sein.

Das wirklich wichtige am PRP ist die Tatsache, daß es mathematisch nachgewiesen werden kann und so den gesamten Retrievalprozeß theoretisch beherrschbarer macht.

2.3.2 Gewichten von Deskriptoren

Die probalistische Gewichtung von Deskriptoren wird auch *Binary Independence Retrieval* genannt und in [9] erstmals vorgestellt. Wir nutzen hier die Definitionen von Fuhr [3] über Wahrscheinlichkeitsräume. Aus dem Kapitel 2.3.1 geht hervor, daß die Liste mit Hilfe der Wahrscheinlichkeiten $P(R \mid q, d_j)$ absteigend sortiert werden kann. Um diese Wahrscheinlichkeiten zu bestimmen, werden einige *Annahmen* getroffen. Es wird vorrausgesetzt, daß zu jedem Donkument d_j eine Menge von Deskriptoren $\Phi(d_j)$ und zu jeder Anfrage q ebenfalls $\Phi(q)$ mittels einer Bijektion bestimmbar ist. Als eine Konsequenz hat jedes Ereignis $E(d_j)$ eine Entsprechung $E(\Phi(d_j))$ und es folgt

$$P(R \mid q, d_j) = P(R \mid q, \Phi(d_j)). \tag{2.23}$$

Um die rechte Seite der Gleichung weiter aufzuspalten und so zu bestimmen, nutzt man die Regel von *Bayes*:

$$P(B \mid A) = \frac{P(B) * P(A \mid B)}{P(A)}.$$

Damit kann man die Wahrscheinlichkeit $P(R \mid q, \Phi(d_j)$ mit $A = \Phi(d_j) \mid q$ und $B = R \mid q$ als

$$P(R \mid q, d_j) = \frac{P(\Phi(d_j) \mid R, q) * P(R \mid q)}{P(\Phi(d_j) \mid q)} \tag{2.24}$$

schreiben. Dabei ist $P(\Phi(d_j) \mid R, q)$ die Wahrscheinlichkeit, daß ein Dokument die Deskriptormenge $\Phi(d_j)$ hat unter der Bedingung, daß es relevant für q ist. Weiterhin bezeichnen $P(R \mid q)$ die Wahrscheinlichkeit, daß ein Dokument überhaupt relevant für q ist und $P(\Phi(d_j) \mid q)$, daß ein Dokument im System die Deskriptorenmenge $\Phi(d_j)$ im Bezug auf q hat. Zu bestimmen sind also noch die Werte $P(\Phi(d_j) \mid q)$ und $P(\Phi(d_j) \mid R, q)$. Dazu muß noch die Unabhängigkeit von den Ereignissen t_i und $\neg t_i$ vorrausgesetzt werden. Da diese in Wirklichkeit aber nicht mathematisch unabhängig seien werden, schleicht sich hier eine Ungenauigkeit in die Rechnung ein. Nach Einsetzen ergibt sich dann:

$$P(R \mid q, d_j) = c_q * \prod_{t_i \in \Phi(d_j)} \frac{p_i(1 - p_i)}{q_i(1 - q_i)}. \tag{2.25}$$

Dabei soll c_q eine anfrageabhängige Konstante sein. In der obigen Formel fehlen aber nun noch die Werte der Wahrscheinlichkeiten p_i, q_i. Diese werden über die Relevanzmengen D_i^{rel}, D_i^{non} geschätzt. Wie in 2.2.1 schon gesehen, errechnen wir den RSV über die Retrievalfunktion angewandt auf die Deskriptoren, also $\rho(\vec{q}, \vec{d_j}) = \vec{q}^T \vec{d_j}$. So erhält man

$$RSV_{BIR}(q, d_j) = \sum_{i=0}^{m-1} a_{i,j} b_i \tag{2.26}$$

Dabei hat sich folgende Definition der $a_{i,j}$ und bj als nützlich erwiesen:

$$a_{i,j} := \begin{cases} 1, & \text{falls } t_i \in \Phi(d_j); \\ 0, & \text{sonst.} \end{cases} \tag{2.27}$$

$$b_i := \begin{cases} log\frac{p_i(1-p_i)}{q_i(1-q_i}, & \text{falls } t_i \in \Phi(d_j); \\ 0, & \text{sonst.} \end{cases} \tag{2.28}$$

Nach Einsetzen der Komponentendefintionen 2.27 in den RSV_{BIR} erkennt man, daß gilt $RSV_{BIR}(q, d_j) := log(P(R \mid q, d_j)) - log(c_q)$. Diese Retrievalfunktion wird sicher die Bedingung 2.22 erfüllen. Sie ist weiterhin durch das Probalistic Ranking Principle theoretisch abgesichert. Die Güte dieser Retrievalfunktion wird in der Praxis sicher davon abhängen, wie gut die Parameter q_i, p_i gschätzt werden. Eine weitere Methode, die gesuchten Wahrscheinlichkeiten zu schätzen, stellt das *Binary Independence Indexing* - Modell dar (siehe [3]). Die Bestimmung der gesuchten Werte kann auf so vielfältige Art und Weise erfolgen, wie es Problemstellungen gibt. Hier sollte nur ein kurzer Einblick in die grobe Funktionsweise gegeben werden. Es werden sich sicher Unterschiede in der qualitativen (Güte der erzeugten Liste) und quantativen (Reaktionszeit) Effizienz der verschiedenen Grundalgorithmen bei verschiedenen Anwendungsgebieten ergeben. Mit der Messung dieser Unterschiede im Algorithmenverhaltens soll sich das folgende Kapitel beschäftigen.

2.4 Bewerten von IRS

2.4.1 Beeinflußende Faktoren

In der IR-Forschung existiert eine lange Tradition der experimentellen Erkenntnisgewinnung. Hier seien vor allem die Cranfield-Experimente in den Sechzigern ([1],[4]) und die SMART- Versuche ([7], [8]) in den frühen Siebzigern erwähnt. Diese Experimente brachten wichtige Erkenntnisse, ob und wie man IRS bezüglich ihrer Nützlichkeit bewerten kann. Ob ein System sich als gut erweist oder nicht hängt in erster Linie von folgenden Faktoren ab:

1. Die Effektivität des Retrievalvorgangs: Wie nützlich ist die vom Rechner erzeugte Information?
2. Die Daten: Enthalten sie die verlangten Dokumente?
3. Die Anfrage-Sprache: Wie kompliziert ist eine Anfrageformulierung?
4. Die Anzeige der Ausgabe: Wie gut sind die Daten präsentiert wurden?
5. Die Antwortzeit: Wie lange dauert der gesamte Vorgang?
6. Die Kosten: Wieviel kostet das betrachtete System?

Es ist aber im Allgemeinen nicht möglich, absolute Zahlen für jeden Faktor anzugeben. Deshalb weicht man auf relative Aussagen aus, die z.B. die Retrievaleffektivität zweier IRS auf einem bestimmten Themengebiet vergleichen. Die meisten Forschungsaktivitäten zielen auf die Bestimmung von Maßen für den ersten Punkt: die qualitative Effizienz des Retrievals. Einige sollen hier im weiteren eingeführt werden.

2.4.2 Recall and Precision

Diese beiden Effektivitätsmaßzahlen basieren auf Relevanzbetrachtungen. Dabei unterscheidet man drei Arten von Relevanz:

- Objektive Relevanz: Wie gut befriedigt ein Dokument die Anfrage möglichst objektiv gesehen ?
- Subjektive Relevanz: Wie gut befriedigt ein Dokument die Anfrage eines bestimmten Nutzers aus dessen subjektiver Sicht ?
- RSV: die von der Maschine errechnete Relevanz bzgl. einer Anfrage.

Die im weiteren definierten Recall/Precsion - Werte gehen davon aus, daß ein Retrieval stattgefunden hat. Somit steht also eine geordnete Liste der Dokumente zur Verfügung. Diese Liste wird mit einer vor dem Test bestimmten Liste aller in der Dokumentensammlung enthaltenen für diese Anfrage relevanten Dokumente verglichen. So läßt sich eine zahlenmäßige Feststellung treffen, wie gut die relevanten erfaßt und irrelevante Dokumente abgelehnt wurden.

Bei der Bewertung verwendet man Testmengen, in denen o.g. vorbereitete Liste schon als ein Teil enthalten sind. Diese Testmengen sind experimentell ermittelte Sammlungen von zu bewertenden Daten D_i, Testanfragen Q_0 und zu jeder Anfrage vorbereitete Relevanzlisten. Die Tabelle 2.1 zeigt einige der am meisten verwendeten Datensammlungen. Der zu ermittelnde Graph heißt nach den Werten, die ihn bestimmen Recall/Precision-Graph.
Man nimmt zunächst sinnvollerweise an, daß die beiden Mengen D_i^{rel} und

	#items	#queries	#features
medlars	1033	30	6927
cranfield	1398	225	3763
inspec	12684	84	14255

Tabelle 2.1: Charakteristika einiger Testmengen

D_i nichtleer sind. Um den Recall-Precision-Graphen auszurechnen, wird als erstes für jede Anfrage $q \in Q_0$ die Rangfolge der Dokumente (der RSV) errechnet. Dann werden n Paare von Recall-Precision-Werten bestimmt. Ein solches Paar (π_i, ρ_i) drückt also dann den Grad der Befriedigung der Nutzeranfrage aus. Sei also D_i die dem Nutzer präsentierte Liste der i ersten Einträge in der Liste. Nun mißt der Recall-Wert ρ_i wieviele der überhaupt vorhandenen relevanten Objekte wirklich in der Antwort enthalten waren. Analog dazu mißt dann der Precision-Wert π_i, die Anzahl der irrelevanten

Dokumente in der Antwortliste. In Formeln ausgedrückt:

$$\rho_i := \frac{\mid D_q^{rel} \cap D_i \mid}{\mid D_q^{rel} \mid} \tag{2.29}$$

$$\pi_i := \frac{\mid D_q^{rel} \cap D_i \mid}{\mid D_i \mid} \tag{2.30}$$

Beispiel:
Sei also $D_6 = \{d_2,d_4,d_0,d_3,d_5,d_8\}$ die inspizierte Liste. Es sei weiterhin durch die Testmenge vorgegeben $D_q^{rel} = \{d_0,d_1,d_2,d_4\}$. Dann ergeben sich durch die folgende Recall/Precision-Werte für die Anfrage q:

$$\rho_6 = \frac{\mid \{d_0, d_2, d_4\} \mid}{\mid D_q^{rel} \mid} = \frac{3}{4}; \quad \pi_6 = \frac{\mid \{d_0, d_2, d_4\} \mid}{\mid D_6 \mid} = \frac{1}{2}.$$

Die Länge der Ausgabeliste ist also frei wählbar. Somit stehen also mehrere Recall/Precision-Paare für eine Anfrage zur Auswahl. Wir nehmen den schlechtesten Fall an, und wählen π_i als maximal:

$$\Pi_q := max\{\pi_i \mid \rho_i \geq \rho\} \tag{2.31}$$

Nun sammeln wir die Werte für die einzelnen Anfragen q zu einer Kurve für die gesamte Testcollection ein und normalisieren den Wertebereich der Funktion mittels Division durch $\mid Q_0 \mid$:

$$\Pi_\rho := \frac{1}{\mid Q_0 \mid} \sum_{q \in Q_0} \Pi_q(\rho) \tag{2.32}$$

Der Graph dieser Funktion wird Recall-Precision-Graph genannt. In Abbildung 2.2 ist ein typischer Recall/Precision-Graph angegeben. Diese Kurve wird mit dem Sachgebiet, auf dem die Tests durchgeführt werden, geringfügig variieren. Die wirkliche Aussagekraft ein solchen Graphen hängt stark von der Dokumentensammlung ab. Es sind Beispiele bekannt, bei denen irreführende Interpretationen ([11], [12]) möglich waren.

Wir haben bis hier hin vorrausgesetzt, daß die generierten Listen eineindeutig sind, also alle Dokumente verschiedene RSV erhalten. Wenn dies nicht der Fall ist, wird *geschätzt*, mit welcher Wahrscheinlichkeit die einzelnen möglichen Sortierreihenfolgen auftreten können. So kann man dann Erwartungswerte für die Recall/Precision-Werte pro Anfrage ermitteln. Damit wird die Gesamtaussage weniger genau sein.

2.4.3 Usefulness Measure - Nutzenkennziffer

Dieser Abschnitt befaßt sich mit einer Definitionen dieser Maßzahl. Eine frühere Variante wurde in [10] vorgestellt. Die *usefullness measure* kann nur

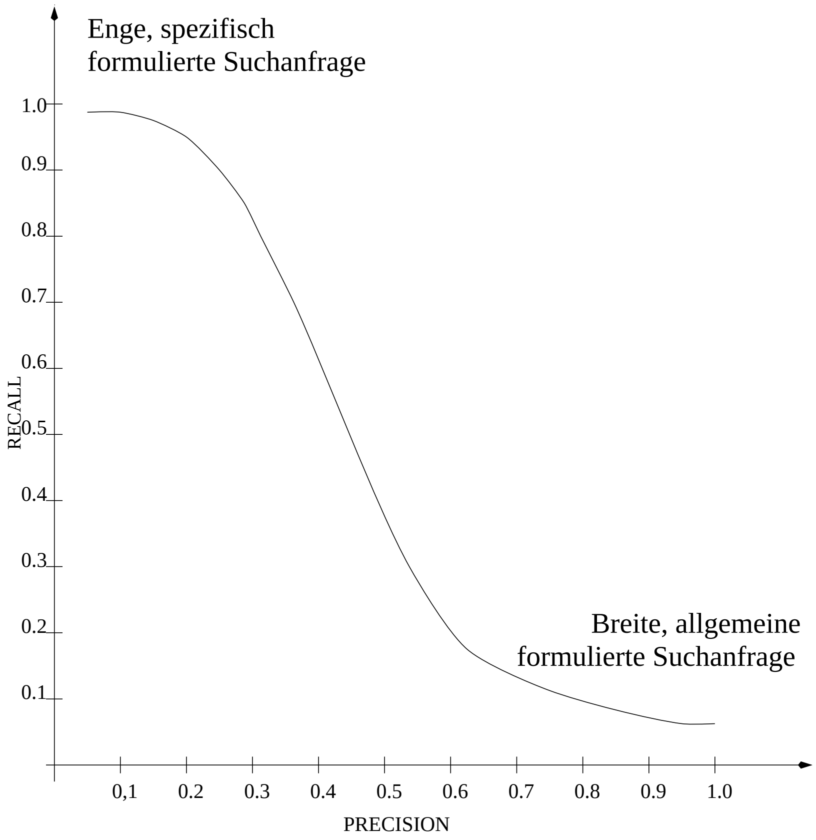


Abbildung 2.2: Typischer Recall/Precisiongraph

relative Aussagen treffen, so z. B. ob eine Retrievalmethode A besser ist als der Gegenspieler B bzgl. der Effektivität der Informationsgewinnung. Beide Methoden seien über ihre RSV gegeben, also RSV_A für Funktion A und RSV_B dementsprechend für B.

Die usefulness measure wird mit Hilfe eines Wahrscheinlichkeitsraumes bestimmt, der mehrere Nutzer und eine veränderliche Dokumentensammlung zuläßt. Aber man kann hier annehmen, daß in jedem Moment das System nur von einem Nutzer mit einer Anfrage beschäftigt wird. Sei nun $P(D, p, q, r)$ die Wahrscheinlichkeit, daß D die aktuelle Dokumentenmenge, p der gegenwärtige Nutzer, q dessen Anfrage sind und der Nutzer spezifiziert die Relevanz von höchstens $2r$ Dokumenten[3]. Gegeben sei weiterhin die vom Rechner erzeugte geordnete Liste, dann wird die Antwortmenge $R_A(D, q, r)$ des Algorithmus A wie folgt definiert:

[3] höchstens 2r folgt aus der Defintion im Folgenden

$$R_A(D,q,r) = \begin{cases} D = \{d_{j(0)}, \ldots, d_{j(r-1)}\} & \text{falls } r \leq \mid D \mid \\ D & \text{falls } r > \mid D \mid . \end{cases}$$

Die Menge $R_B(D,q,r)$ ist analog definiert. Da wir beide Algorithmen vergleichen wollen, wird unser hier betrachtetes Versuchssystem die Vereinigung beider Menge als Ergebnis liefern:

$$R(D,q,r) := R_A(D,q,r) \cup R_B(D,q,r) \tag{2.33}$$

Diese Menge wird also höchstens $2r$ Dokumente enthalten. Aus den obigen Definitionen von R_A und R_B ergibt sich, daß bei $r \leq \mid D \mid$ die Antwortmenge $R(D,q,r)$ aus wenigstens r Dokumenten und bei $r \geq \mid D \mid$ aus den in D vorhandenen Dokumenten bestehen wird. Der Nutzer wird also bei seinen Relevanzbewertungen nicht wissen, ob ein bestimmtes Dokument von A oder von B herausgesucht wurden ist.

Nun wird die *gesamte* Antwortmenge in ihrer Relevanz bewertet. Anders als bei Relevance Feedback (siehe Kapitel 2.2.4) werden *keine* binären Entscheidungen der Art relevant/irrelevant getroffen, sondern der Nutzer ordnet die in der Antwort enthaltenen Dokumente in einer geordneten Liste an. Damit erreicht man, daß die später definierten Zufallsvariablen zur Schätzung der eingangs erwähnten Wahrscheinlichkeit einfacher zu beherrschen sind. Bei der Anordnung der Dokumente in der Liste durch den Nutzer entsteht eine sog. *Präferenzrelation* $<_p$, wobei p den Nutzer identifiziert. Also bezeichnet $d <_p d'$, daß der Nutzer d' für nützlicher als d hält. Bei jeder neuen Antwortmenge wird der Nutzer nun diese Relation spezifizieren. Die Menge seiner Präferenzen wird dann so aussehen:

$$\begin{aligned} & \pi_p \quad \cap \quad R^2(D,q,r) && (2.34) \\ \text{mit} \quad & \pi_p \quad := \quad \{(d,d') \mid d <_p d'\} && (2.35) \end{aligned}$$

Die Paare in π_p repräsentieren die Nutzerpäferenzen. Hingegen stellen die in $\pi_p \cap R^2(D,q,r)$ vorhandenen Paare die explizit durch den Nutzer festgelegten Präferenzen dar. Eine Präferenz $d <_p d'$ heiße verletzt von der jeweiligen Retrievalmethode RSV, wenn $RSV(q,d) > RSV(q,d')$. Daher definieren wir Mengen, die die Präferenzpaare enthalten, die von der jeweiligen Methode nicht verletzt werden:

$$\begin{aligned} \pi_A &:= \{(d,d') \mid RSV_A(q,d) < RSV_A(q,d')\}, && (2.36) \\ \pi_B &:= \{(d,d') \mid RSV_B(q,d) < RSV_B(q,d')\}. && (2.37) \end{aligned}$$

Und weiter legen wir Zufallsvariablen fest, die die Wahrscheinlichkeiten der verschiedenen Verletzungsmöglichkeiten schätzen sollen:

$$X^+_{\{A,B\}}(D,p,q,r) \quad := \quad \frac{\mid R^2(D,q,r) \cap \pi_p \cap \pi_{\{A,B\}} \mid}{\mid R^2(D,q,r) \cap \pi_p \mid} \tag{2.38}$$

$$
\begin{aligned}
X^{-}_{\{A,B\}}(D,p,q,r) &:= \frac{\mid R^2(D,q,r) \cap \pi_p^{-1} \cap \pi_{\{A,B\}} \mid}{\mid R^2(D,q,r) \cap \pi_p \mid} && (2.39)\\
X(D,p,q,r) &:= \frac{X^{+}_{\{A,B\}} - X^{-}_{\{A,B\}}}{2} && (2.40)\\
& && (2.41)
\end{aligned}
$$

Die Zufallsvariable $X(D,p,q,r)$ bezeichnet also, daß 50% der Nutzerpräferenzen von A erfüllt werden und 50% durch A verletzt werden. Wie aus der Wahrscheinlichkeitstheorie und Statistik [5] bekannt, werden dann die tatsächlichen Versuchausgänge mit $x(D,p,q,r)$ und $y(D,p,q,r)$ bezeichnet. Jedes Ereignis ist als 4-Tupel (D_i, p_i, q_i, r_i) mit $0 \leq i < k$ mit k als der Versuchsanzahl gegeben.

Seien also k Versuche erfolgt und die Werte x_i und y_i nach den obigen Formeln errechnet. Aus den Definitionen ergibt sich, daß $x_i, y_i \in [-0.5, 0.5]$ und so $y_i - x_i \in [-1, +1]$. Die Summe der positiven Ränge w_+ wird jetzt bestimmt wie folgt:

1. Errechne alle Differenzen $y_i - x_i$ und entferne alle, die gleich Null sind.

2. Ordne die Differenzen nach ihren absoluten Werten $\mid y_i - x_i \mid$ so daß der kleinste Absolutwert der Listenerste ist. Wenn Gleichheiten der Art $\mid y_i - x_i \mid = \mid y_j - x_j \mid$ auftreten, dann weise $\mid y_i - x_i \mid$ den Durchschnittswert seiner Ränge zu.

3. Setze die Vorzeichen der absoluten Differenzen vor die jeweiligen Ränge.

4. Summiere die Differenzen alle positiven Ränge zu w_+

Die Methode A ist dann klar besser als B, wenn viele Differenzen mit positiven Vorzeichen hohe Ränge einnehmen. Das bedeutet, wenn w_+ groß ist, dann ist B klar besser als A. Es können aber nicht über die Größe von w_+ darauf geschlossen werden, ob B sehr viel oder nur ein wenig besser als A ist.

Um zu dieser Fragestellung eine Aussage treffen zu können, definieren wir die Nutzenkennzahl $w_{A,B}$ als die normalisierte Abweichung von w_+ vom Erwartungswert von W_+. Diesen Erwartungswert erhält man über die Gleichverteilung von X_i und Y_i. Die Anzahl k_0 wird nun abgeleitet von k, indem man die Anzahl der Nulldifferenzen $(y_i - x_i = 0)$ davon subtrahiert.

$$u_{A,B} = \frac{w_+ - \mu}{\mu} \tag{2.42}$$

mit

$$\mu = \frac{k_0(k_0+1)}{4} \tag{2.43}$$

Anschaulich ausgedrückt bedeutet $u_{A,B}$ wie oft im Durchschnitt die Werte y_i größer sind als die x_i. Weiter definiert man ähnlich einen statistischen Mittelwert

$$u^*_{A,B} = u_{A,B}\frac{1}{k} \mid \sum_{i=0}^{k-1}(y_i - xi) \mid \tag{2.44}$$

der die sog. *adjusted usefulness* darstellt. Dieser Wert nun sagt auch von der Größe her etwas über den Unterschied zwischen A und B. Da wir aber beim Messen der Eingangswerte in die Gleichungen sehr wahrscheinlich Fehler machen werden, kann auch eine Interpretation dieses Wertes irreführend sein. Wie man aus den Defintionen erkennt, kann er auch negativ werden, nämlich wenn A besser als B ist. Um eine o.g. Irreführung auszuschließen, prüft man die zu $u_{A,B}$ gehörende Zufallsvariable $U_{A,B}$ auf folgende Signifikanzwahrscheinlichkeit:

$$P_k(U_{A,B} \geq u_{A,B}) \tag{2.45}$$

Wenn diese Wahrscheinlichkeit niedrig ist, dann ist es unwahrscheinlich, daß B besser als A ist. Die Wahrscheinlichkeit (2.45) heißt auch Ablehnungswahrscheinlichkeit (siehe [5]) der Hypothese, daß A besser als B war. Wenn man $k \geq 20$ wählt [4], kann man aus der Statistik folgende Formel entnehmen und so die Wahrscheinlichkeit (2.45) schätzen:

$$P_k(U_{A,B} \geq u_{A,B}) \leq 1 - \Phi\left(\frac{w_+ - \mu}{\sigma}\right) \tag{2.46}$$

mit μ wie in (2.43) und

$$\sigma^2 = \frac{k_0(k_0+1)(2k_0+1)}{24} \tag{2.47}$$

$$\Phi(b) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{b} e^{-x2/2}dx \tag{2.48}$$

Im krassen Unterschied zu den Recall- und Precision-Werten ist die *usefulness*-Kennzahl auf relativen Bewertungen aufgebaut. Dabei hatte der Nutzer nur Bewertungen auf relativ kleinen Teilmengen der gesamten Dokumentensammlung durchzuführen, was um so vorteilhafter ist, wenn man sich vor Augen führt, daß diese Sammlungen in der Praxis Millionen Dokumente enthalten können. Außerdem wird die Dokumentensammlung als dynamisch vorausgesetzt und man kann sagen, wie stabil die Rechnung ist. über die in dieser Arbeit nur kurz erklärten IR-Mechanismen ergeben.

Literaturverzeichnis

[1] C. W. Cleverdon and E. M. Keen. *Factors Determining the Performance of Indexing Systems*, volume 2. Aslib Cranfield Research Project, Cranfield, England, 1966.

[4] in [5] wird erklärt, daß dann die angewendeten Tests sinnvolle Ergebnisse liefern

[2] C. Fox. A stop list for general english. In *SIGIR Forum*, volume 24:(1-2), pages 19–35, 90.

[3] N. Fuhr and C. Buckley. A probalistic learning approach for document indexing.

[4] K. Sparck Jones. Retrieval system tests 1958-1978. In Butterworths, editor, *Information Retrieval Experiment*, London, 1981. K. Sparck Jones.

[5] Helmut Wegemann Jürgen Lehn. *Einführung in die Statistik*. Teubner, Stuttgart, 1992.

[6] M. F. Porter. An algorithm for suffix stripping. In *Program*, 3, pages 130–137, 12 1980.

[7] G. Salton. *The SMART Retrieval System Experiments in Automatic Document Processing*. Prentice Hall, Englewood, Cliffs, NJ, 1971.

[8] G. Salton. A new comparison between conventional indexing (medlars) and automatic text processing (smart). In *Journal of the ASIS*, 23, pages 75–84, 2 1972.

[9] S. E. Robertson und K. Sparck-Jones. Relevance weighing of search terms. In *Journal of the ASIS*, pages 129–146, 4 1989.

[10] H. P. Frei und P. Schäuble. Determining the effectiveness of retrieval algorithms. In *Information Processing and Management*, pages 153–164, 27(2-3) 1991.

[11] H. P. Frei und P. Schäuble. The perils in interpreting recall and precision values. In N. Fuhr, editor, *Workshop Information Retrieval*, Informatik-Fachberichte 289, Darmstadt, 1991. Springer-Verlag.

[12] P. Bollmann V. V. Raghavan, G. S. Jung. A critical investigation of recall and precision as measures of retrieval system performance. In *ACM Transactions on Information Systems*, 3, pages 205–229, 7 1989.

Kapitel 3

Theorie der Lernbarkeit von Pattern und ihre Anwendbarkeit

Timna Esther Schneider

Übersicht

Die Fülle der im Internet angebotenen Informationen und Dienstleistungen nimmt fortwährend zu. Es wird daher nach Möglichkeiten gesucht, Benutzern den Umgang zu vereinfachen und ihnen Arbeit abzunehmen. Eine Möglichkeit der Arbeitserleichterung könnten lernende Patternsysteme, bzw. Daten-Eingabe-Systeme bieten. Da im Internet denkbare Anwendungen in der nachfolgenden Ausarbeitung vorgestellt werden, soll hier eine kurze Einführung in die Theorie der Lernbarkeit von Pattern gegeben werden. Diese Theorie läßt sich jedoch nicht so ohne weiteres umsetzen, deshalb werden auch die dabei auftretenden Probleme aufgezeigt.

3.1 Einführung

Anhand dieser Arbeit soll mit Bezug auf eine mögliche Anwendbarkeit in Daten-Eingabe-Systemen die Idee der Lernbarkeit von Pattern vermittelt werden.

Dabei wird die Lernbarkeit auf grammatikalische Folgerungen beschränkt, oder einfacher, auf Mustererkennungen über einer von Benutzern eingegebenen Menge von Zeichenketten. Für diese soll Schritt für Schritt eine neue Eingabe aufgenommen und mit der vorherigen Folgerung verglichen werden. Je nach Ergebnis dieses Vergleiches wird ein neuer Lösungsvorschlag entwickelt oder der alte übernommen.

Diese Art zu folgern bezeichnet man auch als "induktive Folgerung". Sie läßt sich weiterhin unterscheiden in "Folgerungen aus positiven Daten" und in "Folgerungen aus positiven und negativen Daten". Für erstere werden alle Eingaben als gültige Informationen behandelt, bei letzterer Methode dagegen können negative Daten anschaulich als Gegenbeispiel angesehen werden, d.h. diese Eingaben und alle damit zusammenhängenden Folgerungen dürfen im Ergebnis nicht vertreten sein.

Eine der Sprachenklassen die aus positiven Daten gefolgert werden kann, ist die Klasse der Patternsprachen. Ein Pattern –p– ist hierbei ein nichtleerer, endlicher String aus Konstanten und Variablen, seine Sprache –L(p)– die Menge aller Strings, die durch Ersetzen aller Variablen mit konstanten, nichtleeren Strings entsteht.

Ein wichtiges Verfahren zur Bestimmung einer solchen Sprache ist die MINL-Berechnung. Sie soll eine minimale Sprache finden, die eine gegebene, nichtleere und endliche Menge S von Strings enthält.
Ein für die geplante Anwendung ausreichender Spezialfall der MINL-Berechnung ist ℓ-MINL. Diese Berechnung findet über einer gegebenen Stringmenge S ein Pattern, das alle konstanten Teilstrings der einzelnen Strings aus S enthält, zudem auch eine mögliche Symmetrie der Variablen berücksichtigt und eine minimale, S enthaltende Sprache bildet.

Die Probleme von MINL-Berechnungen liegen in ihrer Berechenbarkeit. Werden die Arten der Pattern jedoch eingeschränkt auf reguläre und auf noncross Pattern, läßt sich dieser Spezialfall ℓ-MINL in polynomialer Zeit berechnen.

Da es Absicht dieser Ausarbeitung ist, nur die "Idee" der Lernbarkeit zu vermitteln, wird für konkrete Fragen auf die Literaturliste verwiesen.
(Quelle und Vorlage dieser Arbeit bildet das Paper von Takeshi Shinohara, $\rightarrow$ [9] im Literaturverzeichnis)

Nachfolgend eine grobe Gliederung der gestreiften Forschungsbereiche:
Induktive Folgerungen allgemein: [10]
Induktive Folgerungen von positiven Daten: [4], [5]
MINL-Berechnungen: [3]
Automatentheorie: [7] (allgemeine Einführung), [8] ,[1], [6], [2]

Kapitelübersicht

In Kapitel 3.2, "Pattern und Patternsprachen", werden Pattern, ihre Sprachen allgemein, sowie zwei spezielle, für Anwendungen ausreichende Patternarten vorgestellt und einige grundlegende Folgerungsregeln eingeführt.
Kapitel 3.3, "Allgemeine ℓ-MINL-Berechenbarkeit", behandelt die ℓ-MINL-Berechnung für allgemeine Pattern und geht auf die dabei entstehenden Komplexitätsprobleme ein.
In Kapitel 3.4, "Berechenbarkeit von ℓ-MINL für reguläre und non-cross Pattern", werden für die speziellen Patternarten Lösungsmöglichkeiten für die Berechenbarkeit von ℓ-MINL in echt polynomialer Zeit angegeben.

3.2 Pattern und Patternsprachen

In diesem Kapitel werden die Grundlagen von Pattern und Patternsprachen eingeführt, sowie einige grundlegende Folgerungsregeln vorgestellt. Im folgenden gilt:

Σ : "Menge der Konstanten",
bezeichnet eine nichtleere, endliche Menge von Symbolen mit mindestens zwei Elementen;
$X = \{x_1, x_2, \ldots\}$ "Menge der Variablen",
ist eine abzählbare, unendliche Menge von Symbolen disjunkt zu Σ: $\Sigma \cap X = \emptyset$.

Mit diesen Vereinbarungen lassen sich Pattern einfach definieren:

Definition 1
Ein Pattern ist ein beliebiger, nichtleerer String über $\Sigma \cup X$.

Darauf aufbauend erhält man

Definition 2
Die Sprache $L(p)$ eines Pattern p ist die Menge aller Strings, die man erhält, indem jede Variable von p durch einen beliebigen, nichtleeren String über Σ ersetzt wird!

Beispiel:
Sei $\Sigma = \{0, 1, 2\}$, $X = \{x_1, x_2, \ldots\}$, Σ^+ ein beliebiger, nichtleerer String über Σ;
$p = x_1 1 x_2 1 x_1 0$ wäre dann ein mögliches Pattern, durch das sich die folgende Sprache ergibt:
$L(p) = \{a1b1a0 \mid a, b \in \Sigma^+\} = \{110110, 210120, 11202110, \ldots\}$

Pattern lassen sich durch bestimmte Formen des Variablenauftretens in verschiedene Klassen einteilen. Zwei spezielle, in Kapitel 3.4 betrachtete Patternarten sind:

Definition 3
Ein Pattern p ist regulär, wenn jede Variable x_i aus p genau einmal in p vorkommt.

Definition 4
Ein Pattern p wird als non-cross bezeichnet, wenn zwischen dem am weitesten links und dem am weitesten rechts liegenden Auftreten einer Variablen x_i keine anderen Variablen vorkommen als nur x_i selbst.

Beispiel: Seien Σ und X wie oben.
Dann ist $p = 1x_1x_300x_2$ regulär, $q = 1x_1x_300x_1$ dagegen nicht regulär. Ebenso ist $p = 20x_211x_2x_20x_2x_1$ non-cross und $q = 20x_211x_1x_20$ nicht non-cross.

Bemerkung: Reguläre Pattern sind eine Untermenge von non-cross Pattern!

Auf Pattern lassen sich auch Abbildungen definieren:
Betrachtet man eine Funktion f, $f : P \rightarrow P$, wobei P die Menge aller Pattern bezeichnet, dann gilt:

- f ist eine Substitution, wenn für eine beliebige Konstante c gilt: $f(c) = c$, oder anschaulicher: Variablen werden durch Konstanten oder Variablen ersetzt.

- f heißt eine Variablenumbenennung, wenn für eine Substitution f mit $x, y \in X$ beliebig und $f(x) \in X$ gilt: $f(x) = f(y) \Rightarrow x = y$; werden also zwei beliebige Variablen mittels f auf die gleiche Variable abgebildet, so sind die Variablen gleich.

Mit Hilfe dieser Funktionen werden zwei binäre Relationen über der Patternmenge P definiert:

Definition 5
$p \equiv' q \Leftrightarrow p = f(q)$ für eine Variablenumbenennung f,
$p \leq' q \Leftrightarrow p = f(q)$ für eine Substitution f.

Vereinbarung:
Um nachfolgend f zu konkretisieren, wird die Schreibweise $[a_1/v_1, \ldots, a_n/v_n]$ eingeführt mit $v_i \in X$, $a_i \in \{\Sigma \cup X\}^+$. Angewendet auf ein Pattern bezeichnet sie die Ersetzung der Variablen v_i durch einen String bestehend aus Konstanten oder Variablen.

Beispiele zu den Relationen:
$p \equiv' q$ bzw. Äquivalenz zweier Pattern bedeutet, daß sie sich nur durch die Namen ihrer Variablen unterscheiden:
$p = 1x_1x_3010x_1 \equiv q = 1x_2x_3010x_2$, da $p = q[x_1/x_2]$.
Nicht äquivalent dagegen sind $p = x_2x_3x_2 \equiv q = x_2x_3x_1$, da zwar $p = q[x_2/x_1]$, aber nicht $p[x_1/x_2] = q$.
$p \leq' q$ bzw. p läßt sich aus q herleiten, wenn q durch Variablensubstitution in p umgewandelt werden kann, q ist demnach allgemeiner als p:
– $p = 1221 \ \leq' q = 1x_11$ durch $q[22/x_1]$
– $p = 1x_1x_1 \leq' q = 1x_1x_2$ durch $q[x_1/x_2]$
– $p = 0x_1x_1 \leq' q = 0x_2$ durch $q[x_1x_1/x_2]$
Nicht erlaubt ist also $p = 0 \leq' q = x_1x_2$, denn sonst würde eine Variable auf ein leeres Element abgebildet.

Bemerkung:
Die Relation $\leq'$ enthält noch eine weitere, interessante Aussage, denn betrachtet man die Beispiele genauer, ist die Länge des Pattern q offensichtlich kleiner oder gleich der Länge des Pattern p.
Es gilt also: Länge des Grundpattern $\leq$ Länge der ableitbaren Pattern.

Mit Hilfe obiger Relationen wurden von Angluin ([3]) Folgerungen für Patternsprachen aufgestellt und bewiesen:

Hilfssatz 1
(1) Für alle Pattern p,q gilt: $p \equiv' q \Leftrightarrow L(p) = L(q)$.
(2) Für alle Pattern p,q gilt: $p \leq' q \Rightarrow L(p) \subseteq L(q)$,
im allgemeinen gilt aber nicht: $L(p) \subseteq L(q) \Rightarrow p \leq' q$.
(3) Sind p und q Pattern mit $|p| = |q|$, dann gilt: $p \leq' q \Leftrightarrow L(p) \subseteq L(q)$.

Erklärungen zum Hilfssatz:

ad 1) "$\Rightarrow$" ist klar per Definition von $\equiv'$.

"$\Leftarrow$": Sind zwei Sprachen gleich, müssen die bildenden Pattern die gleichen Konstanten enthalten und sich die Variablen durch eine Variablenumbenennung aufeinander abbilden lassen.

ad 2) Läßt sich p aus q ableiten, dann können von q alle Strings gebildet werden, die auch p darstellen kann. Da q aber allgemeiner ist als p, enthält die von q gebildete Sprache mehr Strings als die von p gebildete: $L(p) \subseteq L(q)$.

Als Beispiel für die Ungültigkeit der Rückrichtung dieser Aussage betrachte man die Pattern: $p = 0x_1 10 x_1 x_1 1$ und $q = x_1 x_1 x_2$

es gilt zwar: $L(p) \subseteq L(q)$;

es läßt sich aber keine Ersetzung der Variablen x_1 aus q finden, so daß $p \leq' q$.

ad 3) "$\Rightarrow$" gilt nach 2).

"$\Leftarrow$": Gilt $L(p) \subseteq L(q)$ und $|p| = |q|$, dann müssen alle in q vorkommenden Konstanten auch in p vorhanden sein, aber nicht alle Konstanten aus p auch in q auftreten.

(Bei gleicher Länge sind die Möglichkeiten der Variablenpermutation bei dem Pattern mit mehr Konstanten [= weniger Variablen] geringer und $L(p) \subseteq L(q)$.)

Da die Pattern gleichlang sind, muß demnach für alle weiteren Konstanten aus p jeweils eine Variable in q enthalten sein. Ebenso muß jede Variablen in p durch eine Variablensubstitution der restlichen Variablen aus q folgerbar sein; dies ist aber gleichbedeutend mit der Definition von $\leq'$, also gilt $p \leq' q$!

3.3 Allgemeine ℓ-MINL-Berechenbarkeit

Ein Verfahren, um aus einer endlichen Stringmenge ein alle Strings beschreibendes Pattern und somit eine zugrundeliegende Patternsprache zu finden, ist die MINL-Berechnung.

Definition 6

(Angluin): MINL(S) findet zu einer gegebenen, nichtleeren und endlichen Stringmenge S ein Pattern p, so daß $S \subseteq L(p)$ ist, und es kein anderes Pattern q gibt, für das gilt $S \subseteq L(q)$ und $L(q) \subseteq L(q)$.

($L(p)$ soll also eine minimale Sprache sein, die S enthält, denn die maximale Sprache wäre trivial und für alle Strings gleich: mit $p = x$ enthält $L(p)$ alle denkbaren Strings.)

Im Hinblick auf das Thema dieser Ausarbeitung und der damit verbundenen folgenden Ausarbeitung, zusammenfassend ausgedrückt als "Lernbarkeit von Pattern und mögliche Anwendungen im Internet", ist die Menge S als eine von einem Benutzer eingegebene Menge oder Sammlung von Textstrings zu betrachten. Um dem Benutzer bei einer in der folgenden Arbeit vorgeschlagenen Anwendung eine größtmögliche Arbeitserleichterung zu gewähren, sollte eine solche MINL-Berechnung aus den gegebenen Strings ein Maximum an auftretenden konstanten Teilstrings erkennen. Ein diese Art von Pattern erkennender Spezialfall der MINL-Berechnung ist die von Angluin [3] definierte ℓ-MINL-Berechnung.

Definition 7
ℓ-MINL sucht zu einer gegebenen, nichtleeren und endlichen Stringmenge S ein Pattern p (falls eines existiert) mit maximal möglicher Länge, so daß $S \subseteq L(p)$ und es kein anderes Pattern q gibt, für das gilt: $S \subseteq L(q)$ und $L(q) \subsetneq L(p)$.

Anschaulich bedeutet die maximal mögliche Patternlänge, daß die Konstanten des gefundenen Pattern p die allen Strings gemeinsamen Teilstrings sein sollen. Die Minimalität wiederum fordert, daß zu der durch das Pattern beschriebenen Sprache möglichst wenige zusätzliche Strings gehören, die nicht in S enthalten sind, d.h. auch die Variablen müssen auf Mehrfachauftreten und Symmetrie untersucht werden. Zusammenfassend kann also gesagt werden: Es soll die größtmögliche Übereinstimmung aller zu S gehörenden Strings gefunden werden.

Beispiel:
Die durch das Pattern $p = x_1 20 x_1$ gebildete Sprache enthält ohne Zweifel die Menge $S = \{012001, 122012\}$, sie enthält aber zum Beispiel auch noch Strings der Form $1201, \ldots$.
Eine maximale Symmetrie, die auch den Anforderungen der ℓ-MINL Berechnungen genügen würde, wäre dagegen das Pattern $q = x_1 x_2 20 x_1 x_2$.

Das Problem von MINL liegt in der Überprüfung der Minimalität, bzw. konkret in der Entscheidbarkeit von "$L(q) \subsetneq L(p)$". Nach Hilfssatz 1.(2) gilt $L(q) \subseteq L(p)$ nur dann, wenn $q \leq' p$ ist, für $q \not\leq' p$ dagegen kann keine Aussage gemacht werden. Generell ist also "$\subseteq$" für zwei beliebige Pattern unentscheidbar.
Aus Hilfssatz 1.(3) ist jedoch bekannt, daß dieses Problem für zwei Pattern der gleichen Länge entscheidbar ist. Im Korrektheitsbeweis des folgenden Algorithmus wird nun gezeigt, daß ein die Menge S erzeugendes Pattern eine bestimmte Länge haben muß. Darauf aufbauend kann dann mit Hilfssatz 1.(3) gezeigt werden, daß das im Algorithmus berechnete Pattern eine minimale, S enthaltende Sprache erzeugt.

Algorithmus:
Gegeben sind die nichtleere, endliche Stringmenge S und einer der kürzesten Strings aus S: $w = a_1 \dots a_m$ mit $a_i \in \Sigma$, $i = 1 \dots m$, durch den das gesuchte Pattern aufgebaut werden soll.
(Zur Erinnerung: Soll für alle $v \in S$ gelten $v \leq' p$, dann muß $|p| \leq |v|$ ($\forall\, v \in S$), d.h. das Pattern p darf nicht länger als der kürzeste String aus S sein, und kann außerdem keine anderen Konstanten enthalten, als in den kürzesten Strings vorkommen.)

```
p_1 := x_1 ... x_m;
for i := 1 to m do
   begin
     q := p_i[a_i/x_i];
     j := i + 1;
     while S ⊈ L(q) and j ≤ m do
          begin
            q := p_i[x_j/x_i];
            j := j + 1
          end;
     if S ⊆ L(q) then p_{i+1} := q
                 else p_{i+1} := p_i
   end;
output p_{m+1};
halt;
```

Dieser Algorithmus baut das gesuchte Pattern von links nach rechts auf, prüft zuerst, ob Symbole aus dem Hilfspattern w Konstanten der Strings aus S sind und wenn nicht, ob Variablen mehrfach auftreten. (Bei gleichen Variablen werden die mit kleinerem Index umbenannt in den höheren Index.)

Korrektheit:
Dieser Algorithmus hält offensichtlich nach Beendigung der for-Schleife immer mit "halt".
Im folgenden wird durch einen Widerspruchsbeweis gezeigt, daß die Ausgabe p_{m+1} die Forderungen von ℓ-MINL erfüllt:
Angenommen p ist die Ausgabe des Algorithmus und es existiert ein weiteres Pattern r so, daß $S \subseteq L(r)$ und $L(r) \subset L(p)$. Da $S \subseteq L(r)$, ist damit auch $w \in L(r)$ und somit gilt $|r| \leq |w|$.
Betrachte:

Fall 1: $|r| < |w|$
Die Länge von p_1 wird durch den Algorithmus nicht verändert, daher ist $|w| = |p_1| = |p|$ und damit auch $|r| < |p|$

$\Rightarrow r$ kann nicht aus p hergeleitet werden, $r \not\leq' p$
Mit Hilfssatz 1.(2) und $L(r) \not\subseteq L(p)$ folgt dann ein Widerspruch zur Annahme, $|r| < |w|$ ist also falsch!

Fall 2: $|r| = |w| \;\; \Rightarrow |r| = |p|$.
Mit $L(r) \subseteq L(p)$ und Hilfssatz 1.(3) gilt $r \leq' p$,
da aber $L(r) \neq L(p)$ folgt mit Hilfssatz 1.(1): $r \neq' p$.
r entsteht aus p also nicht nur durch Variablenumbenennung,
d.h. es muß einen Index i geben so, daß $x_i \in p$ und
$r \leq' p[a_i/x_i]$ oder
es existiert in p eine von x_i verschiedene Variable x_j mit $i < j$ und
$r \leq' p[x_j/x_i]$.

Nun führt man eine längenerhaltende Funktion $g : P \to P$ ein, die ein beliebiges p_i aus dem Algorithmusverlauf, $p_i \neq p$, auf p abbildet: $p = g(p_i)$. Somit muß durch g mindestens die Variable an i-ter Stelle ersetzt werden, sei also $\alpha = a_i$ oder x_j.
Dann gilt: $p[\alpha/x_i] = g(p_i)[\alpha/x_i] = g(p_i[\alpha/x_i])$ bzw. $p \leq' p_i$
und daher mit $r \leq' p[\alpha/x_i]$: $r \leq' p_i[\alpha/x_i]$
$\Rightarrow$ (mit Hilfssatz 1.(3)) $S \subseteq L(p_i[\alpha/x_i])$.
Wird aber in p_i die Variable x_i durch α ersetzt, kann in den durch den weiteren Verlauf des Algorithmus berechneten Pattern p_i x_i nicht mehr vorkommen, also auch nicht mehr in der Ausgabe p — d.h. die Annahme war falsch und p ist das längste Pattern, dessen Sprache S enthält und eine minimale Sprache erzeugt!

<u>*Komplexität:*</u> Nachdem die Überprüfung der Minimalität durch obigen Algorithmus umgangen wird, ist ℓ-MINL also berechenbar. Allerdings ist das Problem "$q \in L(p)$" ein NP-vollständiges Problem (Angluin, 1979) und somit ℓ-MINL wenigstens NP-hart.
Der Algorithmus überprüft $S \subseteq L(p)$ höchstens $(m+1)(m+2)/2$-mal.

($\sum_{i=1}^{m}(m-i+2) = \sum_{i=0}^{m-1}(i+2) = \sum_{i=1}^{m}(i+1) = (m+1)(m+2)/2$
entspricht Anzahl der Abfragen in while in Abhängigkeit von i: ($m-i+1$), plus eine Abfrage in if.)

Ist n die Anzahl der in S enthaltenen Strings, dann wird $S \subseteq L(p)$ durch maximal n Zugehörigkeitstests entschieden. Unter der Annahme, daß diese Zugehörigkeit durch ein Orakel innerhalb eines Zeitschrittes berechnet werden kann, gilt also:

Satz 2
ℓ-MINL ist in polynomialer Zeit ($O(m^2n)$) durch eine deterministische Turing-Maschine mit Orakel in NP berechenbar!

<u>Beispiel zum Algorithmus:</u>
Seien $\Sigma = 0, 1, 212$; $X = x_1, x_2, x_3, \ldots$ und $S = \{000, 12120\}$;
w der kürzeste String aus S: $w = 000$.

$$\begin{array}{c}
p_1 = x_1 x_2 x_3 \\
\swarrow \quad \searrow \\
[0/x_1] \quad [x_2/x_1] \\
\downarrow \\
p_2 = x_2 x_2 x_3 \\
\swarrow \downarrow \searrow \\
[0/x_2] \qquad [x_3/x_2] \\
\downarrow \\
p_3 = x_2 x_2 x_3 \\
\downarrow \\
[0/x_3] \\
\downarrow \\
p_4 = x_2 x_2 0
\end{array}$$

Bild 1) Tree Search Methode in ℓ-MINL

3.4 Berechenbarkeit von ℓ-MINL für reguläre und non-cross Pattern

Im vorherigen Kapitel wurde gezeigt, daß ℓ-MINL in polynomialer Zeit berechenbar ist, wenn ein Orakel benutzt wird, daß die Zugehörigkeit zu einer Patternsprache entscheidet. Schränkt man jedoch die Pattern auf die in Kapitel 3.2 vorgestellten Arten der regulären und der non-cross Pattern ein, dann lassen sich Automaten angeben, die die Zugehörigkeit aufgrund der konkreten Struktur berechnen können. In dieser Ausarbeitung wird auf die Realisierung dieser Aufgabe jedoch nicht näher eingegangen, da diese Betrachtungen den Rahmen der Ausarbeitung sprengen würden.

3.4.1 Reguläre Pattern

Die Zugehörigkeit zu regulären Patternsprachen läßt sich in linearer Zeit berechnen, somit ist ℓ-MINL in echt polynomialer Zeit durchführbar:
In regulären Pattern kommt jede Variable höchstens einmal vor, ihre verallgemeinerte Form ist damit $p = w_1 x_1 w_2 x_2 \ldots x_n w_{n+1}$.

Dabei sind die w_i $(i = 1, 2, \ldots, n+1)$ beliebige, möglicherweise leere Strings über Σ und $x_1, \ldots, x_n$ verschiedene Variablen.
Dann ist es möglich, einen deterministischen, endlichen Automaten DFA_1 zu entwerfen, der zu einem gegebenen Pattern p den ersten konstanten String w_1 in $O(|w_1|)$ erkennt. Ebenso kann für die übrigen Konstanten w_i $(i = 2, \ldots, n+1)$ jeweils ein deterministischer, endlicher Automat DFA_i konstruiert werden, der die durch $\Sigma^+ w_i$ dargestellte Sprache in einem $O(|w_i|)$ Zeitaufwand mit Hilfe der Pattern-matching-machine-Methode erkennt ([2]).
Ein die Sprache $\Sigma^+ w_i$ akzeptierender Automat sucht also die Variable x_{i-1}, bestimmt die Konstanten rechts davon als w_i und alle Symbole links von x_{i-1} einschließlich als beliebigen String über Σ.

Werden die Automaten DFA_i verbunden, erhält man einen einzigen deterministischen und endlichen Automaten $DFA[p]$, der die von p gebildete Sprache in $O(\sum_{i=1}^{n+1} |w_i|)$ erkennt:

$$\hookrightarrow DFA_1 \longrightarrow DFA_2 \longrightarrow \cdots \longrightarrow DFA_{n+1}$$

Diese Betrachtungen begründen den folgenden Hilfssatz:

Hilfssatz 3
Für ein beliebiges, reguläres Pattern p kann ein deterministischer, endlicher und $L(p)$-erkennender Automat in einem Zeitaufwand von $O(|p|)$ entwickelt werden.

Bemerkenswert hierbei ist, daß nur die Konstanten relevant sind. Da aber per Definition der regulären Pattern alle in p vorkommenden Variablen verschieden sind, muß also kein zusätzlicher Aufwand betrieben werden, um die Variablenwerte für Vergleiche zu speichern. Theoretisch denkbar wäre demnach, daß alle Variablen in p durch die gleichen Konstanten ersetzt werden könnten; die Klasse der regulären Pattern ist jedoch zu allgemein, um solche Symmetrien zu erkennen, dafür ist ein $L(p)$ erkennender Automat aber in konstanter Zeit berechenbar.

Hilfssatz 4
Für ein beliebiges reguläres Pattern p und einen beliebigen String w ist $w \in L(p)$ in $O(|p| + |w|)$ Zeit entscheidbar.

Beweis:
Nach Hilfssatz 3 ist $DFA[p]$ in $O(|p|)$ Zeit konstruierbar, ob $DFA[p]$ w akzepiert, ist ähnlich der Konstruktion von DFA in $O(|w|)$ entscheidbar.

Nachdem die Frage des Enthaltenseins in einer regulären Patternsprache geklärt ist, ist es folglich möglich, ℓ-MINL über regulären Pattern schnell zu berechnen.
Der ursprüngliche Algorithmus kann dafür leicht abgeändert werden, da, wie oben schon erwähnt, keine Betrachtung der Variablenwerte nötig ist.

Algorithmus:
S sei wieder eine gegebene, nichtleere Stringmenge,
$w = a_1 \ldots a_k$ mit $a_i \in \Sigma$ einer der kürzesten Strings in S.

```
p_1 := x_1 ... x_k;
for i := 1 to k do
   begin
      q := p_i[a_i/x_i];
      if S ⊆ L(q) then p_{i+1} := q
                  else p_{i+1} := p_i
   end;
output p_{m+1};
halt;
```

Der Korrektheitsbeweis ist dem für allgemeine Pattern ähnlich und wird hier nicht wiederholt.

Komplexität:
Die Frage $S \subseteq L(p)$ wird im Algorithmus genau k-mal in der for-Schleife durchgeführt. Enthält S n Elemente, dann wird $S \subseteq L(p)$ durch maximal n Zugehörigkeitsprüfungen entschieden. Die Zugehörigkeit wiederum ist nach Hilfssatz 4 in $O(k+m)$ feststellbar, wobei $k = |p|$ und m die Länge des längsten Strings aus S ist. Nimmt man für den worst-case an, daß alle Strings die Länge m haben, erhält man als Komplexität von ℓ-MINL $O(m \star n \star (m+m))$ bzw. $O(m^2 n)$.

Es ergibt sich daher:

Satz 5
ℓ-MINL(S) für reguläre Pattern ist in $O(m^2 n)$ berechenbar mit
$m = max|w|\,; w \in S$ *und* $n = |S|$.

3.4.2 Non-cross Pattern

Nachfolgend wird gezeigt, daß sich ℓ-MINL für non-cross Pattern ebenfalls in polynomialer Zeit berechnen läßt. Dazu muß zuerst wieder das Problem $S \subseteq L(p)$ zu einem gegebenen Pattern p gelöst werden. Ähnlich der vorherigen Lösung für reguläre Pattern geschieht dies mit einem 2-Wege, nichtdeterministischen und finiten Automaten mit 4 Bearbeitungsbändern (2NFA(4)). Einzelheiten über diesen Automatentyp können bei Ibarra ([8]) nachgelesen werden.

Non-cross Pattern sind dadurch charakterisiert, daß darin vorkommende gleiche Variablen immer aufeinanderfolgend auftreten, ihre allgemeine Form kann daher leicht angegeben werden: $p = w_1 x w_2 x \dots x w_n y \dots$
Jedes w_i ist dabei wieder ein beliebiger, gegebenenfalls leerer, konstanter String; x, y und alle folgenden Variablen sind verschieden. Ein 2NFA(4) untersucht solange Variable für Variable, bis er zwei verschiedene entdeckt und hält die bis dahin gefundenen Konstanten w_i auf einem ersten Band fest. Für die beiden Variablen rät er dann versuchsweise (nichtdeterministisch) die Länge der ersetzenden Strings und legt den entsprechenden Platz auf jeweils einem weiteren Band ab.
Soll also für einen String entschieden werden, ob er akzeptiert wird, versucht der Automat anhand der geratenen Länge die ersetzenden Strings zu finden und übernimmt sie versuchsweise in die Bänder. Für jede dadurch mögliche (aber von der eingegebenen Stringlänge abhängigen, endlichen) Permutation wird mit dem auf dem vierten Band abgelegten String ein Pattern-matching durchgeführt. Zu testen ist dabei, ob die Konstanten übereinstimmen und ob die geratene Variablenersetzung von y und x (für jedes Auftreten von x) korrekt ist. Sind alle Vergleiche positiv ausgefallen, tritt y an die Stelle von x, die nächste, von y verschiedene Variable wird bestimmt und das Vorgehen beginnt von neuem. Auf diese Weise wird höchstens die gesamte Länge des Pattern p abgearbeitet. (Besteht das Pattern nur aus Variablen, müßte das Verfahren $|p|$-mal durchlaufen, jeder Durchlauf entspräche dabei einem Zustand des Automaten.)

Anhand dieser Betrachtung wird klar:

Hilfssatz 6
Für ein beliebiges non-cross Pattern p kann ein $L(p)$ erkennender Automat 2NFA(4) mit $O(|p|)$ Zuständen in $O(|p|)$ entwickelt werden.

Zieht man die Längenrateversuche des Nichtdeterministischen-Automaten für die Variablenersetzung in Betracht, ist die Aussage des folgenden, ohne Beweis gegebenen Hilfssatzes vorstellbar:

Hilfssatz 7
Für ein beliebiges non-cross Pattern p und einen beliebigen String w ist $w \in L(p)$ in Abhängigkeit von $|p|$ und $|w|$ in polynomialer Zeit entscheidbar.

Folgender modifizierte Algorithmus für reguläre Pattern berechnet ℓ-MINL für non-cross Pattern:
Gegeben sind wiederum die nichtleere, endliche Stringmenge S
und $w = a_1 \ldots a_k$, $a_i \in \Sigma$, einer der kürzesten Strings aus S.

```
p_1 := x_1 ... x_k;
j := 0;
for i := 1 to k do
   begin
      q := p_i[a_i/x_i];
      if S ⊈ L(q) then
         begin
            if j ≠ 0 then
               begin
                  q = p_i[x_i/x_j];
                  if S ⊆ L(q) then p_{i+1} := q
                                else  p_{i+1} := p_i
               end;
            j := i
         end
      else p_{i+1} = q
   end;
output p_{k+1};
halt;
```

Im Gegensatz zu den regulären Pattern muß hier noch überprüft werden, ob aufeinanderfolgende Variablen gleich sind. Da die Variablen aus p nacheinander abgearbeitet werden, reicht es aus, immer nur zwei aufeinanderfolgende auf Gleichheit zu untersuchen. Diese versuchsweise Umbenennung erfolgt durch $q = p_i[x_i/x_j]$, wobei links von x_i alle gleichen Variablen durch den vorherigen Schleifendurchlauf schon in x_j umgewandelt wurden.
Der Korrektheitsbeweis verläuft ansonsten wieder analog dem der allgemeinen Pattern.

Komplexität: Enthält die Menge S n Elemente, wird die Zugehörigkeit $2m \star n$-mal entschieden, nach Hilfssatz 7 wird eine Zugehörigkeit in polynomialer Zeit berechnet, abhängig von k und m ($m = max|w|\,; w \in S$).
Zusammenfassend erhält man:

Satz 8
ℓ-MINL(S) für non-cross Patternsprachen ist in polynomialer Zeit berechenbar.

Abschließende Bemerkung:
Betrachtet man die in diesem Kapitel gewonnen Erkenntnisse hinsichtlich der Anwendung in einem Dateneingabesystem, dann fällt auf, daß reguläre Pattern ausreichen bzw. allgemein genug sind, um alle denkbaren Eingabemöglichkeiten in ein Dateneingabesystem zu erfassen. Der Berechnungsaufwand ist polynomial und bleibt dies selbst beim Übergang zu non-cross Pattern. Dies gilt auch für die in der folgenden Ausarbeitung "Patternlernende Systeme in Dateneingabesystemen" eingeführte "Erweiterung der regulären Pattern". In der Klasse der *extended regulär Pattern* ist es erlaubt, Variablen durch das leere Element zu ersetzen. Diese Sprache erkennende Automaten können auf die gleiche Art entwickelt werden, wie Automaten für reguläre Pattern (Kapitel 3.4.1). Die Komplexität hierbei steigt in der eigentlichen ℓ-MINL-Berechnung durch die Entwicklung des Pattern.

Literaturverzeichnis

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Time and Tape Complexity of Pushdown Automaton Languages.* Inform. Contr. 13, 186-206, 1968.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[3] D. Angluin. *Finding Patterns in Common to a Set of Strings.* in Proceeding, 11th Annual ACM Symposium on Theory of Computing, pp. 130-141, 1979.

[4] D. Angluin. *Inductive Inference of Formal Languages from Positiv Data.* Inform. Contr. 45, 117-135, 1980.

[5] E.M. Gold. *Language Identification in the Limit.* Inform. Contr. 10, 447-447, 1967.

[6] J.E. Hopcroft and J.D. Ullman. *Formal Languages and their Relation to Automata.* Addison-Wesley, 1969.

[7] John E. Hopcroft and Jeffrey D. Ullman. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie.* Addison-Wesley, 1988.

[8] O.H. Ibarra. *On Two-Way Multihead Automata.* JCSS, 7, 28-36, 1973.

[9] Takeshi Shinohara. *Polynomial Time Inference of Pattern Languages and Its Application.* Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science, May 1982.

[10] R. Solomonoff. *A Formal Theory of Inductive Inferences.* Inform. Contr. 7, 1-22, 1964.

Kapitel 4

Patternlernende Systeme in Dateneingabesystemen

Guido Kühn

Übersicht

In diesem Vortrag wird die praktische Anwendung von patternlernenden Systemen dargestellt. Zunächst wird noch einmal kurz über die Grundlagen der patternlernenden Systeme referiert. Um ein Gefühl für den Einsatz dieser Techniken in der Praxis zu bekommen, werden danach drei Vorstudien zu einem real implementierten System vorgestellt, welches auf der Basis eines patternlernenden Systems ein flexibles, lernendes Dateneingabesystem für ein Bibliothekssystem realisiert. Am Beispiel dieses Systems werden zunächst wesentliche Eigenschaften des ersten, relativ einfachen Ansatzes besprochen, sowie darauf aufbauend weitere Verfeinerungen des zugrundeliegenden Algorithmus betrachtet.

Nach dieser systemunabhängigen Darstellung der dabei verwendeten Methoden werden die Möglichkeiten solcher Systeme in der Anwendung im Internet dargelegt. Es werden dabei Ideen für Systeme vorgestellt, welche dem Benutzer die Erstellung von HTML-Dokumenten und die Navigation im Internet vereinfachen sollen. Es wird aber auch auf Einsatzmöglichkeiten hingewiesen, die nicht direkt an der Mensch-Maschine-Schnittstelle, sondern eher an der Netz-Rechner-Schnittstelle zum Tragen kommen.

4.1 Einleitung

Wo immer Daten in einen Computer übertragen werden sollen, kommen Dateneingabesysteme zur Anwendung, also eben Systeme, mit deren Hilfe Daten eingegeben werden. Egal ob es sich um eine mechanische, elektronische oder manuelle Datenübertragung handelt, stets werden die Daten von einem solchen System eingelesen.

Speziell bei der manuellen Übertragung der Daten durch Menschen soll dieses System aber nicht einfach nur die Eingabe von Daten ermöglichen, sondern es soll für diesen Menschen diese Eingabe möglichst einfach und bequem gestaltet werden, um ihm diese ermüdende und anstrengende Tätigkeit zu erleichtern. Dies ist insbesondere dann notwendig, wenn es sich dabei um einen Menschen handelt, der es nicht gewohnt ist, große Datenmengen schnell und fehlerfrei einzugeben, sondern diese Aufgabe nur ab und zu wahrnimmt. Um dies zu erreichen muß man große Sorgfalt auf die Gestaltung des zur Anwendung kommenden Dateneingabesystems verwenden. Eine wesentliche Erleichterung stellt es dar, wenn das System dem Benutzer überflüssige Tipparbeit abnehmen kann.

Patternlernende Systeme sind in der Lage, in einer Menge von Daten strukturelle Gemeinsamkeiten aufzuspüren und diese beim Hinzufügen neuer Daten anzupassen. Bei Daten, deren Aufbau bestimmten, wiederkehrenden Mustern folgt (also z.B. Adressen im Gegensatz zu Fließtext) kann ein patternlernendes System diese Muster erkennen und z. B. wiederkehrende, konstante Zeichenfolgen dem Benutzer bereits vorgeben, so daß dieser solche Zeichenfolgen nicht wiederholt selbst einzugeben braucht.

Hierzu ein Beispiel: In einer Behörde existieren für die verschiedensten Anträge und Vorgänge unterschiedliche Formulare. Die mit diesen Formularen gewonnenen Daten sollen elektronisch verarbeitet werden. Eine Anforderung an das System soll die absolute Benutzerfreundlichkeit sein. Insbesondere soll den Sekretärinnen, die diese Daten erfassen sollen, erspart werden, sich mit komplizierten Menü- oder Befehlsstrukturen befassen zu müssen.

Die Bedienung soll lediglich durch das Wort „Eingabe" oder „Anfrage" zwischen Erfassen und Abrufen von Daten unterscheiden können, danach wird das entsprechende Formular im Klartext erfaßt. Das System soll nun möglichst schnell erkennen, um welches Formular es sich handelt und dem Datenerfasser danach die konstanten Teile, also die Namen der Felder des Formulars, vorgeben, damit er sie nicht immer wieder tippen muß.

Leider ist diese Behörde recht papierfreudig, so daß oft neue Formulare mit neuem Zweck und neuem Aufbau entworfen und eingeführt werden. Das System soll nun möglichst von sich aus auf diese Formulare reagieren können; die Möglichkeit, daß ein Systemverwalter die Struktur jedes Formulares in das System eingibt, scheidet aus.

Hier kann ein patternlernendes System zur Anwendung kommen, das nach wenigen Beispielen für ein neues Formular dieses bereits „gelernt" hat und die entsprechenden Konstanten vorgeben kann.

4.2 Benötigte Begriffe aus der Theorie des Patternlernens

In diesem Abschnitt werden noch einmal kurz die Begriffe aus der Theorie des Patternlernens angerissen, die in diesem Vortrag verwendet werden. Die meisten Begriffe wurden im vorhergehenden Vortrag bereits ausführlich mit ihren theoretischen Grundlagen vorgestellt; hier sollen sie eher im Hinblick auf eine praktische Verwendung dargelegt werden. Dabei kommt es teilweise zu Vergröberungen der Betrachtungsweise.

4.2.1 Patternlernendes System

Gegen Ende der 70er Jahre hat D. Angluin die Klasse der Sprachen eingeführt, die aus positiven Daten inferierbar sind. Zu dieser Klasse gehören die Pattern Languages, Sprachen, deren Worte durch ein Muster beschrieben werden.

Inferierbar aus Daten bedeutet im Fall von Sprachen, daß man Informationen darüber erhält, welche Worte in der Sprache enthalten sind, indem man konkrete Worte aus dieser Sprache angegeben bekommt. Aus diesen Informationen bemüht man sich, auf das Muster zu schließen, welches die Sprache beschreibt.

Inferierbar aus positiven Daten schränkt die Möglichkeit des Schließens insofern ein, als man nur Bespiele für Worte erhält, die in der Sprache enthalten sind, jedoch nicht für Worte, die nicht in der Sprache enthalten sind. Insbesondere hat man keine Möglichkeit, „nachzufragen", ob Worte, die man aus dem gefundenen Muster generieren kann, in der Sprache liegen. Man stellt also ein Muster auf, welches eine Sprache beschreibt, in der alle bisher bekannten Worte enthalten sind. Erhält man ein weiteres Wort, so liegt es entweder in dieser Sprache und bestätigt so das Muster, oder es liegt nicht in dieser Sprache, und man muß ein neues Muster aufstellen, welches dieses Wort ebenfalls abdeckt. Diesen (im allgemeinen unendlichen) Prozeß der Anpassung kann man auch als „Lernen des richtigen Musters" bezeichnen.

In diesem Vortrag wird auf zwei bestimmte Klassen von Pattern Languages näher eingegangen: Regular und Extended Regular Pattern Languages. Diese gehören zu einer Klasse von Pattern Languages, welche mit polynomialem Zeitaufwand „gelernt" werden können [5] [6].

4.2.2 Pattern

Sei Σ eine endliche, nicht-leere Menge von Symbolen und $X = \{x_1, x_2, \ldots\}$ eine abzählbare Menge von Symbolen mit $\Sigma \cap X = \emptyset$. Man bezeichnet die Elemente von Σ als Konstanten und die Elemente von X als Variablen.

Eine nicht-leere Zeichenkette p über $\Sigma \cup X$ heißt ein Pattern. Kommt in p kein x_i mehrfach vor, so bezeichnet man p als regulär.

4.2.3 RPL: Regular Pattern Language

Die Klasse der regulären Pattern Languages ist eine Unterklasse der Pattern Languages, wie sie von Angluin [1] eingefürt wurden.

Die Sprache $L(p)$ eines Patterns p ist die Menge aller Zeichenketten, die man erhält, wenn man für jede Variable in p alle nicht-leeren Zeichenketten über Σ einsetzt.

Die Klasse der Sprachen der Regulären Pattern RPL ist $\{L(p); p \text{ regulär}\}$.

4.2.4 ERPL: Extended Regular Pattern Language

Die Klasse der erweiterten Pattern Languages wurde von Shinohara eingeführt [6]. Der Unterschied zu RPL besteht darin, daß Variablen zusätzlich auch mit der leeren Zeichenkette belegt werden können.

Die erweiterte Sprache $EL(p)$ eines Patterns p ist die Menge aller Zeichenketten, die man erhält, wenn man für jede Variable in p alle Zeichenketten über Σ inclusive der leeren Zeichenkette einsetzt.

Entsprechend zu RPL definiert man die Klasse der erweiterten (extended) Sprachen der regulären Pattern ERPL als $\{EL(p); p \text{ regulär}\}$.

4.2.5 Berechnung von Pattern

Für eine Menge von Worten S gibt es immer mehrere Sprachen, in denen sie enthalten sein können. Je nach dem zugrundeliegenden Pattern haben diese Sprachen verschiedene Mächtigkeiten. Allerdings gibt es nur eine minimale Sprache, in der neben S die kleinste mögliche Zahl an zusätzlichen Worten enthalten ist. Alle anderen Sprachen sind mächtiger als diese Sprache. Die dazugehörigen Pattern werden mit zunehmender Mächtigkeit entsprechend immer „ungenauer“.

Man versucht nun, zu einer Menge von Zeichenketten das genaueste Pattern aufzustellen, das heißt, diese minimale Sprache zu finden. Ein Algorithmus, der so etwas leistet, ist ein MINL-Algorithmus (MINimal Language), formal von Angluin [1] folgendermaßen definiert:

MINL (S): Für eine nicht-leere Menge von Zeichenketten S und eine indizierte Familie von Sprachen $CL = L_1, L_2, \dots$ finde einen Index i, so daß $S \subseteq L_i$ und es keinen Index j gibt mit $S \subseteq L_j \subseteq L_i$.

4.2.6 Berechnung von Pattern für RPL

In [5] gibt Shinohara einen Algorithmus zur MINL-Berechnung für Regular Pattern Languages an.

Eingabe: S, nicht-leere endliche Menge von Zeichenketten über Σ.

Bestimme $w = a_1 \dots a_m$, $a_i \in \Sigma$, eine der kürzesten Zeichenketten in S.
$p \leftarrow x_1 \dots x_m$, $x_i \in X$.
Für $i \leftarrow 1 \dots m$:
  $q := p[a_i/x_i]$
  Falls $S \subseteq L(q)$, dann $p \leftarrow q$
p ist das gesuchte Pattern

$p[a/x]$ bezeichet die Ersetzung einer Variablen x durch eine Konstante a im Pattern p.

Umgangssprachlich funktioniert dieser Algorithmus so: Ausgangspunkt ist das allgemeinste Pattern p, welches nur aus Variablen besteht. Es hat die Länge des kürzesten Wortes w in S. Da in RPL jede Variable mit mindestens einem Zeichen belegt werden muß, darf p nicht länger sein als w, welches sonst nicht von dem Pattern beschrieben würde. Falls mehrere Worte diese kürzeste Länge haben, nimmt man ein beliebiges von diesen.

Nun werden nacheinander alle Variablen in p durch das Zeichen ersetzt, welches sich in w an derselben Stelle befindet wie die Variable. Damit ist gewährleistet, daß w in $L(p)$ liegt. Für alle anderen Worte in S muß das noch geprüft werden.

Liegen alle Worte aus S in $L(p)$, dann ist dieses Pattern korrekt, und die vorgenommene Ersetzung hat ein genaueres Pattern hervorgebracht. Falls nicht, so wird die Ersetzung rückgängig gemacht.

Am Ende erhält man ein Pattern, welches die kleinste Sprache aus RPL beschreibt, in der die Worte aus S enthalten sind.

Dieser Algorithmus hat eine Zeitkomplexität von $O(n^2)$ [5].

4.2.7 Berechnung von Pattern für ERPL

Der Algorithmus zur Berechnung von Pattern für ERPL ist wesentlich komplexer und kann bei Shinohara [6] genau nachgelesen werden. Hier soll nur eine umgangssprachliche Beschreibung der Berechnung erfolgen. Nach jedem Schritt wird anhand eines Beispieles gezeigt, welche Auswirkungen er hat.

Ein Pattern p hat die allgemeine Form $w_0x_1w_1x_2w_2\ldots$, wobei x_i Variablen und w_i beliebige (auch leere) konstante Zeichenketten aus Σ sind. Der Algorithmus baut nun fortlaufend die einzelnen w auf. Ist ein w_i vollständig gefunden, so wird x_{i+1} eingefügt und das entsprechende nächste w_{i+1} bestimmt.

Beispiel:

$s_1 = \texttt{(TI)␣FINDING␣PATTERNS␣COMMON}$
$s_2 = \texttt{(TI)␣INDUCTIVE␣INFERENCE}$
$w_0 = \texttt{(TI)␣}$
$w_1 = \texttt{IND}$
$p_1 = w_0x_1w_1x_2 = \texttt{(TI)␣?IND?}$

Diese Bestimmung des w_i erfolgt für alle Zeichenketten s_j aus S parallel; es entstehen also j Worte w_{i_j} Das erste Zeichen von w_{i_j} ist das erste Zeichen in s_j, welches wieder zu einem gültigen Pattern führt.

$w_{2_1} = \texttt{IN}$
$w_{2_2} = \texttt{C}$
$p_{2_1} = w_0x_1w_1x_2w_{2_1}x_3 = \texttt{(TI)␣?IND?IN?}$
$p_{2_2} = w_0x_1w_1x_2w_{2_2}x_3 = \texttt{(TI)␣?IND?C?}$

Nun muß entschieden werden, welches der gefunden Pattern p_{i_j} das gesuchte ist. Gesucht wird das längste Pattern, also das Pattern, welches die meisten konstanten Teile besitzt. Das wird dasjenige sein, in welchem die längste notwendige Belegung der Variable x_i minimal ist, da in diesem Fall mehr Zeichen zur weiteren Suche nach Konstanten übrig bleiben.

Beispiel:

p_{2_1} matcht s_1 mit $x_1 = \texttt{F}$ und $x_2 = \epsilon$
p_{2_1} matcht s_2 mit $x_1 = \epsilon$ und $x_2 = \texttt{UCTIVE␣}$

p_{2_2} matcht s_1 mit $x_1 = \texttt{F}$ und $x_2 = \texttt{ING␣PATTERNS␣}$
p_{2_2} matcht s_2 mit $x_1 = \epsilon$ und $x_2 = \texttt{U}$

Die längste notwendige Belegung von x_2 hat bei p_{2_1} eine Länge von 7 Zeichen, bei p_{2_2} eine Länge von 13 Zeichen. Also wird p_{2_1} das neue Pattern p_2.

Beispiel:

$s_1 = \texttt{(TI)␣FINDING␣PATTERNS␣COMMON}$
$s_2 = \texttt{(TI)␣INDUCTIVE␣INFERENCE}$
$w_0 = \texttt{(TI)␣}$
$w_1 = \texttt{IND}$
$w_2 = \texttt{IN}$
$p_2 = w_0 x_1 w_1 x_2 w_2 x_3 = \texttt{(TI)␣?IND?IN?}$

Dieser Vorgang wiederholt sich, bis eine Zeichenkette s_j vollständig verbraucht wurde. Dann sind keine weiteren Zeichen mehr vorhanden, und das gefundene Pattern ist das längste mögliche. Dadurch ist die zum Pattern gehörende Sprache minimal.

Dieser Algorithmus ist wesentlich aufwendiger als der Algorithmus zur Berechnung von Pattern für RPL. Sein Rechenaufwand beträgt $O(n^4)$ [6].

4.2.8 Codieren von Zeichenketten

Es gibt viele Anwendungsfälle, wo man als kleinste Einheit der Eingabe sinnvollerweise nicht den Buchstaben sondern das Wort betrachtet. In diesem Fall besteht das Muster nicht aus Variablen und konstanten Buchstaben, sondern aus Variablen und konstanten Worten.

Beispiel:

Eingabe: `Das␣Erkennen␣von␣Pattern.`
Eingabe: `Das␣Erkennen␣von␣Mustern.`
Ausgabe: `Das␣Erkennen␣von␣?tern.` (Buchstabenweises Vergleichen)
Ausgabe: `Das␣Erkennen␣von␣?.` (Wortweises Vergleichen)

Dieses wortweise Vergleichen kann man leicht dadurch erreichen, daß man alle Worte der Eingabe durch Indices in eine Tabelle ersetzt, in die alle auftretenden Worte eingetragen werden. Gleiche Worte erhalten natürlich den gleichen Index. Das Mustervergleichen geschieht dann auf Ketten von Indices. Die Menge Σ der Konstanten ist dann die Menge aller Indices. Zu

Beachten ist, daß auch Sonderzeichen wie das Leerzeichen in die Tabelle aufgenommen werden müssen.

Die obige Eingabe führt zu folgender Tabelle:

1: `Das`
2: ␣
3: `Erkennen`
4: `von`
5: `Pattern`
6: `.`
7: `Mustern`

und zu folgenden Indexketten:

Eingabe: `12324256`
Eingabe: `12324276`
Ausgabe: `123242?6`

Als weiterer Vorteil sinkt die Länge der Eingabeketten drastisch, was bei einem von der Länge der Eingabe abhängigen Rechenzeitaufwand eine entsprechend hohe Zeitersparnis einbringt (bei einem Aufwand $O(n^4)$ der MINL-Berechnung für ERPL bei diesem Beispiel ein Faktor um 600).

4.3 Dateneingabesystem SIGMA

Es soll nun ein praktisches Beispiel für ein lernendes Dateneingabesystem gezeigt werden. Hier betrachten wir dazu das System SIGMA, ein Informationssystem, welches von Arikawa und Shinohara in den frühen 80er Jahren an der Universität von Kyushu implementiert wurde [3].

Die Daten, welche in SIGMA verwendet werden, sind ausschließlich Texte oder Zeichenketten. Das System besitzt ein lernendes Dateneingabesystem als Mensch-Maschine-Schnittstelle, um den Anwendern die Eingabe der Daten zu vereinfachen.

Zur Illustration dienen hier bibliographische Daten. Diese werden vom Anwender in folgender Form eingegeben:

```
$
Author: Angluin, D.
Title: Inductive Inference of Formal Languages from Positive Data
Journal: Information and Control, 40
Year: 1980
```

```
$
Author: Gold, E.M.
Title: Limiting Recursion
Journal: Journal of Symbolic Logic, 30
Year: 1965
$
...
```

„\$" ist hier ein spezielles Zeichen zur Trennung der Datensätze. Im Verlauf der Dateneingabe wird das System erkennen, daß die eingegebenen Daten dem Muster

`Author:` w
`Title:` x
`Journal:` y
`Year:` z

entsprechen. Dabei stellen w, x, y und z die Variablen dar.

Hat das System diesen Aufbau erkannt, so gibt es die konstanten Teile `Author:`, `Title:`, `Journal:` und `Year:` selbständig aus, und der Benutzer muß nur noch die variablen Teile eingeben.

In diesem Vortrag wird nun nicht das gesamte Dateneingabesystem von SIGMA besprochen, welches den Rahmen sprengen würde, sondern nur Vorstudien, welche vorbereitend zur Implementierung des Dateneingabesystems von SIGMA ausgeführt wurden. Sie zeigen deutlich die verschiedenen Ansätze und deren Probleme.

Das Dateneingabeformat wurde dahingehend modifiziert, daß es nun dem Muster

`(AR)` x `(TI)` y `(JP)` z

entspricht.

4.3.1 Studie A: RPL

Hier wird ein patternlernendes System auf der Basis der Klasse der Regular Pattern Languages realisiert. Als Eingabe dient die Menge aller Zeichenketten die über die Tastatur eingegeben werden können.

Es läuft ein Dialog mit dem Benutzer ab, der diesem Schema folgt:

1) Eingabe der Zeichenkette
2) Prüfung mit dem bereits gelernten Pattern

3) Falls eine Änderung des Patterns notwendig ist:
MINL berechnen und das neue Pattern ausgeben
4) Ab Punkt 1) wiederholen

Eine Beispielsitzung mit diesem System lief folgendermaßen ab:

E: `(AR) D. ANGLUIN (TI) FINDING PATTERNS COMMON (JP) 11TH ANNUAL ACM`
E: `(AR) D. ANGLUIN (TI) INDUCTIVE INFERENCE (JP) INFORM. CONTR. 45`
A: `(AR) D. ANGLUIN (TI) ?ND????????????????????????????? ??N??? ??`

E: `(AR) S. ARIKAWA (TI) ONE-WAY SEQUENTIAL SEARCH (JP) BULL. MATH. STAT.`
A: `(AR) ?. A????I? ??????????????????????C????????????????????????`

E: `(AR) E.M. GOLD (TI) LANGUAGE IDENTIFICATION (JP) INFORM. CONTR. 10`
A: `(AR) ?. ????????????????????????????????????????????? ?????????`

E: `(AR) R. SOLOMONOFF (TI) A FORMAL THEORY (JP) INFORM. CONTR. 7`
A: `(AR) ?. ???????????????????????????????????? ??????? ????????`

E: `(AR) ZZ ZZ (TI) ZZ ZZ (JP) ZZ ZZ`
A: `(AR) ?? ?? (TI) ?? ?? (JP) ?? ??`

Offenbar ist dieses System nicht besonders gut zur praktischen Anwendung mit den Ziel der Eingabeerleichterung von Daten geeignet. Es erkennt zwar konstante Zeichen, nicht aber konstante Zeichengruppen, wenn sich diese in ihrer relativen Stellung ändern. So verschwinden zum Beispiel die Zeichengruppen „(TI)“ und „(JP)“ vollständig aus dem Muster, da sie nicht immer an derselben Stelle in der Eingabe erscheinen.

Auch erscheinen einem unbedarften Benutzer die Muster ziemlich willkürlich geraten, und die einzige gelernte konstante Zeichenfolge, die für die gewünschte Anwendung relevant ist, ist „(AR)“. Dieses System würde einen Benutzer eher behindern, als ihm helfen.

Allerdings treibt das System auch keinen übermäßig hohen Rechenaufwand; die verwendete Inferenzmaschine für RPL hat eine Zeitkomplexität von $O(n^2)$ (n ist die Länge der Eingabezeichenketten) [4].

4.3.2 Studie B: ERPL

Hier wird nun ein musterlernendes System auf der Basis der Klasse der Extended Regular Pattern Languages realisiert. Als Eingabe dient die Menge aller Zeichenketten, die über die Tastatur eingegeben werden können.

Das Schema des Dialogs ist unverändert.

1) Eingabe der Zeichenkette
2) Prüfung mit dem bereits gelernten Pattern
3) Falls eine Änderung des Patterns notwendig ist:
   MINL berechnen und das neue Pattern ausgeben
4) Ab Punkt 1) wiederholen

An der Beispielsitzung mit den gleichen Eingaben wie in Studie A erkennt man deutlich einen Fortschritt:

E: `(AR) D. ANGLUIN (TI) FINDING PATTERNS COMMON (JP) 11TH ANNUAL ACM`
E: `(AR) D. ANGLUIN (TI) INDUCTIVE INFERENCE (JP) INFORM. CONTR. 45`
A: `(AR) D. ANGLUIN (TI) ?IND?IN?ER?N?C? (JP) ? ?N? ?`

E: `(AR) S. ARIKAWA (TI) ONE-WAY SEQUENTIAL SEARCH (JP) BULL. MATH. STAT.`
A: `(AR) ?. A?I? (TI) ?N?I? ?E?R?C? (JP) ? ? ?`

E: `(AR) E.M. GOLD (TI) LANGUAGE IDENTIFICATION (JP) INFORM. CONTR. 10`
A: `(AR) ?. ? (TI) ?N? ?E?N?C? (JP) ? ? ?`

E: `(AR) R. SOLOMONOFF (TI) A FORMAL THEORY (JP) INFORM. CONTR. 7`
A: `(AR) ?. ? (TI) ? ?E? (JP) ? ? ?`

E: `(AR) ZZ ZZ (TI) ZZ ZZ (JP) ZZ ZZ`
A: `(AR) ? ? (TI) ? ? (JP) ? ?`

Hier erkennt man einen deutlichen Fortschritt gegenüber Studie A. Da in der ERPL auch leere Zeichenketten als Belegung für Variablen zugelassen sind, ist das System in der Lage, die konstanten Zeichenkette so „zu verschieben“, daß sie von jedem Muster überdeckt werden.

„e“ und „n“ als häufigste Buchstaben halten sich entsprechend lange im Muster, genauso der Punkt nach dem ersten Initial. Diese Überreste würden bei weiterer Dateneingabe aber verschwinden.

Dieses System ist für den gewünschten Zweck bereits viel geeigneter, es ist in der Lage, die relevanten konstanten Zeichenketten zu finden und (schnell) zu lernen. Leider ist es im Vergleich zu Studie A viel rechenzeitaufwendiger. Es besitzt eine Zeitkomplexität von $O(n^4)$. [6] Diese Steigerung des Rechenaufwandes rührt hauptsächlich von der Erweiterung der möglichen Belegungen der Variablen um leere Zeichenketten her.

4.3.3 Studie C: ERPL mit Codierung

Wenn man sich die Aufgabenstellung genauer anschaut, dann stellt man fest, daß es sich nicht um das Erkennen konstanter Zeichen sondern um das Erkennen konstanter Zeichenketten handelt. Man kann zum Beispiel davon ausgehen, daß sich einmal als konstant erkannte Worte in ihrem Aufbau nicht mehr verändern.

Trägt man dieser Tatsache Rechnung, so kann man nach dem Eingabeschritt einen weiteren Verarbeitungsschritt einführen, welcher die Zeichenkette zunächst in einzelne Wörter zerlegt und diese danach durch Symbole, also beispielsweise durch Indices auf eine Wörtertabelle darstellt. Auf diesen Symbolfolgen wird nun die Mustererkennung durchgeführt.

Genaugenommen arbeitet das System nun nicht mehr auf Ketten von Zeichen, wie sie die Tastatur liefert, sondern auf einem viel größeren, potentiell unendlichen Eingabealphabet. Diese Unendlichkeit wird in der Praxis aber kaum zum Tragen kommen, da Eingaben mit unendlicher Länge eher selten sind. Der Vorteil an Erkennungsgenauigkeit und Rechengeschwindigkeit, den man aber durch diese Modifikation erzielt, ist signifikant.

In Studie C wird nun ebenfalls ein musterlernendes System auf der Basis der Klasse der Extended Regular Pattern Languages realisiert. Als Eingabe dient die Menge aller Zeichenketten, die über die Tastatur eingegeben werden können. Allerdings wird an den entsprechenden Stellen der beschriebene Codierungsschritt eingefügt. Bei der Ausgabe des gelernten Musters muß diese Codierung natürlich in umgekehrter Richtung erfolgen, um das Pattern lesen zu können.

Das geänderte Schema des Dialogs:

1) Eingabe der Zeichenkette
2) Codierung der Wörter
3) Prüfung mit dem bereits gelernten Pattern
4) Falls eine Änderung des Patterns notwendig ist:
   MINL berechnen und das neue Pattern decodiert ausgeben
5) Ab Punkt 1) wiederholen

An der Beispielsitzung mit den gleichen Eingaben wie in Studie A und B erkennt man den Vorteil dieses Ansatzes:

E: `(AR) D. ANGLUIN (TI) FINDING PATTERNS COMMON (JP) 11TH ANNUAL ACM`
E: `(AR) D. ANGLUIN (TI) INDUCTIVE INFERENCE (JP) INFORM. CONTR. 45`
A: `(AR) D. ANGLUIN (TI) ? ? (JP) ? ? ?`

E: `(AR) S. ARIKAWA (TI) ONE-WAY SEQUENTIAL SEARCH (JP) BULL. MATH. STAT.`
A: `(AR) ? ? (TI) ? ? (JP) ? ? ?`

E: `(AR) E.M. GOLD (TI) LANGUAGE IDENTIFICATION (JP) INFORM. CONTR. 10`
E: `(AR) R. SOLOMONOFF (TI) A FORMAL THEORY (JP) INFORM. CONTR. 7`
E: `(AR) ZZ ZZ (TI) ZZ ZZ (JP) ZZ ZZ`
A: `(AR) ? ? (TI) ? ? (JP) ? ?`

Die generierten Muster sind wesentlich natürlicher und intuitiv richtiger als die Muster in den vorhergehenden Studien. Das richtige Muster wird nach viel weniger Eingaben erkannt und auch richtig beibehalten.

Die Schritte Codierung und Decodierung benötigen bei der verwendeten Implementierung $O(n)$ bzw. $O(\log n)$ Rechenzeit; die Zeitkomplexität der MINL-Berechnung liegt unverändert bei $O(n^4)$. Da sich aber die Länge der Zeichenketten durch das Codieren drastisch verringert, wird die hohe Komplexität der Inferenz mehr als ausgeglichen.

Dieses System würde die Anforderungen, die aus der Aufgabenstellung hervorgehen, bestens erfüllen.

4.3.4 Diskussion der drei Ansätze

Deutlich war zu sehen, daß die Einführung leerer Zeichenketten durch ERPL erheblich zur Steigerung der Leistung des Systems beiträgt.

Auch ist es sinnvoll, sich vor der Entwicklung eines patternlernenden Systems Gedanken über die Struktur der Eingabe zu machen und festzulegen, aus welchen Einheiten sich die zu erkennenden Muster eigentlich zusammensetzen und diesbezügliche Codierungsschritte einzuführen. Zwar mag man so auf umfangreichere Alphabete kommen, aber im Vordergrund sollte stets die Effektivität (und auch die Effizienz) des patternlernenden Systems stehen.

So ist die Lösung aus Studie A im praktischen Einsatz kaum anzuwenden. Zwar ist sie relativ schnell, aber die Ergebnisse sind schlicht unbrauchbar. Studie B ist schon deutlich besser, leider aber auch deutlich langsamer. Sowohl was Geschwindigkeit als auch Präzision der Erkennung anbelangt steht Studie C ungeschlagen an der Spitze.

Für den praktischen Einsatz eines solchen Systems ist allerdings die Beschränkung auf nur ein Pattern nicht praktikabel. Sollen zum Beispiel Zeitschriftenartikel und Bücher und vielleicht noch weitere Medien von ein und demselben System erfaßt werden, so darf die Eingabe des einen Medientyps nicht das bereits gefundene Muster für einen anderen Typ zerstören. Daher ist es unumgänglich, den Algorithmus auf die Verwaltung mehrerer Pattern zu erweitern.

4.4 Weitere Anwendungsmöglichkeiten

Hier werden nun einige (teilweise hypothetische) Anwendungen beschrieben, in denen patternlernende Systeme eine wesentliche Rolle spielen können.

4.4.1 Editoren für HTML-Dokumente

Die vorherrschende Beschreibungssprache für Dokumente im Internet ist HTML, HyperText Markup Language. Obwohl sie keinen allzu komplizierten Aufbau hat, ist die Erstellung von Dokumenten doch mühselig. Nicht jedem Benutzer ist die Struktur sofort einsichtig. Daher gibt es bereits zahlreiche Textverarbeitungen, welche die Erstellung solcher Dokumente unterstützen.

Streift man durch das Netz, so stellt man fest, daß sich auch beim elektronischen Publizieren von Dokumenten ähnliche Probleme ergeben wie beim Desktop Publishing: struktureller und (obwohl HTML bereits die übelsten Fehler vermeiden hilft) typographischer Wildwuchs.

Hier wären unterstützende Systeme hilfreich. Da aber jeder Benutzer doch einen gewissen Grad an Individualität anstrebt, wäre ein System praktisch, in welches der Benutzer HTML-Seiten, die er im Netz gefunden hat und die seinem Geschmack entsprechen, übertragen kann, und das als Ausgabe einen Vorschlag für ein HTML-Dokument liefert, welches in Struktur und Aussehen den wesentlichen Mustern der Vorgaben entspricht.

Der hier zur Anwendung kommende Mustererkenner müßte allerdings so stabil arbeiten, daß er durch ein einzelnes Dokument, welches dem bisher gefundenen Muster nur wenig oder überhaupt nicht entspricht, nicht „aus der Bahn geworfen“ wird.

4.4.2 Agenten

Bei der großen Informationsflut im Internet ist es fast nicht mehr möglich, den Überblick über die für einen speziellen Benutzer interessanten Dokumente zu behalten beziehungsweise diese Dokumente überhaupt erst einmal zu finden. Hier helfen Agenten weiter, also Programme, welche das Netz selbsttätig durchsuchen und dem Benutzer für ihn interessante Seiten heraussuchen.

Ein möglicher Ansatz, den Interessantheitsgrad einer Seite zu bestimmen, ist in den vom Benutzer als interessant bezeichneten Seiten Muster aufzuspüren, und Seiten, welche diesem Muster mehr oder weniger entsprechen dem Benutzer anzubieten. Hier könnten Muster zur Anwendung kommen, welche durch Codieren nicht nur struktureller, sondern auch inhaltlicher Informationen erhalten werden.

Der Mustervergleicher müßte ebenfalls etwas „unscharf“ arbeiten, damit auch Seiten gefunden werden, die dem Pattern nur „grob“ entsprechen.

4.4.3 Makrofindendes System

Oft muß ein Benutzer in gleichen Situationen das gleiche tun. Hier kann ein System hilfreich sein, welches in den Aktionen des Benutzers Muster, also wiederkehrende Situationen und Verhaltensweisen findet und dem Benutzer in ähnlichen oder gleichen Situationen Aktionsfolgen zur Ausführung vorschlägt.

Solche Muster könnten auch vom Hersteller eines Programmes mitgeliefert werden, um dem Käufer die Anwendung des Programmes durch Hilfestellungen zu erleichtern.

Durch Auffinden immer wiederkehrender Verhaltensweisen und Aktionsfolgen von (Test-) Benutzern könnte der Hersteller eines Systems Informationen über häufig auftretende Anwendungsfälle erhalten und vielleicht die eine oder andere ergonomische Klippe in seinem System beseitigen, zum Beispiel indem er oft wiederkehrende Aktionsfolgen auf spezielle „Shortcut“-Tasten legt.

Beispiel: In einem Betriebssystem kann man Faxe verschicken, indem man als Drucker das Fax angibt. Der übrige Ablauf entspricht dem normalen Ausdrucken, allerdings muß man noch eine Faxnummer eingeben. Nachdem der Benutzer diese Handlungen einigemale ausgeführt hat, erkennt das System diesen Ablauf als neues Muster und fragt den Benutzer, ob er diesem Ablauf eine Bezeichnung zuweisen möchte beziehungsweise kann. Nachdem der Benutzer angegeben hat, daß es sich um das „Faxen eines Dokuments“ handelt, kann er dem Betriebssystem entweder den Auftrag geben: „Dokument faxen“ oder das System kann die Aufgabe selbständig zu Ende führen, sobald der Benutzer die ersten Schritte ausgeführt hat.

4.4.4 Intelligenter Server

Ein (Proxy-) Server dient dazu, ein kleineres lokales Netz mit schnellen Datenwegen von einem größeren, aber vielleicht viel langsameren Netz zu entkoppeln. So müssen Anfragen aus dem lokalen Netz nicht in jedem Fall an das größere weitergereicht werden; in vielen Fällen reicht es aus, diese Anfragen aus dem Puffer des Servers zu bedienen.

Ein solches System könnte nun beispielsweise Muster in der Reihenfolge der angefragten HTML-Seiten erkennen und bei erneuten Anfragen dieser Seiten alle Folgeseiten (entsprechend dem gefundenen Muster) bereits aus dem größeren Netz anfordern (sofern sich nicht noch im Pufferspeicher vorhanden sind), bevor der Benutzer selbst sie benötigt. Damit ist der Benutzer nicht mehr so stark von der Übertragungskapazität des großen Netzes abhängig.

Da Proxyserver im allgemeinen viele Benutzer bedienen, aber naturgemäß nur eine beschränkte Pufferkapazität besitzen, könnte durch vorausschauende Anforderungsstrategien die Wartezeit auf Daten verringert werden.

4.4.5 Sicherheitsüberwachung in Netzwerken

In jüngster Zeit werden immer mehr Rechner in Netzwerkumgebungen eingesetzt, und diese immer häufiger auch an das Internet angeschlossen. Das Internet ist ein sehr offenes System in welchem relativ freier Zugang zu den angeschlossenen Rechnern besteht. Die Wartung und Pflege eines vernetzten Systems verlangt Fähigkeiten, die häufig gerade in kleinen Firmen die Kenntnisse des Systemverwalters übersteigen.

Ein System, welches lernen könnte, welche Aktionen die Stabilität des Rechnerverbundes gefährden, wäre in der Lage, bereits vor Eintreten eines kritischen Zustandes das Bedienungspersonal zu warnen. Verbunden mit einem System, welches von seinen Benutzern Handlungsabläufe lernen kann, könnte ein solches System eventuell mit wenig oder keiner Hilfe des Systemverwalters diese Probleme selbständig bewältigen oder zumindest ungeübtes Personal anleiten.

Wenn man ein solches System bei einem guten Systemverwalter (zum Beispiel in der Entwicklungsfirma des Betriebssystems) „in die Lehre schicken“, es also an einem Ort, wo häufig Ausnahmesituationen auftreten, auf welche dann professionell reagiert wird, trainieren würde, könnte seine Mustersammlung zur Unterstützung und Anleitung an andere Systemverwalter weitergegeben werden.

Verbindet man viele solcher Systeme, die miteinander dann gelernte Muster austauschen und ihrerseits aus diesen Mustern weiterlernen, so kann man innerhalb von relativ kurzer Zeit zu verhältnismäßig starken, autarken Systemen kommen.

Literaturverzeichnis

[1] D. Angluin: *Finding Patterns Common to Set of Strings*, Proc. 1th Annual ACM Symposium on Theory of Computing, 1979, 130–141

[2] D. Angluin: *Inductive Inference of Formal Languages from Positive Data*, Inform. Control, 45, 1980, 117-135

[3] S. Arikawa, T. Shinohara, S. Shiraishi, Y. Tamakoshi: *SIGMA — An Information System for Researchers Use*, Bull. Informatics and Cybernetics, 20, 1982, 97-114

[4] E. Y. Shapiro: *Inductive Inference of Theories From Facts*, Res. Rept. 192, Dept. Comp. Sci., Yale Univ., 1981

[5] T. Shinohara: *Polynomial Time Inference of Pattern Languages and Its Application*, Proc. 7th IBM Symp. on Math. Found. Comp. Sci., 1982

[6] T. Shinohara: *Polynomial Time Inference of Extended Regular Pattern Languages*, Proc. SSE Kyoto Symp., 1982

Kapitel 5

Einführung in die Agententheorie

Stefan Albus

Übersicht

Dieser Vortrag gibt eine Einführung in die Agententheorie. Anhand von Beispielen und durch die Erörterung der Eigenschaften eines Agenten, wird zunächst versucht, dem Hörer eine Vorstellung davon zu geben, was ein Agent überhaupt ist. Das daran anschließende Kapitel über Agenthentheorien beschäftigt sich mit der Spezifikation und der Modellierung von Agenten. Agentenarchitekturen stellen den Schritt von der Spezifikation zur Implementierung dar. Sie kümmern sich um das Problem, wie man die komplexen Eigenschaften und Fähigkeiten eines Agenten angemessen auf einer real existierenden, endlichen Maschine realisieren kann. Das Spektrum der Implementierungen reicht dabei von rein reaktiven Systemen bis hin zu reflexiven, wissensbasiert arbeitenden Architekturen. Das letzte Kapitel stellt kurz die Agentensprache TELESCRIPT vor. Agentensprachen sind Syteme, die zum Programmieren von Agenten dienen und deren Entwicklung vereinfachen sollen.

5.1 Einleitung

Um den Lesern überhaupt eine Vorstellung davon zu geben, was alles unter den Begriff *Agent* fällt, beginnt dieser Artikel mit einigen Beispielen von Agenten.

- Nach dem Einloggen in den Rechner präsentiert der persönliche digitale Assistent (PDA) eine Liste der eingetrofffenen E-mails, die er nach ihrer Wichtigkeit sortiert hat. Ebenfalls wartet er mit einer Auswahl von news-Articeln aus dem Internet auf, von denen der PDA glaubt, daß sich der Benutzer für ihren Inhalt interessiert. (fiktives Beispiel)
- Nachdem der PDA Kontakt mit dem WWW-Servern verschiedener Fernsehanstallten aufgenommen hat um das aktuelle Fernsehprogramm abzurufen, erstellt er eine kleine Auswahl von Magazin-Sendungen und Spielfilmen, die an diesem Abend ausgestrahlt werden und den Vorlieben des Benutzers entsprechen. (fiktives Beispiel)
- Für die bevorstehende Auslandsreise durchsucht ein passender Agent die zu dieser Reise passenden Angebote der Fluglinien und kehrt mit einer Liste der preiswertesten Anbieter zurück. Nachdem der Benutzer sich für ein Angebot entschieden hat, tätigt der Agent die entsprechende Reservierung. Aus seiner Datenbank für Reisevorbereitungen erfährt er, daß für das geplante Reiseziel Schutzimpfungen empfohlen werden. Deshalb nimmt er über das Netzwerk Kontakt mit dem digitalen Terminplaner des Hausarztes auf und handelt mit Hilfe des eigenen Terminkalenders einen geeigneten Termin für die Impfung aus. (fiktives Beispiel)
- Der Personal Portfolio Tracker, ein Agent der auf einem externen Server läuft, überwacht die Börsenkurse der zuvor vom Benutzer genannten Aktien. Nach einem vom Benutzer definierten Informationsprofil bewertet er die Aktienkurse und versendet eine EMail an den Benutzer, sobald Gewinn oder Verlust droht. (Demonstrationsanwendung von Telescript)
- Am Flughafen von Sydney unterstützt eine Hundertschaft von Agenten die Flugsicherung. Während eine Gruppe von Agenten die einzelnen Flugzeuge überwacht, kümmern sich andere Agenten um die Wetterlage, überwachen die Flugbahnen und legen die Start und Landereihenfolgen fest. (Dies System befindet sich zur Zeit (März 96) im Test und wird parallel zur regulären Flugsicherung betrieben. [4])

Die oben genannten Systeme werden Agenten genannt. Den Begriff Agent genau zu definieren ist mindestens genauso schwer wie eine Definition für den

Begriff Intelligenz anzugeben, vorallem wenn es dann darum geht, eine Definition zu finden, die von allen an diesem Thema arbeitenden Personen und Gruppen akzeptiert wird und keinen ausschließt. Aus diesem Grund wird auf eine exakte Definition des Begriffs verzichtet und es folgt im nächsten Kapitel eine Beschreibung der Eigenschaften, die einen Agenten kennzeichnen.

5.2 Eigenschaften

Die folgenden Eigenschaften lassen sich bei Agenten finden. Dabei muß ein Agent nicht immer alle Eigenschaften aufweisen. Vielmehr sind einige Punkte wie zum Beispiel die Mobilität optional.

Hauptmerkmale: (nach [7])

Autonom: Ein Agent kennt seine Fähigkeiten und hat Kontrolle über seine Handlungen und kann daher ohne direkten Eingriff seines Benutzers oder anderer Menschen tätig werden.

Kommunikativ: Agenten interagieren mit anderen Agenten (und möglicherweise Menschen) mittels einer Art von Agenten-Kommunikations-Sprache. Stößt ein Agent an die Grenzen seiner Fähigkeiten kann er somit Kontakt zu anderen Agenten aufnehmen und deren Fähigkeiten in Anspruch nehmen. Idealerweise wird dadurch die Komplexität eines einzelnen Agenten geringer als wenn der Agent alle Leistungen selbst aufbringen müßte.

Reaktiv: Agenten nehmen ihre Umgebung wahr und reagieren in begrenzter Zeit auf Änderungen in ihr. Die Umgebung kann dabei die reale Welt, ein Benutzer, den der Agent über eine (grafische) Benutzerschnittstelle wahrnimmt, eine Gruppe anderer Agenten, das Internet, andere derartige Sachen oder eine Kombination davon sein. Außerdem sollte er in der Lage sein, mit unvorhergesehenen Ereignissen fertig zu werden, was besonders für das Überleben in dynamischen Umgebungen wie dem Internet wichtig ist.

Eigeninitiative: Agenten reagieren nicht nur einfach stur auf ihre Umwelt, sondern sie zeigen auch zielgerichtetes Verhalten, indem sie zum Beispiel die Initiative ergreifen, und somit übergeordnete Ziele selbständig und aktiv verfolgen.

weitere (eher optionale) Eigenschaften:

Mobilität: Mobile Agenten sind nicht an einen festen Ort gebunden, sondern in der Lage herumzureisen, in dem sie zum Beispiel mittels eines elektronischen Netzwerkes von einem Rechner zum Nächsten wandern.

Kooperativ: Damit ist die Annahme gemeint, daß verschiedene Agenten keine gegensätzlichen Ziele haben und sie sich somit nicht gegenseitig behindern.

Rational: Ein Agent handelt rational, wenn er auf eine Weise agiert, mit der er seine Ziele möglichst gut erreichen kann bzw. wenn er glaubt, mit seinen Aktionen sein Ziel möglichst gut zu erreichen (Manchmal unternimmt man halt Aktionen in der festen Überzeugung damit seine Aufgaben gut zu lösen, um dann im nachhinein das Gegenteil festzustellen, daß man sich nämlich von seinem Ziel entfernt hat.) und keine sonstigen Aktionen unternimmt.

Da, wie bereits gesagt, nicht alle Eigenschaften zwingend in einem Agenten auftreten müssen, fallen viele Systeme unter diesen Begriff. Eine einfache Art, einen Agenten zu verwirklichen, ist ein UNIX-Prozeß der einige der oben angegebenen Eigenschaften erfüllt. Desweiteren tendieren solche neuen Begriffe, wenn sie sich erst mal als innovativ und als Inbegriff des 'Guten' herumgesprochen haben, zur inflationären Verwendung. Ähnlich wie heute, wo alles, was neu und gut ist, mindestens objektorientiert sein muss, wird in Zukunft wohl eine Flut an agentenorientierter Software kommen. (Es klingt halt in der Werbung besser, wenn man seinem Produkt dieses Attribut hinzufügen kann, denn der Begriff 'Objektorientiert' ist heute ja schon fast abgenutzt.)

5.3 Agententheorie

Viele Leute betrachten eine Agententheorie als eine Spezifikation für einen Agenten und sie entwickeln Formalismen, um die Eigenschaften eines Agenten modellieren zu können. Da es hierbei um Modellierung geht, hat das Gebiet eine gewisse Nähe zum Software Enineering, da es sich hier im Prinzip um die Spezifikation von Software handelt.

Agenten werden meist mit Ausdrücken beschrieben wie Wissen, Glauben und Intention. Agenten haben ein gewisses Wissen über ihre Umwelt, und sie machen Annahmen über ihre Umgebung (Glauben), die natürlich auch mal falsch sein können. Desweiteren sollen ihre Aktionen eine gewisse Aufgabe erfüllen, das heißt , sie haben irgendeinen näheren Zweck (Intention). Diese (doch recht schwammigen) Begriffe müssen nun irgendwie (möglichst konkret) modelliert werden.

5.3.1 Possible World Semantik

Bei diesem Modell werden die Annahmen, die ein Agent über die Umwelt macht, als eine Menge von möglichen Welten (*possible Worlds*) beschrieben. Dies soll am folgendem Beispiel verdeutlicht werden. Man stelle sich einen Agenten vor, der ein Kartenspiel, wie z.B. Poker, spielen soll. Die Menge aller möglichen Kartenverteilungen stellt hier nun die Menge aller Welten dar. Üblicherweise kennt der Agent nicht die aktuelle Kartenverteilung, aber er kennt die Karten, die er besitzt, und, je nachdem welche Variante der Regeln man benutzt, kennt er vielleicht ein paar Karten, die seine Mitspieler halten. Also kann er aus der Menge aller Kartenverteilungen diejenigen als unmöglich herausstreichen, die diesem Wissen nicht entsprechen. Übrig bleibt eine kleinere Menge von möglichen Verteilungen (Welten), die nun das darstellen, was der Agent über seine Umwelt glaubt oder weiß.

Die Semantik für ein solches Modell wird im allgemeinem als eine Modallogik definiert. Eine Modallogik ist zunächt einmal grundsätzlich die klassische Aussagelogik, die um zwei zusätzliche Operatoren erweitert wird: $\Box$ (unbedingt, notwendigerweise) und $\Diamond$ (vielleicht) . Sei $P = \{p, q, \ldots\}$ die Menge der Aussagevariablen. Dann wird die Menge F der Formel induktiv definiert durch:

1. $P \subseteq F$
2. $\varphi, \psi \in F \Rightarrow \neg\varphi, \varphi \vee \psi \in F$ ($\neg$ und $\vee$ haben die normale Bedeutung)
3. $\varphi \in F \Rightarrow \Box\varphi, \Diamond\varphi \in F$

Auf den möglichen Welten wird nun eine Erreichbarkeitsrelation eingeführt. Die Formel $\Box\varphi$ ist genau dann wahr, wenn φ in jeder Welt wahr ist, die von der jetzigen Welt erreicht werden kann (gelesen: unbedingt φ). Die Formel $\Diamond\varphi$ ist genau dann wahr, wenn φ in mindestens einer Welt wahr ist, die von der jetzigen Welt erreicht werden kann (gelesen: vielleicht φ).

Diese Notation kann man nun auf den pokerspielenden Agenten anwenden. Sei φ die Aussagevariable mit der Bedeutung: Der Agent hält die Karte $\heartsuit 7$. Weiterhin habe ψ die Bedeutung: Ein anderer Mitspieler hält die Karte $\heartsuit 7$. Wenn der Agent nun wahrnimmt, daß er diese Karte besitzt, kann er nach den üblichen Spielregeln für Poker davon ausgehen, daß momentan kein weiterer Mitspieler ebenfalls solch eine Karte besitzt. Für die aktuelle Welt gilt also φ und $\neg\psi$. Die Erreichbarkeitsrelation für unsere Welten gibt nun an, welche Kartenverteilung von einer bestehenden Verteilung erreicht werden kann und welche nicht. Ausgehend von der jetzigen Situation in der φ gilt folgt, daß kein Mitspieler die Karte $\heartsuit 7$ im weiterem Spielverlauf besitzen kann, das heißt, in keiner erreichbaren Welt gilt ψ oder kurz: $\Box\neg\psi$.

In der aktuellen Spielsituation hat der Agent aber noch die Möglichkeit, die Karte gegen eine andere auszutauschen, d.h. es existiert noch eine mögliche Spielsituation (eine erreichbare Welt), in der er die Karte $\heartsuit 7$ nicht besitzt ($\neg\varphi$). Also gilt: $\Diamond\neg\varphi$.

Zwei grundlegende Eigenschaften einer Modallogik sind das Axiom K und die Notwendigkeitsregel:

Axiom K: $\Box(\varphi \Rightarrow \psi) \Rightarrow (\Box\varphi \Rightarrow \Box\psi)$

Notwendigkeitsregel: wenn φ allgemeingültig ist, dann ist auch $\Box\varphi$ allgemeingültig

Die meisten weiteren Eigenschaften einer Modallogik folgen aus den Eigenschaften der Erreichbarkeitsrelation, so daß je nachdem, welche Eigenschaften man haben will, eine andere Modallogik entsteht. Ein Beispiel dafür ist die Modallogik *T4* , deren Erreichbarkeitsrelation reflexiv und transitiv ist:

T: $\Box\varphi \Rightarrow \varphi$ (bei reflexiver Relation)

4: $\Box\varphi \Rightarrow \Box\Box\varphi$ (bei transitiver Relation)

$\Box\varphi$ wird auch ausgeprochen als: man weiß, das φ gilt. Axiom 4 wird auch Selbstbetrachtung genannt (wenn man etwas weiß, dann weiss man auch, daß man es weiß, der Agent ist sich also seines Wissens bewußt). Axiom T wird als Wissensaxiom bezeichnet, da es besagt, das das, was man weiß, wahr ist (im Gegensatz dazu kann das , was ein Agent über die Umwelt glaubt, auch falsch sein)).

Mit diesem Modell kann nun das Wissen des Agenten modelliert werden. Was aber weiterhin noch fehlt, ist die Modellierung der Intentionen. So steht bei dem pokerspielenden Agenten noch die Frage aus, welche erreichbaren Welten für ihn günstig sind und auf welche dieser Welten der Agent dann letztendlich zusteuern soll.

Die Notwendigkeitsregel führt dazu, daß der Agent alle gültigen Formeln kennt und somit auch alle Tautologien. Da es davon aber unendlich viele gibt, bedeutet dies, daß der Agent eine unendliche Anzahl an Wissensstücken kennt. So etwas bereitet aber große Schwierigkeiten, es auf einer endlichen, begrenzten Maschine zu implementieren (vor allem wenn der Agent dann auch noch effizient arbeiten soll und und die Eigenschaft *Reaktiv* dann auch noch fordert, daß der Agent innerhalb *begrenzter* Zeit auf seine Umwelt reagiert, während heutige automatische Theorembeweiser dazu neigen, sehr viel Zeit zu benötigen). Dies führt dazu, das viele Forscher nach Alternativen zu diesem Modell suchen.

5.3.2 Kommunikation

Kommunikative Äußerungen von Agenten sind Aktionen gleichzusetzen, da sie ähnliche Eigenschaften haben wie physikalische Aktionen. Das versenden von Nachrichten kann den Zuhörer (sei es ein Mensch oder ein anderer Agent) in einen anderen Zustand versetzen und ihn zu weiteren Aktionen, seien dies nun physikalische Aktionen oder das weitere Versenden von Nachrichten, veranlassen. Damit bewirken sie, genauso wie physikalische Aktionen, eine Änderung in der Umwelt. In der gleichen Weise wie ihr physikalisches Gegenstück können sie auch fehlschlagen. Nachrichten können evtl. verloren gehen, oder der Empfänger reagiert auf eine andere Weise als erwartet. Im allgemeinen kann man zwei Arten von Nachrichten unterscheiden. Repräsentative Nachrichten informieren den Zuhörer über Ereignisse oder Zustände, während direktive Nachrichten Aufträge oder Befehle übermitteln.

5.4 Agentenarchitekturen

Agentenarchitekturen stellen den Schritt von der Spezifikation zur Implementierung der Agenten dar. Die Architektur gibt dabei an, wie man einen Agenten in eine Menge von Komponenten zerlegt und in welcher Weise dann diese Komponenten miteinander interagieren. Die Komponenten und ihre Interaktionen müssen dabei das Problem lösen, wie die Sensordaten (die Wahrnehmungen des Agenten) und der innere Zustand des Agenten die Aktionen und Zustandsänderungen des Agenten bestimmen.

Das Spektrum der Agentenarchitekturen reicht von reflexiven, wissensbasiert arbeitenden Agenten bis hin zu rein reaktiven Systemen.

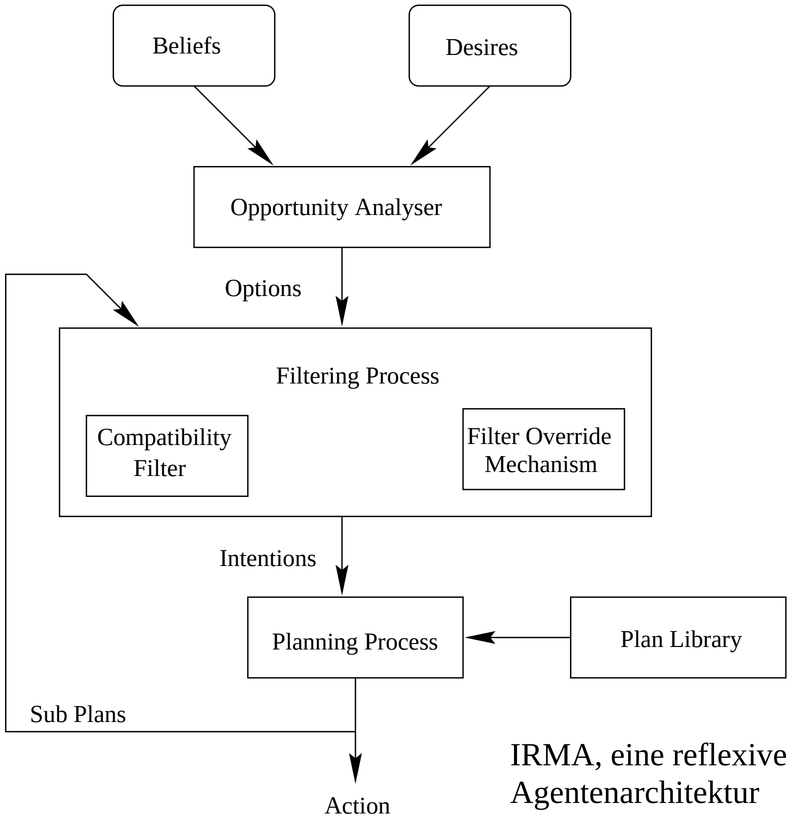
5.4.1 Reflexive Architekturen

Ein reflexiver Agent besitzt ein symbolisches Model seiner Umwelt und dieses Wissen wird im Agenten auch explizit repräsentiert. Entscheidungen über die Aktionen, die er ausführen will, trifft der Agent mittels logischer Schlußfolgerunge, die auf Pattern-Matching und symbolischen Manipulationen innerhalb dieses Modells basieren.

Bei diesem Ansatz ergeben sich zwei Probleme: Das erste Problem ist, wie man die reale Welt in eine passende Beschreibung umsetzt. Das zweite Problem ist, wie man nun die Information über eine komplexe Umwelt auf dem Rechner repräsentiert, so daß der Agent seine logischen Schlußfolgerungen in akzeptabler Zeit treffen kann.

IRMA

IRMA (*Intelligent Resourcebounded Maschine Architecture*) ist ein Beispiel für eine reflexive Architektur. Sei besteht aus vier symbolischen Datenstrukturen: eine Planungsbibliothek (*plan library*) und eine explizite Repräsentation von Annahmen über die Umwelt (*beliefs*), von Wünschen (*desires*) und von Intentionen (*Intentions*).



IRMA, eine reflexive Agentenarchitektur

Die Funktionsweise kann man sich an einem Beispiel klar machen. Der Agent sei leidenschaftlicher Tennisspieler, also befindet sich bei den 'Desires' der Eintrag: Spiele Tennis so oft wie möglich. Von seinen Sensoren erfährt er, daß für das Wochenende gutes Wetter vorhergesagt wird. In der Wissensbasis ('Beliefs') befindet sich dann der Eintrag: Gutes Wetter am Wochenende. Der 'Opportunity Analyser' generiert daraus die Option: Spiele Tennis am Wochenende. Die erzeugten Optionen werden anschließend erst einmal gefiltert, bevor sie zu einer Intention werden können. Der 'Compability Filter' überprüft, ob die Option nicht mit einer anderen Intention kollidiert. Da IRMA zu den beharrlichen Agenten zählt, werden neue Ideen erst einmal verworfen, wenn sie den aktuellen Absichten des Agenten entgegenlaufen. Stehen also für das Wochenende bereits andere Pläne an, so wird das Tennisspielen gestrichen. Ist der Agent allerdings ein fanatischer Tennisspieler, so kann er mit dem 'Filter Override Mechanismus' dem Tennisspielen den Vorzug vor anderen Intentionen geben und somit das beharrliche Verhalten übergehen. Ist das Tennisspielen somit zur Intention geworden, so setzt die Planungskomponente dies nun in die Tat um. Die Planungsbibliothek enthält dabei Pläne aus früheren Zeiten. So werden aus dem Tennisspielen zunächst die Teilpläne 'Tennispartner suchen' und 'Tennisplatz reservieren' generiert, die, genau wie die Optionen auch, wieder gefiltert werden, um zu sehen, ob sie nicht anderen Intentionen zuwider laufen. [3]

5.4.2 Reaktive Architekturen

Reaktive Agentenarchitekturen enthalten im Gegensatz zu den Reflexiven kein zentrales symbolisches Modell ihrer Welt und haben deshalb auch nicht das Problem, komplexe, symbolische Beweise führen zu müssen. Dadurch reagieren sie im allgemeinem wesentlich schneller auf Ereignisse in ihrer Umgebung als ihre reflexiven Kollegen.

Agent Network Architecture

Diese Agentenarchitektur besitzt eine gewisse Ähnlichkeit zu neuronalen Netzwerken. Ein Agent besteht dabei aus einer Menge von Modulen, die spezielle Zuständigkeiten haben. Dabei wird jedes Modul durch eine Reihe von Vor- und Nachbedingungen sowie einem Aktivierungslevel spezifiziert. Die Aktionen, die die Module bei ihrer Aktivierung ausführen, werden direkt kodiert und sind im Agenten nicht explizit repräsentiert.

Ein Modul i ist somit ein Tupel $(c_i, a_i, d_i, \alpha_i)$, wobei c_i eine Liste von Vorbedingungen ist, a_i (add-list) und d_i (delete-list) Listen von Propositionen sind, die die erwarteten Effekte der Module darstellen sollen, (Die Propositionen aus a_i werden wohl gelten und die aus d_i nicht.) und α_i der Aktivierungslevel ist.

Welches Modul zur Ausführung kommt, bestimmt sich wie folgt: (i) Das Modul muß *ausführbar* sein, daß heißt, alle seine Vorbedingungen sind erfüllt, und (ii) der Aktivierungslevel überschreitet eine gewisse Grenze. (iii) Das Modul besitzt den größten Aktivierungslevel aller Module, die (i) und (ii) erfüllen. Nach der Ausführung eines Moduls wird sein Aktivierungslevel auf Null gesetzt. Je höher also der Level ist, desto wahrscheinlicher ist es, daß ein ausführbares Modul aktiviert wird.

Der Aktivierungslevel eines Modules bestimmt sich durch die folgenden Regeln:

Aktivierung durch die Umgebung: Wenn eine der Vorbedingungen aus

c_i erfüllt ist, wird der Aktivierungslevel erhöht.

Aktivierung durch Ziele: Wenn ein Modul die *globalen Ziele* des Agenten anstrebt, daß heißt, ein Element der *add-list* gehört zu den *globalen Zielen*, wird der Aktivierungslevel erhöht.

Deaktivierung durch bereits erfüllte Ziele: Wenn ein *globales Ziel* des Agenten bereits erfüllt ist und steht dieses Ziel in der *delete-liste* eines Moduls, so wird der Aktivierungslevel dieses Moduls reduziert.

Aktivierung der Vorgänger: Ein *nicht ausführbares* Modul erhöht den Aktivierungslevel der Module, die eins der noch nicht erfüllten Prädikate dieses Moduls in ihrer *add-Liste* haben. Der Betrag der Änderung entspricht einem festen Prozentsatz des eigenen Aktivierungslevels. Module fördern somit die Module, die ihre fehlenden Vorbedingungen herstellen.

Aktivierung der Nachfolger: Ein *ausführbares* Modul x erhöht den Aktivierungslevel eines Moduls y , wenn es ein Prädikat $P \in a_x \cap c_y$ gibt, das nicht erfüllt ist. Der Betrag der Änderung entspricht einem festem Bruchteil der eigenen Aktivierung.

Deaktivierung durch Konflikte: Ein Modul mit der erfüllten Vorbedingung P erniedrigt den Aktivierungslevel aller Module, die P in ihrer *delete-liste* haben. Es will somit verhindern, das andere Module seine Aktivierung zurücksetzen. Der Betrag der Änderung ist auch hier wieder vom eigenen Aktivierungslevel abhängig.

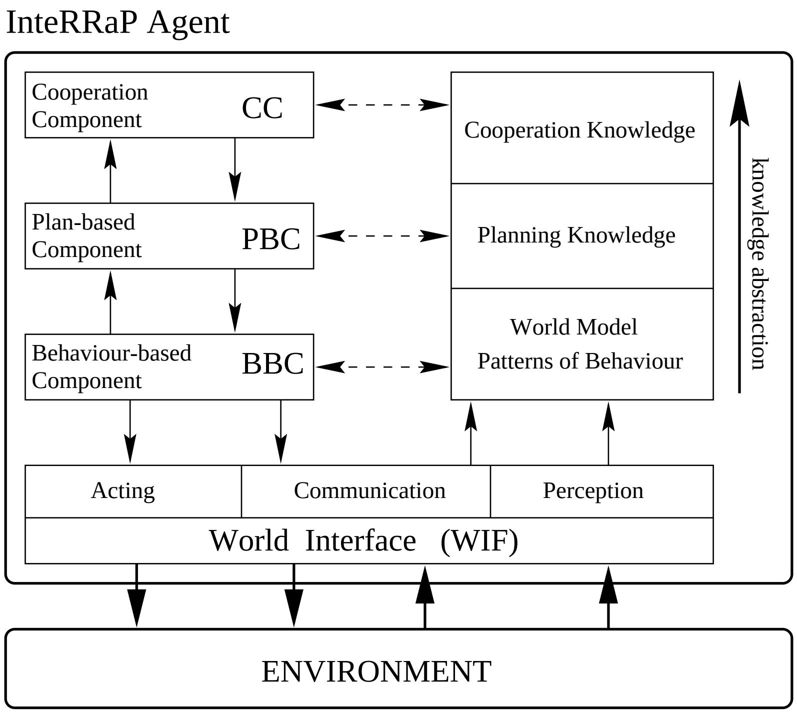
Der Hauptaufwand besteht nun darin, permanent die Aktivierungslevel der einzelnen Module zu bestimmen. [2]

5.4.3 Hybride Architekturen

Hybride Architekturen vereinen nun beides. Sie bestehen aus einem reflexiven Teil, der ein symbolisches Modell der Welt enthält, mittels dessen man logische Entscheidungen treffen kann, und einem reaktiven Teil, der auf Ereignisse in der Umwelt reagieren kann, ohne dabei vorher komplexe Beweise für die auszuführenden Aktionen durchführen zu müssen. Oft wird dem reaktiven Teil Vorrang vor dem Reflexivem eingeräumt, so daß der Agent schnell und direkt auf wichtige Ereignisse in seiner Umwelt reagieren kann. Solche Architekturen sind meist hierarchisch aufgebaut, wobei die unteren Schichten Zugang zu den mehr oder weniger rohen Sensordaten haben und auf spezielle Situationen direkt Aktionen veranlassen können, während sich die höheren Schichten um die langfristigen Ziele und Aufgaben des Agenten kümmern.

5.4.4 InteRRap

InteRRap [5] ist eine solche hybride Architektur. Sie besteht aus fünf Hauptkomonenten: 'world interface, behaviour-based component, plan-based component, cooperation component und knowledge base.



Das 'world interface' (WIF) enthält die Fähigkeiten, die Umwelt zu erkennen (*perception*) und Aktionen in ihr auszuführen (*acting*) und die Einrichtungen zur Kommunikation.

Die verhaltensbasierte Komponente ('BBC, behaviour-based component') realisiert das *reaktive* Verhalten des Agenten und basiert auf Verhaltensmustern ('patterns of behaviour'), welche in der *knowledge base* abgelegt sind und die es ihm erlauben, Routineaufgaben effektiv und ohne aufwendige Planung zu erledigen. Ein Verhaltensmuster besteht aus einer Vorbedingung, die angibt, wann dieses Muster in Kraft treten soll, einem Exekutivteil, der die auszuführenden Aktionen enthält, einigen Bedingungen, mit denen man entscheiden kann, ob die Ausführung erfolgreich war, sowie einer Nachbedingung. Da zu einem Zeitpunkt mehrere Muster aktiv sein können, ist ein Schedulingmechanismus für die (quasi) gleichzeitige Ausführung verschiedener Exekutivteile nötig. Die elementaren Anweisungen bei der Ausführung sind dabei die primitiven Fähigkeiten des World-Interfaces, also das Veranlassen einfacher Aktionen sowie das Ablesen der Sensoren, Aufrufe der Planungskomponente mit Übergabe eines Ziels, sowie das Aktivieren weiterer

Muster. Diese elementaren Anweisungen können nun zu Blöcken zusammengefaßt werden und in Schleifen und bedingten Anweisungen auftauchen.

Die Planungskomponente ('PBC, plan-based component') besteht aus einem Mechanismus, der Pläne für diesen einen Agenten liefert. Diese Pläne sind hierarchisch aufgebaut und können somit wiederum aus Teilplänen bestehen. Desweiteren enthalten sie neu generierte Verhaltensmustern für die BBC sowie primitive Aktionen. Diese Komponente greift dabei auf eine Menge von Plänen zurück, die in der *knowledge base* enthalten sind.

Die Kooperationskomponente ('CC, cooperation component) ist in der Lage, Gemeinschaftspläne für eine Multi-Agenten Umgebung zu entwicken, um die Aktionen mehrerer Agenten zu koordinieren. Sie wird von der Planungskomponente aktiviert, wenn diese feststellt, daß dieser Agent eine Aufgabe alleine nicht zufriedenstellend lösen kann.

Die Wissensbasis (knowledge-base), als fünfte Komponente, ist in drei Schichten unterteilt. Das *world model* enthält Aussagen über die Umwelt auf einfacher Ebene. Das *mental model* enthält selbsterkanntes Wisssen und Information über den mentalen Zustand des Agenten (Ziele, Pläne und Intentionen). Das *social model* repräsentiert seine Annahmen über andere Agenten. Zugriff auf die einzelnen Schichten ist nur von den höheren Schichten auf die Tieferen möglich, d.h die Verhaltenskomponente kann nur auf das *world model* zugreifen, während der Kooperationskomponente das gesamte Wissen zur Verfügung steht.

5.5 Agentensprachen

Agentensprachen sind Systeme, die zur Programmierung von Agenten dienen. Sie sollen die Entwicklung von Agenten wesentlich vereinfachen, indem sie passende Konzepte bereitstellen. Während die meisten Systeme nur von einigen wenigen Forschern benutzt werden und hauptsächlich der Forschung dienen, ist Telescript das wohl erste kommerzielle System, welches im folgendem kurz vorgestellt wird.

5.5.1 TELESCRIPT

Telescript [1] [4] [6] ist eine objektorientierte Sprache, die mit Hilfe von Klassen eine Reihe von Abstraktionen anbietet, um einen konzeptionellen Rahmen für verteilte, agentenbasierte Systeme bereitzustellen. Konkrete Anwendungen (wie Datenbanken, Terminabsprache, Platzreservierung, ...) brauchen ergänzende Protokolle. Alles, was mit der Ausführung von Programmen zu tun hat, ist als Objekt abgebildet. Es existieren Abstraktionen für Plätze, Agenten, Reisen, Treffen, Verbindungen und Verfügungsrechte. Plätze sind logische Rechnerknoten, die im allgemeinen einen Dienst anbieten, der dort von einem residentem Prozeß erbracht wird. Agenten sind mobile Prozesse, die nun von einem Platz zum nächsten Reisen können. Agenten umfassen eine Prozedur und den Zustand ihrer Ausführung. Ein Ticket besteht aus einem Ziel, den Weg dorthin und der Zeitdauer einer Reise und wird von der Methode go benötigt, die eine Bewegung des Agenten auslöst, wobei der Programmcode und der momentane Zustand des Programms über das Netz zu einem anderen Platz transportiert wird, wo dann die Abarbeitung des Programms fortgesetzt wird. Zum Schutz gegen böswillige Agenten und gegen Überlastung können Ressourcen (Speicher, Rechenzeit) für Prozesse (also für Agenten und dienstanbietende Plätze) begrenzt werden. Außerdem hat jeder Agent und jeder Platz einen Besitzer, der für die Handlungen seiner Software zur Verantwortung gezogen werden kann. Agenten sind fähig miteinander zu kommunizieren. Entweder können sie eine Kommunikationsverbindung über das Netz aufbauen, oder sich an einem gemeinsamen Platz treffen.

Das System besteht aus vier Komponenten: *Telescript language*, die eigentliche Programmiersprache, *Telescript engine*, eine Art virtuelle Maschine für die Agenten, ein *protocol set*, der hauptsächlich dazu dient, Agenten für den Transport zu kodieren und anschließend wieder zu dekodieren und eine Komponente mit *Softwaretools*, die die Entwicklung von Applikationen unterstützt.

Literaturverzeichnis

[1] Reiner Fischbach. Softwaretechnologie, Intelligent Agents bei der Arbeit. *iX Multiuser Multitasking Magazin*, pages 134+, April 1996.

[2] Pattie Maes. How To Do the Right Thing. http://lcs.www.media.mit.edu/groups/agents/Publications/Pattie/-consci/connection-science.html.

[3] Martin Römer, Bernhard Quendt, Peter Stenz. Agent IRMA denkt mit. *c't magazin für computertechnik*, page 160, March 1996.

[4] Martin Römer, Bernhard Quendt, Peter Stenz. Autopiloten fürs Netz. *c't magazin für computertechnik*, pages 156+, March 1996.

[5] Müller, J. P., Pichel, M., Thiel, M. Modelling reactive behaviour in vertically layerd agent architectures. In M. Wooldridge and N. R. Jennings, editors, *Intellligent Agents, Theory, Architectures, and Languages*, pages 261–276. Springer-Verlag, Heidelberg, Germany, 1994. INF 150/412-94.

[6] *Online-Dokumentation zu Telescript.* http://www.genmagic.com/Telescript/awt/docs.html.

[7] Michael Wooldridge. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995. http://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.html.

Kapitel 6

Softbots – Agenten interagieren mit Software-Umgebungen

Michael Schlindwein

Übersicht

Genau wie physische Roboter für die Produktion oder als autonome mobile Roboter für den Einsatz in verschiedenen „Hardware"-Umgebungen entwickelt wurden, gibt es Programme, die aufgrund ihrer Funktionalität als Software-Roboter – kurz **Softbots** – angesehen werden können. Diese unterstützen den Benutzer, indem sie in Software-Umgebungen wie Betriebssystemen, Anwendungsprogrammen oder auch dem Internet agieren, für den Menschen Aufträge ausführen und ihm komplexe Arbeiten abnehmen. Gerade bei der Mächtigkeit heutiger Betriebssysteme und Anwendungsprogramme einerseits und dem immer populärer werdenden und stetig wachsenden Internet andererseits werden solche Softbots immer wünschenswerter.

Software-Umgebungen und darin agierende Softbots bieten aus Sicht der KI-Forschung insofern Vorteile, als sich mit ihnen bestimmte Problemstellungen relativ kostengünstig und unter Vermeidung vieler Probleme, wie sie physische Roboter in physischen Umgebungen mit sich bringen, erforschen lassen.

Im folgenden werden kurz die wesentlichen Eigenschaften von Softbots aufgezeigt und auf die Interaktionsmöglichkeiten mit Software-Umgebungen eingegangen.

6.1 Softbots

6.1.1 Der Begriff „Softbot“ – Analogie zum Roboter und Einsatzmöglichkeiten

Softbot ist eine Abkürzung von **Soft**ware Ro**bot**. In der Tat gibt es viele Analogien zu einem Roboter, wie er zum Beispiel in einer Montagestraße zu finden ist oder zu einem autonomen mobilen Roboter, der sich in einer fremden Umgebung zurechtfinden soll.

Roboter [9]	**Softbot**
• „flexible Handhabungsgeräte“	• flexible Programme
• „führen komplexe Tätigkeiten eines Menschen aus“	• erledigen komplexe Aufgaben für Menschen, indem sie
• „Arbeitsweise der eines Menschen nachgebildet“	• ähnlich den Menschen Arbeitsschritte planen und ausführen
• „Effektoren zur Durchführung von Tätigkeiten“	• Kommandos, die die Umgebung verändern
• „Sensoren zur Kommunikation mit der Umgebung, insbesondere zur Aufnahme von Informationen“	• Kommandos, die Informationen über die Umgebung liefern

In [6] werden Softbots treffend so umschrieben:

> *„Ein Softbot ist ein Agent, der mit einer Software-Umgebung interagiert, indem er Kommandos ausführt und die Reaktion der Umgebung interpretiert. Die Effektoren eines Softbots sind Kommandos, die den äußeren Zustand der Umgebung ändern. Die Sensoren eines Softbots sind Kommandos, die den Softbot mit Informationen über die Umgebung versorgen.“*

Weiter verbreitet als die Bezeichnung Softbot sind die allgemeineren Begriffe **Software Agent** bzw. einfach **Agent**. Obwohl diesem Feld der Informatik, dem Paradigma der agentenorientierten Programmierung, große Zukunftsaussichten vorausgesagt werden [12], gibt es noch keine allgemein anerkannte Definition eines Agenten. In 6.1.2 werden daher stattdessen wesentliche Eigenschaften aufgeführt, die in der Literatur vertreten sind und über die teilweise auch Konsens herrscht.

Die folgende Liste enthält einige – teils synonyme, teils spezialisierte – Bezeichnungen der Begriffe ([12], [19], [11], [3], [15], [16], [7]):

- automomer Agent
- intelligenter Agent
- persönlicher Agent
- intelligenter persönlicher Agent
- persönlicher (digitaler) Assistent
- intelligentes/adaptives Interface
- (lernender) Interface-Agent
- Userbot und Taskbot
- Knowbot
- Indizier-Agent
- Netzwerk-Agent
- Betriebssystem-Agent

Weiter kann man Agenten wie in [12] in verschiedene Klassen einteilen. Relativ „dumme" Agenten führen nur relativ einfache Aufgaben nach vordefinierten Regeln aus, während „intelligentere" (Dienste erbringende) Agenten wohldefinierte (komplexe) Aufgaben aufgrund von Benutzeranfragen weitgehend selbständig erledigen. Darüber hinaus sind Agenten vorstellbar, die ihrem Benutzer eigenverantwortlich Dienste oder Informationen anbieten, das heißt, sie tun es aus sich heraus, und zwar dann, wenn es ihnen angemessen erscheint.

Aus den Bezeichnungen heraus sind schon einige Einsatzgebiete für solche Agenten zu erkennen. Sie können zur Unterstützung in komplexen Netzwerken und Betriebssystemen eingesetzt werden oder aber als Schnittstelle (Interface) zu solchen oder zu Anwendungsprogrammen. Dabei bedeutet „persönlich", daß ein Agent sich auf den jeweiligen Benutzer einstellen soll. Folgende – sicher nicht vollständige – Liste mit Einsatzmöglichkeiten zeigt das Anwendungsspektrum auf ([12], [11], [7]):

- personalisiertes Informationsmanagement
- Mail-Programme
- Terminplanung
- Benutzerschnittstellen-Design
- Computerspiele
- Entwicklungswerkzeuge
- Netzwerk-Management
- Betriebssysteme
- Internet/WWW
- Elektronischer Handel
- Auswahl von Büchern, Musik und Unterhaltung
- Management komplexer industrieller/wirtschaftlicher Prozesse (Workflow Management)

6.1.2 Eigenschaften und Fähigkeiten von Agenten

Zwar ist oben so etwas wie eine Definition für Softbots angegeben, doch da diese als Form von Software-Agenten zu sehen sind, reicht das noch nicht aus, um sie umfassend zu beschreiben. Wie schon erwähnt, gibt es keine allgemein anerkannte Definition eines Agenten, jedoch hat sich mit der Zeit eine Menge von Eigenschaften entwickelt, von denen ein Agent anerkanntermaßen eine Teilmenge an Eigenschaften haben sollte und durch die sich Agenten auch von „normalen“ Programmen unterscheiden. Es sei aber bemerkt, daß diese Unterscheidung dennoch oftmals nicht klar zu treffen ist und streitbar bleibt.

Im wesentlichen ist ein Agent ein – in gewisser Weise persistentes – Programm, an das der Benutzer Aufgaben delegieren kann, und das diese selbständig löst, indem es einen Plan entwickelt um entsprechende Ziele zu erreichen und diesen ausführt. Dabei verfolgt es Veränderungen in der Umgebung und reagiert angemessen darauf (z.B. durch Umplanung), trifft Entscheidungen, führt Aktionen aus und kontrolliert diese ([12]).

Im einzelnen sind dabei folgende Eigenschaften wichtig ([14]):

Autonomie: Ein Agent kann selbständig agieren und übt dabei auch Kontrolle über seine eigenen Aktionen aus.

- **zielorientiert:** Ein Agent nimmt Aufgaben entgegen, die ausdrücken, *was* der Benutzer will und ermittelt selbst, *wie* er diese am besten erledigt.
- **kollaborativ:** Ein Agent führt nicht einfach Anweisungen aus, sondern kann Anfragen verändern, bei Unklarheiten nachfragen oder sogar die Ausführung mancher Aufträge verweigern.
- **flexibel:** Ein Agent ist nicht „festverdrahtet“, sondern kann Aktionsfolgen – abhängig vom Zustand seiner Umgebung – dynamisch zusammensetzen.
- **selbststartend:** Ein Agent wird nicht wie normale Programme immer direkt vom Benutzer gestartet, sondern er kann Veränderungen in seiner Umgebung wahrnehmen und entscheiden, wann er agiert.

Zeitliche Kontinuität: Ein Agent ist ein kontinuierlich laufender Prozeß, der nicht einfach – einmal gestartet – eine Verarbeitung nach dem EVA-Prinzip durchführt und dann endet.

Adaptivität: Ein Agent kann sich an seinen Benutzer und auch an Änderungen in seiner Umgebung anpassen.

Kommunikativität: Ein Agent kann mit anderen Agenten oder auch Menschen kommunizieren, um Informationen zu erhalten, deren Hilfe zum

Erreichen seiner Ziele in Anspruch zu nehmen oder auch anderen bei der Durchführung ihrer Aufgaben zu helfen.

Charakter: Ein Agent hat eine wohldefinierte vertrauenswürdige „Persönlichkeit".

Mobilität: Ein Agent kann über verschiedene Systemarchitekturen und -plattformen hinweg von einer Maschine zu einer anderen wandern.

Aufgrund der Eigenschaften *Adaptivität* und *Autonomie* wird der Einsatz von KI-Techniken, vor allem Lernen und Planen, notwendig (siehe dazu 6.2.2).

Agenten sollen in Vertretung für einen Menschen autonom Aktionen ausführen und dabei eigene Ziele verfolgen und Entscheidungen treffen. Bei diesen und den weiteren Fähigkeiten, die ein Agent haben soll oder kann, spielt seine **Sicherheit** und **Verläßlichkeit** eine sehr wichtige Rolle. Weiter **muß** die **Privacy**[1] eines Benutzers **gewährleistet sein**! Dies ist umso bedeutender, als viele Agenten sich auch im Internet bewegen sollen, wo sie bei ihrem autonomen Vorgehen ohnehin schon genug Datenspuren hinterlassen können[2]. Glücklicherweise gehen sie in aktuelle Entwicklungen im Bereich der Software-Agenten oft ein; die Entwickler sehen sehr wohl die Gefahren, die die Fähigkeiten, die sie ihren Programmen geben wollen, mit sich bringen. In [15] werden folgende Sicherheitsmechanismen gefordert:
Ein Agent soll

- **sicher („safe")**
  Keine zerstörerische Änderungen an der Umgebung vornehmen!

- **sauber („tidy")**
  Die Umgebung so nah wie möglich an ihrem ursprünglichen Zustand zurücklassen!

- **wachsam („vigilant")**
  Aktionen von Menschen verhindern, die unerwünschte Konsequenzen haben!

- **sparsam („thrifty")**
  Den Gebrauch wertvoller Ressourcen einschränken!

sein.

[1] Ich verwende das englische Wort, da es keine vergleichbar ausdrucksstarke Übersetzung dafür gibt.

[2] Als Denkanstoß soll das dienen, was der relativ „dumme" Microsoft Registration-Wizard ha(e)tte bewirken können (Microsoft hat dementiert, Festplatteninhalte über das Microsoft Network zu sich übertragen zu haben.).

Es ist nicht gesagt, daß dies ausreicht, aber es ist ein erfolgversprechender Ansatz. Eine mögliche Formalisierung dieser Begriffe, die für den Einsatz in Programmen ja nötig ist, wurde angegangen, und für Sicherheit und Sauberkeit auch schon eine mögliche Lösung angegeben ([1]). Auf alle Fälle aber muß ein Agent sich so verhalten, daß der Benutzer sich im Umgang mit ihm wohlfühlen kann, wozu vor allem auch gehört, daß der Mensch die **absolute Kontrolle** über das Geschehen haben muß. Das heißt, wenn der Benutzer will, daß gewisse Aktionen (z.B. Löschen von E-Mails oder Dateien) nicht autonom oder erst nach einer Sicherheitsabfrage ausgeführt werden, so muß ihm dies möglich sein! Außerdem muß ein Agent auf jeden Fall jederzeit eine Zusammenstellung der von ihm selbständig durchgeführten Aktionen geben können.

6.1.3 Was bringen Softbots? – Vor- und Nachteile

Softbots – oder allgemein Agenten – haben aufgrund ihrer Fähigkeiten, die ihnen oft durch KI-Techniken verliehen werden, Vorteile gegenüber herkömmlichen Programmen. Auf der anderen Seite unterliegen Softbots als Software den bekannten Problemen bei der Software-Entwicklung. Zudem sind gerade die gewünschten anspruchsvollen Fähigkeiten Ursache vieler weiterer Probleme.

Die unten angegebenen Vor- und Nachteile sind unabhängig davon anzusehen, ob heutige Agenten oben angeführte Fähigkeiten schon in einem solchen Maß besitzen, daß diese Vor- und Nachteile heute schon Realität oder noch Fiktion sind.

Vorteile und Chancen

Agenten können helfen, in der immer größer werdenden Datenflut Informationen – insbesondere die jeweils relevanten Informationen – zu finden. Heutige Werkzeuge haben nicht mit den technischen Möglichkeiten Informationen zu erzeugen, zu speichern und zu übertragen schrittgehalten; eine mögliche Lösung bieten Agenten ([14]). Diese werden nicht durch interessant scheinende Links bei der Suche abgelenkt ([12]). Außerdem wird durch *Personalisierung* und Lernfähigkeit der Nutzen von Agenten gegenüber einfachen Filtern und ähnlichen herkömmlichen Programmen wesentlich erhöht.

Ist die Fähigkeit *Autonomie* gegeben, so behandeln Agenten Mehrdeutigkeiten, unvollständige Informationen und auch Fehler ([15]). Bei letzterem kann es sich um fehlerhafte Anfragen handeln aber auch um Fehler und Ausnahmesituationen, auf die der Agent bei der Ausführung stößt. Die zur autonomen Problemlösung eingesetzte Technik des Planens ermöglicht es, nach einem Fehler einen anderen Weg zur Lösung einzuschlagen. Dies zeigt, wie sie für Flexibilität bei der Ausführung eines Auftrages sorgt ([6], [7]). Für die *Adaptivität* spielt das Lernen eine wichtige Rolle. Lernen ermöglicht es, die Wege zur Problemlösung an eine sich wandelnde Umgebung anzupassen oder Regelmäßigkeiten und Präferenzen eines Benutzers bei der Bedienung eines Programmes oder bei der Erfüllung bestimmter Aufgaben festzustellen ([6], [10], [16]). Paßt sich ein Agent an seinen Benutzer an und wird damit zu einem persönlichen Assistenten, so kann er ihn aufgrund der Personalisierung besser unterstützen als bisherige „unpersönliche“ Programme.

Ein Beispiel für die Möglichkeit, sich durch Lernen an eine Umgebung anzupassen und den Benutzer dadurch besser zu unterstützen, besteht darin zu lernen, mit neuen Diensten umzugehen ([10]). Hiermit wird auch eine weitere Chance deutlich: Agenten könnten die Benutzung immer komplexer werdender Systeme vereinfachen, indem sie die Komplexität verstecken ([13]). Der Benutzer könnte angeben, *was* er will, und der Agent kümmert sich darum, *wie* er den Auftrag erfüllt ([6]). Dabei kann der Agent verschiedene Dienste parallel nutzen; der Mensch muß sich nicht merken, wie jeder einzelne zu bedienen ist ([8]). Außerdem kann er dem Menschen zeitaufwendige Routinearbeiten abnehmen und ihm so mehr Zeit für wichtigere Arbeiten verschaffen ([12]).

Nachteile und Risiken

Agenten sollen autonom sein; sie sollen „eigene“ Ziele und Absichten verfolgen, selbständig Aktionen durchführen und dem Menschen Vorschläge machen. Sie sollen Aufgaben erledigen, die an sie delegiert wurden, und somit auf eine menschenähnliche Art Arbeiten verrichten, die andernfalls ein Mensch erledigen würde. Dies wirft außer technischen vor allem auch soziale Fragestellungen auf, zum Beispiel: „wie werden intelligente Agenten mit Menschen interagieren und vielleicht noch wichtiger, wie werden Menschen über Agenten denken?“ ([13]). Damit stellt sich sofort die Frage nach der Vertrauenswürdigkeit und Verläßlichkeit solcher Agenten ([3], [13]). Kann sich ein Benutzer darauf verlassen, daß sein Agent nicht eine äußerst wichtige E-Mail oder Datei löscht? Sehr deutlich wird die Notwendigkeit von Sicherheitsstufen zum Schutz dagegen, daß ein Agent seine Kompetenzen überschreitet, vor allem bei Fiktionen, in denen Agenten finanzielle Verfügungsgewalt haben und *im Namen des Benutzers* autonom und unter Umständen mobil auf Online-Märkten aktiv sind ([12]). Hier zeigen sich auch sehr deutlich rechtliche Probleme: „... sollte ein Benutzer für die Aktionen und Transaktionen seines Agenten verantwortlich gemacht werden?“ ([11]). Auch die schöne Eigenschaft moderner Planungsverfahren, daß ein erzeugter Plan sicher zum Ziel führt, hat seine Nachteile, da mehrere Pläne mit verschiedenen Seiteneffekten existieren können: Soll ein Unix-Softbot zum

Beispiel den Plattenplatzverbrauch auf unter 90% reduzieren, und erreicht er dieses Ziel, indem er wichtige Dateien unwiederbringlich löscht, ohne sie vorher zu sichern, so wäre es dem Benutzer sicherlich lieber gewesen, wenn sein Agent keinen so zielsicheren Plan gehabt hätte ([15]).

Donald A. Norman schreibt in [13] zu Problemen bei der Agenten-Entwicklung:

> *„Keiner dieser negativen Aspekte ist unvermeidbar. Alle können verhindert oder minimiert werden, aber nur wenn wir diese Aspekte im Entwurf unserer Agenten berücksichtigen.“*

6.2 Software-Umgebungen und Softbots

6.2.1 Software-Umgebungen und Interaktion

Eine Software-Umgebung kann ein Anwendungsprogramme, eine (verteilte) Datenbank, ein Betriebssystem oder ein Rechnernetz, insbesondere das Internet oder noch spezieller ein Teil davon – das World Wide Web (WWW) – sein ([15], [6], [8]). Natürlich sind auch beliebige Kombinationen dieser Möglichkeiten denkbar. Software-Umgebungen haben im allgemeinen die wesentlichen Eigenschaften, daß nie alle Informationen über sie bekannt sind (z.B. alle Dateien in einem großen Unix-System oder gar im Internet) und daß sie zudem hochdynamisch sind (z.B. ständiger Wechsel der aktiven Benutzer eines großen Unix-Systems, neue Rechner und Dienste im Internet oder Seiten im WWW). Außerdem handelt es sich um reale Umgebungen, so daß in ihnen immer auch unvorhergesehene Situationen auftreten (z.B Ausfall eines Gerätes oder Rechners, WWW-Seite nicht mehr vorhanden oder an einem anderen Ort).

Damit Softbots in Software-Umgebungen etwas bewirken können, müssen sie mit der Umgebung interagieren. Hierzu hat der Softbot wie schon erwähnt seine Effektoren und Sensoren. Dies soll nun konkretisiert werden. Im Falle eines Unix-Softbots ([6],[7]) können Effektoren Shell-Befehle, Scripts und Programme sein, die etwas verändern (z.B. mv, rm, gzip usw.); für Sensoren gilt gleiches, nur daß sie lediglich Informationen liefern (z.B. in Form von Ausgaben auf die Shell; Kommandos: z.B. ls, pwd, finger usw.). Im Falle eines Anwendungsprogramms können die Effektoren aus den Befehlen der Benutzerschnittstelle bestehen (der Agent emuliert sozusagen den Benutzer). Der Sensor nimmt die Reaktionen des Programmes auf. Dies ist noch vorstellbar, solange es sich um Programme handelt, die über Tastendrücke zu bedienen sind und als Ergebnis mit textuellen Ausgaben auf dem Bildschirm reagieren. Bei graphisch orientierten Programmen werden Probleme beim Wahrnehmen der Umgebung deutlich, da der Agent an die Ereignisse, die in der Anwendung auftreten, gelangen müßte, um zu agieren oder zu reagieren. Unter Umständen besteht die Möglichkeit, den Softbot an einer Stelle zwischenzuschalten, wo er die Ereignisse zunächst abfangen und auswerten kann, um sie anschließend zur Anwendung weiterzuleiten (z.B. zwischen X-Server und X-Anwendung).

In vielen Fällen muß also eine Unterstützung für den Agenten von der Anwendung vorgesehen sein. Auch im Fall des Unix-Softbots sind manche Aufgaben ohne spezielle Unterstützung von Seiten des Betriebssystems nicht effizient oder gar nicht durchführbar. So ist zum Beispiel das Feststellen oder Überwachen bestimmter Vorgänge im Betriebssystem (z.B. Einloggen eines Benutzers, Ende eines Druckjobs usw.) über die Shell nur über ineffizientes Polling möglich, das heißt der Agent ruft regelmäßig Befehle (z.B. who oder finger bzw. lpq) auf, um sich ein Bild von der aktuellen Situation zu verschaffen. Unter Umständen ist ein Vorgang auch auf diese Weise nicht zu überwachen, weil dies vielleicht nur mit privilegiertem Zugriff auf das Betriebssytem möglich wäre. Sind spezielle Unterstützungen in das Betriebsystem, genauer in den Kernel integriert, so sind Dienste über einen Agenten-Server denkbar, der privilegierten Zugriff auf das Betriebssystem hat ([7]).

Um Softbots durch die Software-Umgebung zu unterstützen, sind die Programme (ggf. der Kernel) zu modifizieren ([7]) oder gar neu zu implementieren ([11]) um dabei die erforderlichen Schnittstellen für den Softbot vorzusehen. Ein Vorteil hiervon – der direkte Zugriff auf die Datenstrukturen des jeweiligen Programmes – kann auch zugleich wieder nachteilig sein, weil die internen Zusammenhänge komplexer werden, da das Programm nicht mehr alleine für seine Datenstrukturen verantwortlich ist. Hauptnachteil ist aber, daß dies nur für Programme möglich ist, deren Quellen zur Verfügung stehen. Hierzu kommt außerdem die notwendige Arbeit zusätzlich zur Implementierung des eigentlichen Agenten. Selbst wenn für ein Programm die Quellen zur Verfügung stehen und die Änderungen mit vertretbarem Aufwand durchführbar sind, führt dies zu Problemen, wenn eine neue Version des Programmes erscheint. Zum einen müssen die Modifikationen erneut erfolgen, zum anderen kann es aufgrund von geänderten Datenstrukturen oder sonstigen Änderungen in der neuen Version vorkommen, daß die bisherige Implementierung der Agenten-Unterstützung nicht mehr paßt und geändert werden muß, was nochmals einen Mehraufwand bedeutet.

Dies legt nahe, daß sich die Entwicklung solcher Softbots erst dann wirklich lohnt, wenn eine Art genormter Schnittstelle für Agenten zur Verfügung steht oder daß nur der Hersteller eines kommerziellen Programmes selbst die Möglichkeit hat, solche Agenten anzubieten, bzw. der Agent als Funktionalität des Programmes mitgeliefert wird.

6.2.2 Einsatz von KI-Techniken

Aufgrund der Eigenschaften von Software-Umgebungen und der gewünschten Fähigkeiten der Softbots ist der Einsatz von KI-Techniken notwendig. Neben „intelligentem“ Pattern-Matching und fallbasiertem Schließen sind dies vor allem:

- **Planen**
  Um die Ausführung einer Aufgabe flexibel zu gestaleten, müssen Aktionsfolgen dynamisch zusammengesetzt werden. Dazu muß aufgrund eines bestimmten Zieles aus einer gegebenen Situation heraus geplant werden, welche Befehle in welcher Reihenfolge zum gewünschten Ergebnis führen. Weiterhin muß bei auftretenden Ausnahemesituationen zur Fehlerbehebung umgeplant werden. Hierzu kommt das Problem, daß der Softbot keine vollständige und unter Umständen falsche Informationen über die Umgebung haben kann, wodurch die in der KI oft benutzte *Closed World Assumption* hinfällig wird. Fehlende Informationen müssen also zwischendurch – mit den Sensoren – beschafft werden, was durch Verzahnung von Planung und Ausführung geschieht. Hierdurch gehen unter Umständen die schönen Eigenschaften moderner Planungsverfahren – *Korrektheit* und *Vollständigkeit* – verloren ([5], [4]).

- **Lernen**
  Um Adaptivität – an einen Benutzer oder an eine sich wandelnde Umgebung – zu gewährleisten, muß ein Softbot lernen. Im Falle eines Interface-Agenten, der sich an die Arbeitsweise eines Benutzers anpaßt, kann dies durch „beobachten“ der Aktionen des Menschen, durch positives und negatives Feedback des Benutzers auf Vorschläge und autonom vorgenommene Aktionen und durch explizites Trainieren anhand von Beispielen erfolgen ([11], [16]). Eine weitere Möglichkeit, neue Situationen zu lernen, besteht darin, andere Agenten zu fragen und die Lösung zu nehmen, die von der Mehrheit der Agenten vorgeschlagen wird. Außerdem könnte ein Agent dabei mit der Zeit lernen, welche Ratschläge von welchem Agenten am besten für den eigenen Benutzer geeignet sind und kann deren Ratschläge dann bevorzugen oder nur noch Ratschläge von diesen Agenten erfragen. Ein Beispiel für das automatische Lernen bestimmter Dienste im Internet bzw. im WWW ist in [10] bzw. [17] dargestellt. Diese Softbots interagieren mit den Diensten und lernen, wie diese zu bedienen sind, indem sie Anfragen mit bekannten Objekten stellen und die Ergebnisse interpretieren.

6.2.3 Suche im Internet mit lernenden Systemen

Verschiedene Suchprobleme

Bei „Suche im Internet“ fällt einem zunächst die Suche nach Informationen ein, vor allem im rasant wachsenden WWW. Aber es gibt noch andere Suchprobleme im Internet, bei denen Softbots die Menschen unterstützen können. Zum Beispiel die Suche nach

- einer Person, eventuell einem Experten für ein bestimmtes Problem,
- der Homepage einer Person im WWW,
- interessanten Büchern, Filmen oder bevorzugter Musik,
- einem günstigen Angebot,
- Antworten auf häufig gestellte Fragen (FAQ) oder
- der Bedeutung eines Wortes.

Außerdem könnte die Suche nach Software oder spezieller Hardware von Interesse sein, und sicher fällt dem Leser sofort noch ein weiteres Suchproblem ein, bei dem er gerne kompetente Unterstützung hätte.

Lösungsansätze mit Software-Agenten

- **Suche nach Informationen**
  Zunächst wäre sicher viel gewonnen, wenn man einen persönlichen Assistenten für die Suche hätte, also einen personalisierten Softbot, der auf die Interessen des jeweiligen Benutzers abgestimmt ist und entsprechend effizient für ihn suchen kann. Bei den heute verbreiteten Suchmaschinen hat man aufgrund der Einschränkung auf einen – nach irgendwelchen Kriterien zusammengestellten – Index, in dem man mit Schlüsselworten und meist nur einfachen logischen Verknüpfungen suchen kann, zu viele Fehlschläge.

  Bei dem rasanten Wachstum von Internet und WWW wird auch die Anzahl der Fehlschläge rasch zunehmen. Solche Indizier-Agenten werden nie das gesamte Web indizieren können; dies ist aufgrund des Umfangs und der hohen Dynamik gar nicht möglich. Außerdem werden Informationen in Datenbanken, auf Servern und von Informationsdiensten nicht in den Index aufgenommen, da sie – obwohl vom WWW aus zugreifbar – keine Web-Dokumente sind ([14]).

Dennoch werden diese Agenten gebraucht, da sie eine gute – wenn auch nicht vollständige – Basis für eine Suche nach Informationen bieten. In [8] ist ein Such-Softbot – der *MetaCrawler* – beschrieben, der zwar noch keine KI-Techniken einsetzt, aber immerhin erweiterte Suchmöglichkeiten bietet und außerdem von anderen Suchmaschinen profitiert, indem er neun populäre Maschinen für seine Suche benutzt und deren Ergebnisse sammelt. Dabei lädt er von Suchmaschinen, die manche Suchmöglichkeiten nicht bieten, gefundene Seiten zu sich herunter und führt die Suche selbst durch.

- **Suche nach einer Person, eventuell einem Experten für ein bestimmtes Problem**
  In [2] wird ein Softbot (*Directory Access Service*) vorgestellt, der zwar in seinen Fähigkeiten sehr eingeschränkt ist, aber bei der Suche nach einer Person im Internet hilft, indem er parallel mehrere Dienste benutzt und deren Ergebnisse in einer einheitlichen Form präsentiert.

  Eine speziellere Form der Suche nach einer Person ist das „Ausfindigmachen von Expertenwissen" (engl. *expertise location*). Das heißt, es wird im Internet nicht direkt nach den Informationen gesucht, sondern nach einem Experten, der das benötigte Wissen besitzt. Dies ist von großer Bedeutung, da niemals alle möglichen Informationen im Internet liegen können, vor allem nicht das gesamte Erfahrungswissen menschlicher Experten. Ein Beispiel, wie dieses Suchproblem unterstützt werden kann, ist die *EMMS-Datenbank* des *Zentrums für Mensch-Maschine-Systeme (ZMMS)* bei der TU Berlin, in der Experten im Bereich Mensch-Maschine-Systeme (Mensch-Computer-Schnittstellen) eingetragen sind. Hierbei handelt es sich zwar nicht um einen Softbot, es zeigt aber, daß zu diesem wichtigen Suchproblem Lösungen existieren und sich Softbots in Zukunft darum werden kümmern müssen.

- **Suche nach der Homepage einer Person im WWW**
  In [8] ist der *Ahoy!*-Softbot beschrieben, der den Benutzer bei der Suche nach der Homepage einer Person im WWW wirkungsvoll unterstützt. Er benutzt den MetaCrawler und filtert dessen Ergebnisse mit Heuristiken und aufgrund seiner Kenntnisse über die Web-Geographie. Weiter verwendet er Lernverfahren, um Konventionen bei den Adressen von Homepages zu finden, die ihm bei zukünftigen Suchen helfen und mit denen er sogar in der Lage ist, eine Homepage zu finden, noch bevor sie indiziert ist.

- **Suche nach interessanten Büchern, Filmen oder bevorzugter Musik**
  Ein beim MIT entwickelter Agent bzw. ein System von Agenten, das

mittlerweile im Rahmen eines von den Entwicklern gegründeten Unternehmens im WWW zur Verfügung gestellt wird, ist *firefly*. Für jeden Benutzer wird ein Agent generiert, der ihm nach einer „Gewöhnungsphase“ Tips zu Filmen oder Musik gibt, die wahrscheinlich der eigenen Interessenlage entsprechen ([14]). Dies geschieht auf Basis der eigenen Bewertung von gesehenen Filmen bzw. von Musik, die einem gefällt, und aufgrund der Beurteilungen anderer Benutzer.

  Solche Agenten sind auch für die Suche nach Büchern oder allgemein nach Unterhaltung im weitesten Sinn denkbar.

- **Suche nach einem günstigen Angebot**
  Bei der zunehmenden Kommerzialisierung des Internet wird es auch immer interessanter, auf Online-Shopping Tour zu gehen. Dabei bietet das Internet die schöne Möglichkeit, sehr viele Händler abzufragen und sich den günstigsten herauszusuchen. Dies von Hand zu machen, wäre zeitaufwendig, ziemlich unproduktiv und mit der Zeit auch nervend. Diese Suche nach einem günstigen oder gar dem besten Angebot läßt sich aber durch einen Softbot, wie den in [17] beschriebenen Comparison-Shopping Agent *ShopBot*, beschleunigen und erleichtern. Der Softbot kann mehr Händler besuchen als dies einem Menschen möglich wäre, er hat bessere Möglichkeiten, viele Angebote zu verarbeiten – insbesondere zu vergleichen – und sie so darzustellen, daß der Benutzer sich leichter einen Überblick verschaffen kann.

- **Suche nach Antworten auf FAQs**
  Ein Softbot für die sehr spezielle Suche nach Antworten auf häufig gestellte Fragen – der *FAQ-Finder* – ist in [14] genannt. Er nimmt eine Frage natürlichsprachlich entgegen, sucht dann eine passende FAQ-Datei und darin eine Frage, die der gestellten Frage sehr ähnlich scheint, und zeigt das Ergebnis dem Benutzer an. Dabei benutzt er auch einen Service, der Informationen zu Worten liefert – das *WordNet* ([18]). Dieses eignet sich auch für die **Suche nach der Bedeutung eines Wortes**.

  Da der FAQ-Finder FAQ-Dateien indiziert hat, von denen es sehr viel weniger gibt als WWW-Dokumente, und da diese Dateien außerdem noch teilweise strukturiert vorliegen, kann er sehr viel zuverlässiger sein als allgemeine Indizier-Agenten.

Softbots heute haben noch nicht viele der in 6.1.2 genannten anspruchsvollen Eigenschaften und Fähigkeiten. Es sind aber erste Ansätze zu einer besseren Unterstützung der Menschen in Internet und WWW; sie liefern ihren Beitrag dazu, die Gefahr des „lost in cyberspace“ zu veringern – und wer will schon auf einer „Datenautobahn“ überfahren werden ;-) ?

Literaturverzeichnis

[1] D. Weld, O. Etzioni. The First Law of Robotics (a call to arms). In *Proceedings 12th Nat. Conf. on A.I.*, 1994. Verfügbar via anonymous FTP von cs.washington.edu in /pub/etzioni/softbots.

[2] R.E. Droms. Access to Heterogeneous Directory Services. In *IEEE INFOCOM '90*, pages 1054–1061, San Francisco, 1990. IEEE.

[3] H.A.Kautz et al. A Bottom-Up Design of Software Agents. In *Communications of the ACM*, volume 37(7), pages 143–146. Juli 1994.

[4] K. Golden et al. Omnipotence Without Omniscience: Efficient Sensor Management for Planning. In *Proceedings 12th Nat. Conf. on A.I.*, 1994. Verfügbar via anonymous FTP von cs.washington.edu in /pub/ai.

[5] O. Etzioni et al. An Approach to Planning with Incomplete Information. In *Proceedings of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 1992. Verfügbar via anonymous FTP von cs.washington.edu in /pub/ai.

[6] O. Etzioni et al. Building Softbots for UNIX (Preliminary Report). Technical report, University of Washington, November 1992. Verfügbar via anonymous FTP von cs.washington.edu in /pub/etzioni/softbots.

[7] O. Etzioni et al. OS Agents: Using AI Techniques in the Operating System Environment. Technical report, University of Washington, August 1994.

[8] O. Etzioni. Moving Up the Information Food Chain: Deploying Softbots on the World Wide Web. Technical report, University of Washington, November 1992. Verfügbar im WWW unter http://www.cs.washington.edu/homes/etzioni.

[9] H. Engesser (Hrsg.). *Duden Informatik*. Dudenverlag, 2nd edition, 1993.

[10] M. Perkowitz, O. Etzioni. Category Translation: Learning to understand Information in the Internet. In *Proceedings 15th IJCAI*, 1995. Verfügbar im WWW unter http://www.cs.washington.edu/homes/etzioni.

[11] P. Maes. Agents that Reduce Work and Information Overload. In *Communications of the ACM*, volume 37(7), pages 31–40. Juli 1994.

[12] M.Wooldridge N. Jennings. Software agents. In *IEE Review*, pages 17–20, Januar 1996.

[13] D.A. Norman. How Might People Interact with Agents. In *Communications of the ACM*, volume 37(7), pages 68–71. Juli 1994.

[14] O. Etzioni, D. Weld. Intelligent Agents on the Internet: Fact, Fiction, and Forecast. Verfügbar im WWW unter http://www.cs.washington.edu/homes/etzioni.

[15] O. Etzioni, D. Weld. A Softbot-Based Interface to the Internet. In *Communications of the ACM*, volume 37(7), pages 72–79. Juli 1994.

[16] P. Maes, R. Kozierok. Learning Interface Agents. In *Proceedings of AAAI-93*, pages 459–465, 1993.

[17] R.B. Doorenbos et al. A Scalable Comparison-Shopping Agent for the World-Wide Web. Technical report, University of Washington, 1996. Verfügbar im WWW unter http://www.cs.washington.edu/homes/etzioni.

[18] R.D. Burke, K.J. Hammond, E. Cooper. Knowledge-based information retrieval from semi-structured text. Verfügbar via anonymous FTP von cs.uchicago.edu in /pub/users/burke.

[19] D. Riecken. Intelligent Agents – Introduction. In *Communications of the ACM*, volume 37(7), pages 18–21. Juli 1994.

Kapitel 7

Suchmaschinen im WWW

Georg Oehl

Übersicht

Within the past few years the World Wide Web has experienced a fast growth [1] in which more and more companies, institutions as well as private individuals strive to provide information of all kinds that others may find useful.

It is obvious that with this vast amount of data it is impossible to find specific information without some means of organizing that data. It is also obvious that without the help of some automated procedure the task of organizing would be a tedious and long process that could probably not keep up with the speedy changes of the information to be organized.

So-called *Search Engines* have been designed to tackle this task.

Search Engines are computer programs that scan pages on the *World Wide Web* and extract and index the information found according to the programmer's criteria.

This ordered and indexed information is in turn made publicly available on the *World Wide Web* to allow everyone to search the information database by supplying one or more search keywords.

This paper tries to provide an overview of various *Search Engines*. Different techniques of automated scanning of the *World Wide Web*, constraints while scanning as well as derivatives of the classic and some exotic *Search Engines* will be presented.

7.1 Einleitung

Die Suchmaschinen im World Wide Web stellen ein mittlerweile nicht mehr wegzudenkendes Werkzeug im Internet dar, das es erlaubt, Informationen zu praktisch jedem beliebigen Interessengebiet zu finden.

In dieser Ausarbeitung soll eine kurze Einführung in die Funktions- und Arbeitsweise, die Vor- und Nachteile verschiedener Suchmaschinen dargelegt werden.

7.1.1 Klassische Suchmaschinen

Klassische Suchmaschinen haben in der Regel drei große Bestandteile:

- den eigentlichen Suchmechanismus *(robot)*, mit dem WWW-Seiten durchsucht werden,
- die Datenbank *Resource Discovery Database* zum Ablegen der Suchergebnisse sowie
- das Benutzerinterface zum Wiederfinden gewünschter WWW-Seiten.

Robots

Robots sind Computerprogramme, die selbständig HTML-Dokumente des *World Wide Web* – meist in rekursiver Manier – „durchwandern“, Informationen aus diesen herausfiltern und ihre Ergebnisse in die ihnen angeschlossene Datenbank zur Katalogisierung stellen.

Ein detaillierterer Einblick in die Vorgehensweise von *robots* wird im Abschnitt 7.1.1 auf Seite 110 gegeben.

Organisation der Datenbank

Die Organisation der Datenbank, die den *robots* zur Datenaufnahme dient, würde den Umfang dieser Ausarbeitung übersteigen und wird deshalb hier nicht behandelt.

Wiederfinden und Präsentation der katalogisierten Daten

In den gesammelten katalogisierten Informationen der Datenbank soll nun möglichst effektiv die gewünschte Referenz gefunden werden.

Suchmaschinen bieten eine Fülle von Optionen zur Eingabe von Suchworten an, die sich prinzipiell gleichen, aber syntaktisch oft völlig verschieden sind. Es ist deshalb mühsam, sich vor Eingabe eines komplizierteren Suchausdrucks damit zu befassen, wie dies der Suchmaschine richtig mitgeteilt wird.

Hat der Benuzter sich jedoch mit der Syntax einer Suchmaschine einmal vertraut gemacht, so kann dies die Suchergebnisse oft erheblich verbessern.

Bei AltaVista [10] gibt es z.B. die Möglichkeit im Rahmen einer *Advanced Query* mit dem Schlüsselwort „NEAR“ als Verküpfungsoperator zweier Suchworte, solchen Suchergebnissen höheres Gewicht zu geben, in denen zwei vorgegebene Suchworte nicht weiter als zehn Worte voneinander entfernt sind.

7.1.2 Andere Konzepte für Suchmaschinen

Neben den klassischen Suchmaschinen, die automatisch das Internet katalogisieren, ihre Ergebnisse in eine Datenbank stellen und diese über ein Benutzerinterface zur Verfügung stellen, gibt es noch andere Ansätze.

ALIWEB

ALIWEB [4] basiert auf dem Prinzip, daß zu katalogisierende Informationen vom Informationsanbieter selbst und vor Ort in einem Index aufbereitet werden und dieser Index einem zentralen Anlaufpunkt für Suchanfragen zugestellt wird.

Dazu stellt der Informationsanbieter für jedes zu referenzierende Dokument dem Informationssammler eine Kurzbeschreibung zur Verfügung, etwa wie folgt:

```
Template-Type: SERVICE
Title:         The ArchiePlex Archie Gateway
URL:           /public/archie/archieplex/archieplex.html
Description:   A Full Hypertext interface to Archie.
Keywords:      Archie, Anonymous FTP.

Template-Type: DOCUMENT
Title:         The Perl Page
URL:           /public/perl/perl.html
Description:   Information on the Perl Programming Language.
               Includes hypertext versions of the Perl 5 Manual
               and the latest FAQ.
```

```
Keywords:      perl, programming language, perl-faq
```

Wie zu erkennen ist, ermöglicht diese Methode – im Gegensatz zu den selbständigen Suchmaschinen – eine Kategorisierung eines jeden Dokumentes ohne großen Aufwand. Dies wiederum verbessert die Qualität der späteren Suchergebnisse ganz erheblich.

Darüberhinaus ist die Netzlast dieses Ansatzes äußerst gering, da lediglich die Sammlung aller Dokumentkurzbeschreibungen vom Informationsanbieter zum Informationssammler zu übertragen ist. Es müssen nicht sämtliche Dokumente von einem dritten einzeln „besucht“ werden.

Als Nachteile wären zu nennen, daß im ALIWEB-Modell die Mitarbeit von Server-Betreibern notwendig ist, was organisatorisch bedingt vorerst zu kleineren Dokumentkatalogen führt als dies bei den selbständigen Suchmaschinen der Fall ist.

Auf lange Sicht gesehen werden ALIWEB und ähnliche Konzepte jedoch an Bedeutung gewinnen, da immer mehr Informationen ins Internet gestellt werden, die ohne vernünftige Katalogisierung und ohne eine die Netzlast schonende Suche nicht mehr handhabbar bzw. wiederfindabr sein werden.

Fish-Search

Fish-Search [5] ist eine Kleinst-Suchmaschine, die ohne Datenbank auskommt und stattdessen Dokumente erst nach Aufforderung durch den Benutzer durchsucht.

Es wird ein Startpunkt in Form einer URL in ein Formular eingegeben sowie der Suchstring. Sodann beginnt Fish-Search mit der sofortigen Suche des Strings vom angegebenen Startpunkt aus und geht dabei rekursiv den Hyperlinks nach bis zu einer vordefinierten Tiefe. Die Suche stoppt, wenn die Rekursionstiefe erreicht ist oder eine vorgegebene Zeit abgelaufen ist.

Fish-Search ist ideal beim Durchsuchen einer verzweigten Dokumenthierarchie, von der der Benutzer sich sicher ist, bezüglich seines gesuchten Themas fündig zu werden. Hält man die Rekursionstiefe und die Abbruchzeit der Suche allerdings nicht in Maßen, so ist Fish-Search in der Lage WWW-Server in die Knie zu zwingen. Dieses automatisch durchgeführte Durchsuchen einer Hierarchie durch unmittelbar aufeinanderfolgendes Laden von Dokumenten eines Servers nennt man auch „Rapid-Fire“ und ist allgemein unerwünscht.

Fish-Search sollte daher nur sehr sparsam und im kleinen Maßstab benutzt werden.

WebWatcher

WebWatcher [6] – ein an der Carnegie Mellon Universität laufendes Forschungsprojekt – stellt einen völlig anderen Ansatz einer Suchmaschine dar, indem es versucht durch Benutzerentscheidungen zu erlernen, welche Dokumente für diesen von Interesse sein können.

In der Praxis sieht eine WebWatcher-Sitzung so aus, daß der Benutzer in Form von Suchworten bekannt gibt, woran er interessiert ist. WebWatcher antwortet daraufhin mit einer WWW-Seite und schlägt Hyperlinks zu anderen Seiten vor, die für den Benutzer interessant sein könnten. Der Benutzer entscheidet nun, ob er den Vorschlägen folgt oder andere Hyperlinks auswählt.

Am Ende einer Sitzung wird WebWatcher mitgeteilt, ob ein gewünschtes Dokument gefunden wurde oder die Suche erfolglos war. Aus diesen Daten versucht WebWatcher nun, ein Muster zu erkennen und dem Benutzer bei der nächsten Suche bessere Vorschläge machen zu können.

Leider ist dieser Ansatz in der Praxis wenig hilfreich. Die Prozedur ist langwierig und erfordert vom Benutzer eine ständige Interaktion. Genügend Zeit ist aber oft nicht vorhanden. Der Benutzer verliert schnell das Interesse und wendet sich ab. Darüberhinaus müßte für jede neue Suchanfrage die Prozedur erneut gestartet werden.

7.2 Robots: Durchsuchen des Internet

Ein *robot* [7] ist ein Programm, das (meist) den HTML-Raum des *World Wide Web* durchläuft, indem es HTML-Dokumente lädt und rekursiv die dort referenzierten Dokumente besucht.

Oft werden *robots* auch als *Web Wanderers*, *Web Crawlers* oder *Spiders* bezeichnet.

Die Funktions- und Vorgehensweise von *robots* ist sehr unterschiedlich.

Einige *robots* durchsuchen und indizieren ausschliesslich das *World Wide Web*, während andere zusätzlich Artikel des *Usenet* katalogisieren oder sogar in eMail-Adreßdatenbanken suchen. Auch gibt es *robots*, denen manuell eine URL als Startpunkt zur Suche vorgegeben werden kann.

Der nächste Schritt, der wiederum auf unterschiedliche Weise gelöst wird, betrifft die Frage, welche Daten indiziert werden sollen. Ein *robot* mag ein Dokument vollständig in seine Datenbank stellen, indem er jedes Wort katalogisiert und wichtet, um später entscheiden zu können, wie wichtig dieses

Dokument bzgl. eines bestimmten Suchwortes ist. Dies erleichtert das Wiederfinden von Dokumenten ganz erheblich, da ein Dokument, in dem ein Suchwort vielleicht nur ein- oder zweimal vorkommt, sicherlich weniger von Interesse ist, als eines, in dem dasselbe Suchwort zwanzigmal erscheint.

Andere *robots* katalogisieren vielleicht nur den Titel oder den ersten Absatz eines Dokumentes, wodurch die anfallende Datenmenge natürlich reduziert wird, aber das spätere Wiederfinden der passenden Dokumente sich unter Umständen schwieriger gestaltet.

Ein weiterer wichtiger Punkt beim Durchlaufen des *World Wide Webs* durch einen *robot* ist die Rücksichtnahme auf andere Teilnehmer. Das bedeutet, daß z.B. bestimmte WWW-Server schnell überlastet werden können, falls ein *robot* im schnellstmöglichen Tempo sämtliche Dokumente dieses Servers anfordern sollte.[8]

Oft ist es aber auch gar nicht sinnvoll, bestimmte Dokumente zu katalogisieren, was bei sich häufig ändernden Daten wie z.B. Nachrichten der Fall ist. Ebenfalls macht es wenig Sinn, URLs zu indizieren, die selbst Suchanfragen an Suchmaschinen sind.

Da aber kein *robot* feststellen kann, ob auf einem besuchten Server eine Suche denn nun sinnvoll oder gar vom Betreiber erwünscht ist, gibt es einige Möglichkeiten *robots* dies mitzuteilen.

Zum einen sollten *robots* so programmiert werden, daß sie selbst ihre Suchaktivitäten auf ein vertretbares Minimum reduzieren, um die Netz- und Serverlast so gering wie möglich zu halten – indem sie z.B. maximal ein Dokument innnerhalb von fünf Minuten von demselben WWW-Server anfordern –, zum anderen sollten sie den *Standard for Robot Exclusion* [9] befolgen.

Der *Standard for Robot Exclusion* beschreibt eine unverbindliche – aber allgemein anerkannte – Regelung, die den Betreibern von WWW-Servern ein Werkzeug zur Steuerung der sie besuchenden *robots* in die Hand gibt, mit dem sie in der Lage sind, *robots* vollständig oder nur teilweise von ihren Servern zu verbannen.

Über die Datei `robots.txt` im Hauptverzeichnis eines Servers können an *robots* Informationen übermittelt werden, die ihnen zu erkennen geben, welche Bereiche des besuchten Servers für sie tabu sind und welche nicht.

Bevor ein *robot* also z.B. einen WWW-Server `www.zzz.com` besucht, sollte er die Datei `http://www.zzz.com/robots.txt` versuchen zu laden. Existiert diese Datei nicht, so gelten für *robots* keine Einschrnkungen an diesem Server.

Nachfolgend ein Beispiel für eine Datei `robots.txt`:

```
# /robots.txt file for http://webcrawler.com/
```

```
# mail webmaster@webcrawler.com for constructive criticism

User-agent: webcrawler
Disallow:

User-agent: lycra
Disallow: /

User-agent: *
Disallow: /tmp
Disallow: /logs
```

Die ersten beiden Zeilen – mit '#' beginnend – sind Kommentarzeilen.

Im ersten Abschnitt wird dem *robot* mit Namen `webcrawler` uneingeschränkt erlaubt, Dokumente dieses Servers zu laden.

Der *robot* `lycra` hingegen darf vom Hauptverzeichnis '/' aus kein Dokument besuchen, was ihn also vollständig von diesem Server ausschließt.

Der letzte Abschnitt betrifft alle anderen *robots* ('*') und teilt diesen mit, daß die Verzeichnisse '/tmp' und '/logs' sowie alle in der Hierarchie darunter liegenden Verzeichnisse für sie tabu sind.

7.3 Beispiele konkreter Suchmaschinen

Suchmaschinen präsentieren ihre Suchergebnisse in einem einander ähnelnden Format. In der Regel werden die Suchergebnisse auf eine bestimmte Anzahl pro Seite begrenzt. Oft erhalten die einzelnen Ergebnisse Wertungen, die aus verschiedenen Faktoren berechent werden. Ein Dokument, das ein haüfigeres Vorkommen eines Suchwortes aufweist als ein zweites Dokument, kann z.B. eine höhere Wertung erhalten.

7.3.1 AltaVista

Alta Vista [10] – eine von DEC betriebene, im Herbst 1995 ins Leben gerufene Suchmaschine – ist mit über 30 Millionen indizierter WWW-Seiten wahrscheinlich die zur Zeit größte und schnellste Suchmaschine im Internet.

Es kann sowohl im WWW als auch in Usenet Artikeln gesucht werden. Die gängigen Bool'schen Verknüpfungen zur Kombination von Suchwörtern sind erlaubt.

Die Suchergebnisse können – benutzerdefiniert – verschieden detailliert angezeigt werden. Entweder als eine Zeile pro Eintrag (Titel, Datum und die ersten 30 Zeichen der referenzierten Seite) oder als vierzeiliger Eintrag (mit Titel, die ersten zwei Zeilen, URL, Größe in KB sowie das Datum bei der Suche der referenzierten Seite).

Die Suchergebnisse werden – laut AltaVistas „Bedienunsanleitung“ [11] – nach Wichtigkeit sortiert (mit dem wichtigsten Eintrag zuerst) dem Benutzer angezeigt. Ein Dokument wird stärker gewichtet, wenn die Suchworte am Anfang des Dokuments oder sogar im Titel stehen, wenn die Suchworte nahe zueinander im Dokument vorkommen oder wenn ein oder mehrere Suchworte mehrmals im Dokument vorkommen. Bei der Anzeige der referenzierten Dokumente wird allerdings versäumt, diese Wichtung mit anzuzeigen, so daß es schwierig bis unmöglich ist zu beurteilen, bis zu welchem der Dokumente diese für den Benutzer noch relevant sind.

Spezielle Sucharten stehen zur Verfügung. So besteht z.B. die Möglichkeit, die Suche auf einen bestimmten Server zu beschränken oder nach Dokumenten zu suchen, die eine vorgegebene URL referenzieren.

Die Antwortzeit der AltaVista Suchmaschine ist je nach Netzauslastung relativ kurz. Allerdings lassen die Suchergebnisse oft zu wünschen übrig, da Mehrfachnennungen der selben URL nicht selten auftreten. Das macht die Liste der Suchantworten unnötig lang und unübersichlich. Dem Benutzer wird viel Geduld abverlangt, da er sich durch mehrere Ergebnisseiten arbeiten muß.

7.3.2 Yahoo!

Yahoo! [12] – auf einem maßgeschneiderten Datenbanksystem beruhend – wurde im April 1994 von zwei Doktoranden der Universität Stanford als Hobby begonnen, dessen Popularität schnell die Leistungsfähigkeit der universitären Computer überforderte. Anfang 1995 zog Yahoo! auf größere Computer im Hause Netscape um und entlastete dadurch Stanford.

Das auffälligste Merkmal des Benutzerinterfaces von Yahoo! – im Gegensatz zu AltaVista – ist die Verwendung von Kategorien. Der Benutzer ist nicht gezwungen, Suchworte einzugeben, sondern kann sich hierarchisch zu seinem Interessengebiet vorarbeiten und somit seine Suche verfeinern.

Die Eingabe von Suchworten ist natürlich auch möglich und kann sich entweder auf den gesamten Datenbestand von Yahoo! oder lediglich auf bestimmte Kategorien beziehen. Letzteres kann unter Umständen die Suche nach der gewünschten Information erheblich vereinfachen und Zeit sparen, da die Qualität der möglichen Antworten durch die Kategorisierung erheblich höher ist. Der Benutzer muß also nicht mühsam Antworten aussortieren – wie dies der Fall bei AltaVista ist –, die überhaupt nichts mit seiner Suchanfrage zu tun haben oder doppelt vorkommen.

Yahoo! erlaubt die Suche im WWW, in Usenet Artikeln und sogar nach eMail Adressen. Die Suchergebnisse werden nach Kategorien geordnet angezeigt und innerhalb der Kategorien alphabetisch sortiert. Boole'sche Verknüpfungen der Suchworte sind nur eingeschränkt möglich; lediglich reine OR- und AND-Verknüpfungen ohne Kombination sind zulässig.

Es stehen keine weiteren speziellen Sucharten zur Verfügung, wie z.B. bei AltaVista.

Die Antwortzeit von Yahoo! ist mäßig bis schnell, in der Regel jedoch länger als bei AltaVista. Die Kategorisierung von Yahoo! ist grundsätzlich keine schlechte Idee, jedoch ist unverständlich, warum die Suchergebnisse alphabetisch geordnet werden, da man in dieser Ordnung keine Wichtung der Suchergebnisse erkennen kann. Das Suchergebnis – nach Kategorien geordnet – ist auch oft unübersichtlich und ermüdend und erfordert eine gewisse Ausdauer des Benutzers.

Zusätzlich zur Suche nach Informationen bei Yahoo! ist es möglich, Yahoo! selbst eine Empfehlung für einen neuen Eintrag mitzuteilen. Dabei wird die Einsortierung eines Dokumentes vom Benutzer selbst vorgenommen. Allerdings steht das Dokument noch nicht sofort anderen Benutzern als mögliche Suchantwort zur Verfügung, da es erst manuell von Yahoo!-Mitarbeitern auf Richtigkeit und Relevanz überprüft werden muß. Diese Verfügbarmachung kann unter Umständen jedoch mehrere Wochen dauern.

7.3.3 Infoseek

Infoseek Corporation wurde im Januar 1994 gegründet und bietet – neben einem kommerziellen Suchdienst – mehrere andere Suchdienste an, wie z.B. Ausgabe von Kartenausschnitten anhand von US-Adressen oder Suche in Datenbanken von US-Telefonnummern.

Im Rahmen des *Infoseek Guide* [13] ist es möglich nach Informationen im WWW, in Usenet Artikeln, nach Firmen, nach eMail Adressen, in Nachrichten des vergangenen Monats sowie in FAQs der Newsgruppe news.answers zu suchen.

Ähnlich wie Yahoo! erlaubt auch Infoseek eine auf bestimmte Kategorien beschränkte Suche.

Suchergebnisse werden bewertet und gemäß ihrer Wichtung sortiert aufgelistet. Darüberhinaus gibt es zu jedem gefundenen Eintrag über die Option *Similar Pages* die Möglichkeit, eine Liste ähnlicher Dokumente angezeigt zu bekommen. Hierbei handelt es sich um Dokumente, die unter Umständen nur entfernt mit den gesuchten Informationen zu tun haben.

Zusätzlich wird zu jedem Suchergebnis eine Liste von Kategorien angezeigt, die ebenfalls mit dem Gesuchten thematisch zu tun haben.

Der *Infoseek Guide* stellt eine Art Mischung aus AltaVista und Yahoo! dar, da er kategorisiert wie Yahoo! und wichtet wie AltaVista.

7.3.4 MetaCrawler

MetaCrawler [14] ist eine Suchmaschine, die Suchanfragen lediglich an andere Suchmaschinen weitergibt, selbst aber keine Datenbank unterhält.

Eine Suchanfrage an MetaCrawler wird simultan an eine Auswahl von zur Zeit neun Suchmaschinen gesendet. Die verschiedenen Ergebnisse werden verglichen, auf Existenz überprüft und dem Benutzer nach verschiedenen Kriterien, die er selbst bestimmen kann, gefiltert und sortiert angezeigt.

Die WWW-Server der Suchergebnisse können sowohl logisch auf bestimmte Internet *domains* (`.com, .edu, ...`) als auch geographisch auf bestimmte Kontinente (Nordamerika, Europa, Asien,...) beschränkt werden. Es ist außerdem möglich, die Anzahl der Antworten an jede Suchmaschine und die Zeit der Suche zu begrenzen.

Vorteile des MetaCrawlers sind:

- Das Spektrum der Suchergebnisse ist größer.
- Dem Benutzer wird ein einheitliches Benutzerinterface zur Verfügung gestellt. Unterschiede zwischen den einzelnen Suchmaschinen in der Syntax zur Formulierung von Suchanfragen bleiben ihm verborgen.
- Der Benuzter kann im Falle komplizierterer Suchanfragen Zeit sparen, da er nicht mehrere Suchmaschinen besuchen muß.
- Nicht-existente Dokumente werden vor der Ergebnisausgabe gefiltert und dem Benutzer nicht angezeigt.
- Die Relevanz bestimmter Suchergebnisse kann anhand des Feedbacks von mehreren Suchmaschinen besser beurteilt werden.
- Kein speicher- und rechenintensiver Computer muß unterhalten werden, da auf die Ressourcen von Dritten zurückgegriffen wird.

In den meisten und simpelsten Fällen ist das Mehr an Rechenaufwand zur Filterung nicht-existenter Dokumentverweise sowie die simultane Anfrage bei mehreren Suchmaschinen unnötig. Hier reicht oft schon eine Anfrage bei einzelnen Suchmaschine aus, die zudem auch noch schneller beantwortet wird. Bei Anfragen jedoch, die nicht zufriedenstellend von einer bestimmten Suchmaschine beantwortet wird oder wenn Zeit keine Rolle spielt, bildet der MetaCrawler eine durchaus nützliche Alternative.

Oft ist aber nicht vorhersehbar, wie gut eine Suchanfrage von einer einzelnen Suchmaschine beantwortet wird; etwas Erfahrung des Benutzers im Umgang mit Suchmaschinen kann hier hilfreich sein.

7.4 Fazit

Die heutzutage angebotenen (klassischen) Suchmaschinen stellen ein durchaus nützliches und brauchbares Hilfsmittel bei der Informationssuche im Internet dar.

Bereits einfache Suchanfragen bringen den Benutzer in den meisten Fällen recht schnell zum Ziel. Redundanzen in der Ausgabe von Suchergebnissen – obwohl nie vollständig vermeidbar – lassen sich oftmals durch erweiterte Suchanfragen auf ein erträglich Maß reduzieren (z.B. Boole'sche Verknüpfung von Suchworten).

Lernende Suchmaschinen sind noch in der frühen Entwicklungsphase und für den praktischen Einsatz untauglich.

Alternative Ansätze, die die Netzlast auf ein Minimum reduzieren, sind zwar oft nicht so umfangreich in ihren Suchantworten wie klassische Suchmaschinen, werden längerfristig gesehen jedoch mehr und mehr zum Einsatz kommen müssen. Denn: der bisher unaufhaltsame und unverkennbare Wachstumsstrend des Internet wird den Einsatz der klassischen Suchmaschinen

Darüberhinaus existieren eine unübersehbare Vielzahl an spezialiserten Suchmaschinen, darunter sind z.B. solche, die

- nach Eingabe einer US-Adresse einen Kartenausschnitt der Umgebung zurückliefern, sowie den exakten Ort der Adresse mit einem Kreuz kennzeichnen,
- einen *Thesaurus* (Synonymwörterbuch der englischen Sprache) online anbieten,
- den momentanen Standort eines mit `UPS` oder `Federal Express` verschickten Paketes anzeigen.

So komfortabel die Suche nach Informationen im Internet auch sein mag, oft bekommt man auch einen gehörigen Schrecken, wenn man herausbekommt, wieviel man über einen Menschen oder einen selbst in Erfahrung bringen kann.

So bietet z.B. *DejaNews* (`http://www.dejanews.com`) einen Service an, mit dem sich unter Angabe einer eMail-Adresse sämtliche Postings der betreffenden Person im Usenet zurück bis Mitte 1995 anzeigen lassen. Vorausgesetzt, die Person hat genügend „Material" publiziert, läßt sich aus diesen Informationen ein recht gutes Interessenprofil ableiten, das zu weiteren Zwecken durchaus mißbraucht werden könnte.

7.5 Tabellarische Funktionsübersicht ausgewählter Suchmaschinen

Suchmaschine	Wichtng.	Sort.	Boole'sch	Kateg.	Antw.-Zeit
AltaVista	n	j	komfort.	n	gut
Excite	n	j	n	n	mittel
Infoseek	n	j	j	j	mittel
Lycos	j	j	minimal	n	mittel
MetaCrawler	j	j	minimal	n	mäßig
OpenText	n	j	minimal	n	mäßig
Yahoo!	n	n	minimal	j	mittel

Erläuterungen:

Wichtung – gibt an, ob eine Wichtung für jedes gefundene Dokument angezeigt wird.

Sortierung – gibt an, ob die Antworten sortiert angezeigt werden (können).

Boole'sch – gibt an, ob eine Boole'sche Verknüpfung von Suchworten möglich ist.

Kategorie – gibt an, ob Suchergebnisse in Kategorien sortiert angezeigt werden können.

Antwortzeit – gibt an, wie schnell in der Regel die Suchmaschine antwortet.

Literaturverzeichnis

[1] A.M. Rutkowski. *WWW-Prefixed Hosts.* (Grafik: Entwicklung der Anzahl WWW-Hosts zwischen Juni 1994 und Januar 1995.) http://www.genmagic.com/internet/trends/sld006.htm

[2] *About AltaVista.* Profil AltaVista. Digital Equipment Corporation. http://www.altavista.digital.com/cgi-bin/query?pg=about

[3] *Yahoo! FAQ.* Häufig gestellte Fragen zu Yahoo! http://www.yahoo.com/docs/info/faq.html

[4] Martijn Koster. *ALIWEB - Archie-like Indexing in the Web.* Proceedings of the First International World-Wide Web Conference, May 1994.

[5] Dr. P.M.E. De Bra, Drs. R.D.J. Post. *Information Retrieval in the World-Wide Web: Making Client-based searching feasible.* http://www.win.tue.nl/win/cs/is/reinpost/www94/www94.html

[6] Thorsten Joachims, Tom Mitchell, Dayne Freitag, Robert Armstrong. *Web Watcher: Machine Learning and Hypertext.* Fachgruppentreffen Maschinelles Lernen, Dortmund, 1995. http://www.cs.cmu.edu/afs/cs/project/theo-6/web-agent/www/project-home.html

[7] Martijn Koster. *WWW Robot Frequently Asked Questions*, Feb. 1996. Häufig gestellte Fragen zu *Robots* im WWW. http://info.webcrawler.com/mak/projects/robots/faq.html

[8] Thomas Boutell. *World Wide Web FAQ.* Häufig gestellte Fragen zum World Wide Web. (Frage: 'Hey, I know, I'll write a WWW-exploring robot!'.) http://info.ox.ac.uk/help/wwwfaq/robots.htm

[9] Martijn Koster. *A Standard for Robot Exclusion.* http://info.webcrawler.com/mak/projects/robots/norobots.html

[10] *AltaVista Main Page.* WWW-Startseite AltaVista. Digital Equipment Corporation. http://www.altavista.digital.com

[11] *Altavista Help for Simple Query.* Hinweise für Benutzer zur Formulierung von Suchanfragen. Digital Equipment Corporation. http://www.altavista.digital.com/cgi-bin/query?pg=h&what=web

[12] *Yahoo! Main Page.* WWW-Startseite Yahoo! http://www.yahoo.com

[13] *Guide Infoseek.* WWW-Startseite Infoseek. http://guide.infoseek.com

[14] *MetaCrawler.* WWW-Startseite MetaCrawler. http://metacrawler.cs.washington.edu:8080/index.html

Kapitel 8

Multi-Service Search

Markus Fiege

Übersicht

Ein großes Problem bei der Nutzung des World Wide Webs besteht im Auffinden bestimmter Ressourcen. Zwar wächst die Anzahl der unterschiedlichsten Suchdienste ständig, doch für ein normales Suchprogramm ist es unmöglich, Verweise über alle Ressourcen des World Wide Webs zu speichern und auf dem neuesten Stand zu halten. Deshalb unterstützt ein Suchservice nur die Suche auf einem Teilgebiet der Ressourcen. Das Suchen einer bestimmten Ressource erfordert also das wiederholte Stellen der Anfrage in verschiedenen Suchdiensten, bis sie von einem Suchprogramm gefunden wurde. Diese umständliche und zeitaufwendige Art der Suche wird von Multi-Service Suchagenten umgangen. Bei ihnen ist es nur noch nötig, die Anfrage ein einziges mal zu stellen. Der Suchdienst sendet die Anfrage parallel an andere herkömmliche Suchdienste weiter, so daß ein genügend großes Teilgebiet des World Wide Webs abgedeckt wird. Die zurückgelieferten Ergebnisse werden zusammengefaßt und geschickt präsentiert. Die genauen Vorteile von Multi-Service Suchagenten sowie sich ergebende Probleme und mögliche Problemlösungen sollen im Folgenden diskutiert werden, wobei die Betrachtungen am Beispiel der Realisierung der Multi-Service Suchdienste „MetaCrawler" und „SavvySearch" gestützt werden.

8.1 Einführung

8.1.1 Wozu Multi-Service Suchdienste ?

Dem Nutzer des World Wide Webs (WWW) steht eine riesige Anzahl von Ressourcen in Form von Texten, Bildern, Software, Animationen und vielen anderen Formen zur Verfügung. Diese Ressourcen liegen jedoch oft brach, da ihre Existenz dem Anwender oft nicht bewußt ist. Aber selbst das Wissen über die Existenz hilft ihm oft nicht weiter, denn ein großes Problem bei der Nutzung des Internets besteht im Auffinden bestimmter Ressourcen. Um dieses Problem zu beheben, gibt es eine ständig wachsende Anzahl der unterschiedlichsten Suchdienste im WWW.

Ein gewöhnlicher Suchdienst deckt immer nur einen Teil des Internets ab. Jeder Suchservice durchläuft mit Hilfsprogrammen die ihm vorgegebenen Bereiche von Servern und Dokumenten und wendet verschiedene Methoden des Auffindens von Informationen an. Einige Suchprogramme durchsuchen nur die Maschinennamen oder die Verzeichnisse und Dateinamen (die URLs), während andere Titel, Überschriften oder sogar ganze HTML-Seiten durchsuchen. Die gefundenen Ressourcen werden in der Datenbank des Suchdienstes gespeichert. Hierbei benutzt jeder Suchdienst seine eigenen Verfahren zur Speicherung sowie verschiedene Update-Perioden. Dem Suchdienstnutzer wird eine Schnittstelle zur Verfügung gestellt, über die er in einem Eingabefeld Informationen über die von ihm gesuchte Ressource eingeben kann. Das Suchprogramm überprüft die Anfrage mit Patternmatchingverfahren auf Übereinstimmungen mit Einträgen seiner Datenbank.

Kein Suchdienste ist in der Lage, das ganze World Wide Web abzudecken. Einerseits sind zu viele Ressourcen vorhanden, als daß alle erfaßt werden könnten, anderseits verändert sich das Netz ständig. Es werden dauernd neue Ressourcen ins WWW eingefügt, verändert oder gelöscht, so daß unmöglich jede Änderung sofort festgestellt werden kann. Deshalb hängt das Ergebnis einer Anfrage stark vom verwendeten Suchdienst ab. Um eine bestimmte Ressource zu finden, muß der Suchende nacheinander verschiedene Suchagenten durchlaufen, bis einer auf seine Anfrage einen Verweis auf die gewünschte Ressource liefert. Diese Art der Suche ist jedoch äußerst zeitaufwendig, denn es muß in verschiedenen Suchdienste die Anfrage gestellt, das Ergebnis abgewartet und die Ergebnisliste auf die gesuchte Ressource hin untersucht werden. Außerdem besitzt jedes Suchprogramm sein eigenes Interface, das der Benutzer erlernen muß.

Aus diesen Gründen sind Multi-Service Suchagenten entwickelt worden. Hier stellt der Anwender nur einmal seine Anfrage, die parallel an verschiedene herkömmliche Suchdienste weitergeschickt wird. Die zurückgelieferten Ergebnisse werden zusammengefaßt und dem Nutzer geschickt präsentiert.

8.1.2 Eigenschaften von Multi-Service Suchdiensten

Durch die Nutzung anderer Suchdienste kann eine eigene interne Datenbank eingespart werden. Dies bietet zwei Vorteile. Zum einen wird praktisch die Suche von Ressourcen im ganzen World Wide Webs unterstützt. Der Multi-Service Suchdienst kann in den meisten Fällen die gewünschten Informationen aus der Datenbank eines gewöhnlichen Suchdienstes entnehmen, denn einer der Suchdienste hat normalerweise immer die gesuchte Ressource erfaßt. Zum anderen kann der Speicherplatz für die Speicherung sowie die Rechenleistung für die Aktualisierung der Datenbank eingespart werden. Während herkömmliche Suchdienste ständig damit beschäftigt sind, neue Ressourcen zu finden und ihre Datenbank zu aktualisieren, kann sich ein Multi-Service Suchdienst ganz darauf konzentrieren, die Anfragen zu bearbeiten, diese zu den verschiedenen Datenbanken anderer Suchdienste weiterzuleiten, sowie die gefundenen Treffer geschickt zu präsentieren. Auf diese Weise kann ein kleines, kompaktes und effizientes Suchprogramm entstehen.

Ein Multi-Service Suchdienst besitzt als weiteren Vorteil nur ein Interface, mit dem die Suche in verschiedenen Suchdiensten durchgeführt wird. Der Benutzer muß hier lediglich den Umgang mit dem einen Interface erlernen, im Gegensatz zu der sequentiellen Suche mit verschieden herkömmlichen Suchprogrammen. Bei dieser Art der Suche muß der Nutzer die Syntax der Anfragesprache des jeweiligen Suchsystems erlernen, dazu muß er sich an die verschiedenen Menüs und Bedienelemente des Suchdienstes gewöhnen.

Multi-Service Suchdienste sind nicht von der Implementierung eines der benutzten Suchdienste abhängig. Zwar ist die Existenz von Suchprogrammen oder Indizes des World Wide Webs notwendig, aber die Art und Weise der Suche, die Suchverfahren und sonstige Details des Suchdienstes sind für den Multi-Service Suchagenten völlig unwichtig. Es sind nur Kenntnisse über die Datenschnittstelle nötig, in welcher Form der Suchdienst eine Anfrage erwartet oder in welcher Form das Suchergebnis zurückgeliefert wird.

Es ergeben sich natürlich auch Probleme bei solchen Multi-Service Suchagenten. Eine Anfrage eines Multi-Service Suchagentens mit einer einfachen Auflistung der Suchergebnisse mehrerer Suchdienste ergibt häufig ein Vielfaches der Verweise eines herkömmlichen Suchdienstes. Neben der größeren Anzahl der relevanten Treffer vervielfacht sich natürlich auch die Anzahl der irrelevanten Daten. Dem Nutzer sollten diese Daten nicht alle präsentiert werden, da er sonst mit Informationen förmlich überhäuft wird. Es muß eine geschickte Vorauswahl getroffen werden bevor das Suchergebnis dem Anwender präsentiert wird. Verfahren zur Vorauswahl der Daten werden in Kapitel 8.2.4 näher betrachtet.

Ein weiterer Nachteil besteht in der längeren Zeitdauer der Suche gegenüber einem einzigen herkömmlichen Suchdienst. Die Abarbeitung einer Anfrage durch einen Multi-Service Suchdienst besteht aus mehreren Schritten. Die Anfrage muß in eine für den untergeordneten Suchdienst verständliche Form gebracht und abgeschickt werden. Dann müssen die Ergebnisse sämtlicher Suchdienste abgewartet, zusammengefaßt, noch einmal überarbeitet und dem Suchenden präsentiert werden. Es kommt also zur Dauer des Suchens des langsamstens Suchdienstes noch die Zeit zur Vorverarbeitung der Anfrage sowie der Nachbearbeitung der Ergebnisse hinzu.

Selbstverständlich sind Multi-Service Suchdienste höchstens so gut wie die herkömmlichen Suchdienste, die sie benutzen. Es können dem Suchenden nur Verweise geliefert werden, die die untergeordneten Suchdienste bereits gefunden haben. Das Ergebnis einer Anfrage hängt zudem immer von der Qualität der Anfrage ab. Da die semantische Spracherkennung heutzutage noch nicht einsetzbar ist, kann in den Suchprogrammen auch nur auf Übereinstimmungen des Suchstrings mit Einträgen in der Datenbank geprüft werden. Die Weichen für eine erfolgreiche Suche müssen deshalb vom Suchenden gestellt werden. Ohne das Suchprogramm mit den nötigen Informationen zu versorgen, ist eine erfolgreiche Suche unwahrscheinlich. Eine schlecht gestellte Anfrage wird logischerweise auch selten ein gutes Suchergebnis liefern, egal ob mit Hilfe eines herkömmlichen Suchdienstes oder eines Multi-Service Suchagentens.

8.1.3 Messung der Qualität eines Suchergebnisses

Bei der Suche gibt es drei wichtige Größen zur Messung der Güte der gewonnenen Informationen. Diese sind:

- **Relevanz:** Sie drückt aus, wie nützlich ein Suchergebnis für den Suchenden ist. Das Ziel eines jeden Suchdienstes besteht im Auffinden möglichst relevanter Dokumente.
- **Präzision:** Sie spiegelt die Qualität einer Menge von Suchergebnissen wieder. Eine hohe Präzision bedeutet, daß die meisten der Suchergebnisse relevant sind. Um eine hohe Präzision zu erzielen ist es erforderlich, die zurückgelieferten Suchergebnisse auf die mit hoher Wahrscheinlichkeit als relevant anzunehmenden Dokumente zu beschränken. Ein präzises Suchergebnis enthält also nur relativ wenig Dokumente.
- **Recall:** Der Recall mißt, wie nahe das Suchergebnis an alle zu findenden relevanten Dokumente herankommt. Ein hoher Recall bedeutet, daß das Suchergebnis nahezu alle relevanten Dokumente enthält; d.h. auch die Dokumente, die nicht unbedingt relevant sind. Ein Suchergebnis mit einem hohem Recall enthält dementsprechend viele Dokumente.

Bei einer genaueren Betrachtung stellt man fest, daß die Größen Präzision und Recall gegensätzlich sind. Eine hohe Präzision impliziert einen geringen Recall, während ein hoher Recall eine geringe Präzision nach sich zieht. Je nach Art der Suche wird entweder auf die Präzision oder auf den Recall größeren Wert gelegt. Um sich einen Überblick über ein Gebiet zu verschaffen, ist ein hoher Recall von Vorteil; wird nach einer ganz bestimmten Ressource gesucht, ist eher eine große Präzision zu bevorzugen.

8.2 Probleme bei Multi-Service Suchdiensten

8.2.1 Bedienung

Multi-Service Suchagenten werden von zwei Personengruppen benutzt. Die eine Gruppe besteht aus Personen, die eher unerfahren im Umgang mit dem Computer und speziell den Suchagenten sind. Diese Personengruppe sollte das Suchprogramm einfach bedienen können, ohne für eine Suche erst viele Anpassungen vornehmen zu müssen. Die Suchparameter sollten hierzu mit günstigen Defaultwerten belegt sein, so daß ohne eine Änderung ein befriedigendes Suchergebnis erzielt werden kann. So sollte standardmäßig ein guter Kompromiß zwischen einem hohen Recall und einer hohen Präzision eingestellt sein. Für den Anwender ist es dann nur noch nötig, die Anfrage einzugeben und die Suche zu starten. Dem erfahrenen Suchprogrammnutzer sollten aber Möglichkeiten gegeben werden, die Suche in einer von ihm gewünschten Art und Weise zu beeinflussen. So kann dem Suchenden beispielsweise die Möglichkeit gegeben werden, eine hohe Präzision auszuwählen oder aber auf einen hohen Recall mehr Wert zu legen. Außerdem könnte er die Suche auf ein Teilgebiet des Internets zu beschränken, beispielsweise nur auf Dokumente, die in der WWW-Adresse die Endung *com* besitzen oder auf einer Suche nur in Deutschland. Zudem erweist es sich als zweckdienlich, daß der Anwender Kategorien eingeben kann, zu der die gesuchte Ressource gehört. Es sind Kategorien wie Bilder, Software, WWW-Ressourcen, News, technische Reports, Personen usw. vorstellbar.

Für eine bequeme Bedienung ist ein gutes Eingabefeld vonnöten, d.h. es sollte einfach zu bedienen und übersichtlich sein. Besonders für den im Suchen ungeübten Nutzer ist dies eine sehr wichtig. Allerdingssollte auch die Möglichkeit bestehen, sehr detailliert in den Suchprozeß einzugreifen. Das Suchprogramm könnte beim Aufruf eine einfache Maske anbieten, in der der Suchanfänger nur die Anfrage eingeben muß und die sonst nicht viele Einstellmöglichkeiten bietet. Über einen Auswahlpunkt in der Maske könnte dann der erfahrene Anwender eine zweite Maske auswählen, die alle Menüs und Auswahlpunkte für eine exakte Einflußnahme auf die Suche bietet.

8.2.2 Anfragesprache

Es ist eine Anfragesprache wichtig, die einfach und somit leicht erlernbar ist, aber trotzdem sehr ausdrucksstark. Sie sollte die Anfrage auf mehrere Arten interpretieren können. Es sollte die Suche nach Satzteilen, allen Worten der Anfrage oder einem der Worte unterstützen und von Kombinationen dieser Typen. Beispielsweise sollte die Suche nach dem Satzteil „Sein oder nicht sein“ den wohl bekannten Text von William Shakespeare liefern; hierbei ist es erforderlich, das die Worte nicht nur in dem Dokument vorkommen, sondern auch hintereinander in genau dieser Reihenfolge. Bei einer Suche nach allen Worten müßten diese zwar alle im Dokument vorkommen, aber nicht unbedingt hintereinander und in derselben Reihenfolge. Bei dieser Art der Suche werden deutlich mehr Verweise im Suchergebnis zurückgeliefert. Die Suche nach nur einem der Worte erfordert nur das Vorkommen eines der Worte im Dokument. Es wird hier im Normalfall das umfangreichste Suchergebnis zurückgeliefert. Wenn der Suchende aber nur wenig Informationen über das Dokument besitzt, ist eine andere Form der Suche meistens nicht möglich. Des weiteren sollte es möglich sein, Wörter anzugeben, die nicht in den Dokumenten auftreten dürfen. Hierbei sollten auch die drei Suchtypen Satzteil, alle Worte und einem der Worte sowie der Kombination der Typen unterstützt werden.

Eine von Erik Selberg und Oren Etzioni [3] durchgeführte Analyse ergab, das aufgrund einer gut gestellten Anfrage mit einer hochentwickelten Anfragesprache ca. 40% weniger Verweise durch das Suchprogramm geliefert werden. Diese fehlenden Verweise sind für die Suche jedoch nicht Relevant, da die Suche nur exakter durchgeführt werden konnte. Es wird also sowohl die Relevanz als auch die Präzision des Suchergebnisses deutlich verbessert, ohne gleichzeitig eine Verschlechterung des Recalls hinnehmen zu müssen.

8.2.3 Auswahl der Suchdienste

Ein Problem der Multi-Service Suchagenten besteht in der Auswahl der benutzten Suchdienste. Oft werden so viele verschiedene Suchdienste unterstützt, das es nicht sinnvoll ist, die Anfrage zu jedem weiterzuleiten. Je mehr Suchdienste benutzt werden, um so mehr wird Netz und Rechenzeit beansprucht und die Wahrscheinlichkeit einer langen Suche erhöht. Vor allem steigt die Anzahl der zurückgelieferten Treffer deutlich an, während die Anzahl der relevanten Verweise nur geringfügig verbessert wird; die Präzision des Suchergebnisses läßt also nach. Die Auswahl der Suchdienste sollte das Multi-Service Suchprogramm übernehmen, denn bei einem manuellen Auswählen durch den Benutzer müßte dieser die Schwerpunkte und Besonderheiten aller unterstützten Suchdienste kennen. Ideal wäre es, die Anfrage an nur so wenige Suchdienste weiterzuleiten, das gerade alle relevanten Dokumente gefunden werden. Das ist in der Praxis allerdings so nicht möglich, da man nicht definitiv vorhersagen kann, welcher Suchdienst gesuchte Ressourcen in seiner Datenbank aufgenommen hat.

Um eine Auswahl der Suchdienste treffen zu können, werden die verschiedenen Suchdienste mit Gewichten versehen, die zu einer Gewichtsmatrix zusammengefaßt werden. Für jeden Suchdienst besitzt die Matrix einen Vektor, dessen Elemente die Eignung für die Suche einer Kategorie von Ressourcen beschreiben. Um der ständigen Veränderung des Word Wide Webs und der einzelnen Suchdienste Rechnung zu tragen, sollte die Gewichtsmatrix immer wieder aktualisiert werden. Eine Möglichkeit besteht darin, nach jeder Suche die Relevanz des Suchergebnisses für diese Kategorie zu bewerten und die Gewichte dementsprechend anzupassen.

Aus der Anfrage muß der Multi-Service Suchdienst nun eine Entscheidung über die zu benutzenden Suchdienste treffen. Um den Entscheidungsprozeß zu unterstützen, sollte der Benutzer als zusätzliche Information die Kategorie der gesuchten Ressource eingeben können. Jetzt kann der Multi-Service Suchagent mit Hilfe der Gewichte eine Rangordnung aufstellen, welche Suchdienste für die Anfrage dieser Kategorie am besten geeignet sein müßten. Entsprechend dieser Rangordnung sollten die Suchdienste dann für die Suche ausgewählt werden.

Die Anzahl der zu verwendenden Suchdienste kann durch den Suchenden manuell eingegeben werden, oder aber automatisch durch den Multi-Service Suchagenten ermittelt werden. Zur automatischen Auswahl ist beispielsweise folgendes Verfahren denkbar. Ein Suchdienst nimmt an der Suche teil, sofern sein Gewicht für diese Anfrage eine festgelegte Schranke überschreitet. Falls jedoch zu viele oder zu wenige Suchdienste diesen Test bestehen, sollte eine maximale und eine minimale Anzahl von Suchdiensten festgelegt sein, damit eine vernünftige, ausreichend breite Suche sichergestellt werden kann.

8.2.4 Elimination irrelevanter Verweise

Bei der Elimination von Verweisen sollten als erstes die Duplikate eliminiert werden. Für den Suchenden ist es ausreichend, wenn jeder Verweis nur einmal im Suchergebnis auftaucht. Duplikate entstehen, wenn mehrere Suchdienste die gleiche Ressource gefunden haben oder aber wenn ein Suchdienst selber keine Duplikatelimination durchführt. So können schon einmal eine ganze Reihe von unnötigen Verweisen entfernt werden.

Da nicht alle unterstützten Suchdienste die gleiche ausdrucksstarke Anfragesprache des Multi-Service Suchagenten verwenden, können diese oft nicht alle Feinheiten bei der Suche berücksichtigen. Hier mußte dann die Anfrage in eine allgemeinere, dem untergeordneten Suchdienst verständliche Anfrage umgewandelt werden. Deshalb sollte das Multi-Service Suchprogramm noch einmal das zurückgelieferte Suchergebnis auf die exakte Anfrage des Anwenders hin überprüfen. Desweiteren können bei einer Einschränkung der Suchergebnisse auf die physikalische Lage wie das Land des Anwenders oder einen logischen Domain wie *com* oder *edu* noch einmal ca. 20% der Treffer eliminiert werden.

Von den zurückgelieferten Verweisen sind ca. 15% nicht verfügbar oder es kann nicht auf die Ressource zugegriffen werden. Eine Möglichkeit, diese Informationen aus dem Suchergebnis zu löschen, besteht darin, den Verweis einfach zu laden. So kann eindeutig festgestellt werden, das die Ressource verfügbar ist. Außerdem kann noch einmal genauer nachgeprüft werden, ob das Dokument relevant für die Anfrage ist. Dieses Verfahren eliminiert äußerst effizient irrelevante Daten, es wird sowohl die Präzision als auch der Recall der Anfrage erhöht, allerdings auf Kosten der Dauer der Suche. Eine auf dem MetaCrawler durchgeführte Studie ergab, das für eine automatische Überprüfung mit dem Laden aller Verweise ca. die fünffache Zeit erforderlich ist. Dafür sind die Ressourcen dann im lokalen Cache des Benutzers vorhanden und das spätere Laden der gesuchten Ressourcen durch den Suchenden ist deutlich schneller.

8.2.5 Präsentation der Ergebnisse

Die nach der Elimination verbleibenden Ergebnisse sollten dem Anwender nicht einfach so unüberlegt aufgelistet werden. Die meisten Suchdienste liefern die Suchergebnisse bereits vorsortiert nach vorhergesagter Relevanz zurück. Besitzt der Multi-Service Suchdienst noch zusätzliche Informationen über die Art der gesuchten Ressource wie die Kategorie, so kann er diese Reihenfolge noch einmal überarbeiten. Dann wird jedes Suchergebnis mit der Gewichtsmatrix verrechnet. Dabei sollte zusätzlich für jeden Verweis die Anzahl der Suchdienste mit berücksichtigt werden, die ihn gefunden haben. So ist eine von mehreren Suchdiensten gefundene Ressource wahrscheinlich von größerer Bedeutung, als eine nur von einem Suchdienst gelieferte. Mit Hilfe der überarbeiteten Gewichte kann das Suchergebnis jetzt in der wahrscheinlich günstigsten Reihenfolge präsentiert werden. Hat der Suchende noch eine maximale Anzahl von aufzulistenden Verweisen ausgewählt, so werden die überzähligen mit den geringsten Gewichten einfach weggelassen. Der Anwender sollte auswählen können, ob er nur die Titel aufgelistet bekommen möchte, oder aber auch jeweils die Dokument-Kurzinformation, falls sie vorhanden ist. Damit der Suchende sich schnell einen Überblick über die Relevanz eines im Suchergebnis aufgelisteten Dokumentes schaffen kann, empfiehlt es sich bei der Präsentation, jedes Wort der Suchanfrage hervorzuheben.

8.3 Analyse von Multi-Service Suchagenten

8.3.1 MetaCrawler

Der MetaCrawler wurde von Erik Selberg und Oren Etzioni von der Universität Washington entwickelt. Es wurde angestrebt, die Relevanz der Suche zu erhöhen, bei gleichzeitiger Verbesserung der Präzision. Der Suchende findet den MetaCrawler im Word Wide Web unter der URL *http://metacrawler .cs.washington.edu:8080* [4]. Derzeit werden neun verschiedene Suchdienste unterstützt, nämlich *Alta Vista, Excite, Galaxy, InfoSeek, Inktomi, Lycos, Open Text, WebCrawler* und *Yahoo*. Die Anfagen werden parallel an alle diese Suchdienste weitergeleitet, ohne eine Auswahl zu treffen. Aufgrund der geringen Anzahl der unterstützten Suchdienste ist dieses Vorgehen ausreichend.

Interface

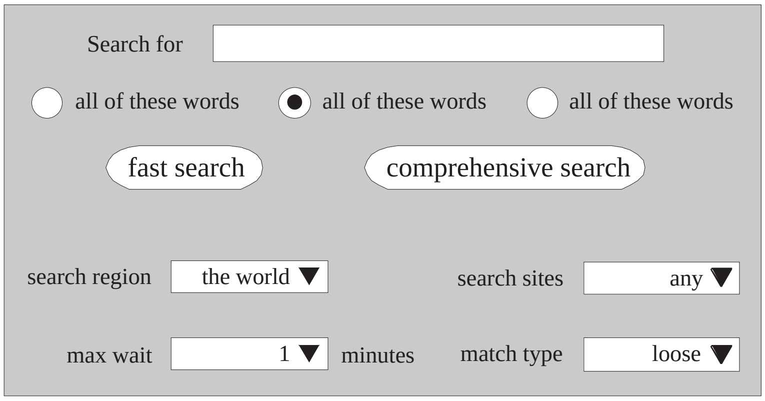


Abbildung 8.1: Das Benutzerinterface des MetaCrawlers

Das Interface des MetaCrawlers ist sehr einfach aufgebaut. Unter dem Eingabefeld für die Anfrage befinden sich drei Auswahlfelder für die Verknüpfung der Worte der Anfrage. Es stehen *as a phrase* für die Suche nach einem Satzteil, die Defaulteinstellung *all of these words* für eine Suche nach allen Worten der Anfrage und *any of these words* für eine Suche nach mindestens einem der Worte zur Auswahl. Die Knöpfe *fast search* und *comprehensive search* starten die Suche. Bei der Auswahl von *fast search* wird eine schnelle Suche gestartet. Dabei werden nicht unbedingt so viele relevante Verweise gefunden, wie bei der Suche im comprehensive Suchmodus. Hier wird auf eine genaue Suche geachtet und auf eine nachträgliche Aufbereitung der Suchergebnisse mehr Wert gelegt.

Weiterhin sind vier Auswahlmenüs vorhanden, mit denen der Anwender detaillierteren Einfluß auf die Suche nehmen kann. Mit dem einen Menü kann er die Suchregion einschränken. Es stehen die Punkte *ganze Welt, eigener Kontinent, eigenes Land, eigener Domain, Nordamerika, Europa, Asien, Australien, Südamerika, Afrika* und *Antarktis* zur Verfügung. Als Defaultwert ist hier *ganze Welt* vorgegeben. Mit dem zweiten Menü kann der Benutzer den Internet Domain auswählen. Hier stehen *beliebig, Company, Educational, Network, Organisation, Government* und *Military* zur Wahl. Als Standardeinstellung ist hier *beliebig* vorgegeben. Mit dem nächsten Menü kann die maximale Suchzeit ausgewählt werden. Standardmäßig ist hier eine Zeit von einer Minute vorgegeben, sie kann aber auch auf drei, fünf, sieben oder zehn Minuten angehoben werden. Falls nach Ablauf der Zeit noch nicht alle Suchdienste ihr Ergebnis geliefert haben, werden nur die vorhandenen Suchergebnisse dem Benutzer präsentiert. Mit dem letzten Menü kann man den für einen Treffer bei der Suche notwendigen Grad der Übereinstimmung festlegen. Zwischen *starker* und *beliebiger* Übereinstimmung stehen noch der Defaultwert *lose* und *mittlere* Übereinstimmung zur Auswahl.

Anfragesprache

Die Anfragesprache des MetaCrawlers ist sehr mächtig, es können praktisch alle möglichen Anfragen gestellt werden. Es wird unterschieden in der Suche nach einem Satzteil, nach allen Wörtern und nach einem der Wörter der Anfrage. Diese Suchmöglichkeiten können schon durch die Auswahlfelder des Suchformulars gewählt werden. Des weiteren sind auch beliebige Kombinationen dieser Suchtypen möglich. Hierzu sind dann schon genauere Kenntnisse des Suchenden erforderlich. Die Verknüpfungen der Worte der Anfrage müssen direkt in den Anfragetext mit eingebaut werden. Die Forderung nach einem Vorhandensein eines Wortes wird durch ein vorangestelltes „+“ gekennzeichnet. Die Suche nach Satzteilen wird durch Klammern um die Worte deutlich gemacht. Bei der Suche nach „Konrad Zuse“ wird als Anfrage (+Konrad +Zuse) eingegeben. Desweiteren ist es möglich, Worte anzugeben, die nicht im dem gesuchten Dokument vorkommen dürfen. Hierzu muß ein „-“ vorangestellt werden. Außerdem sind auch für diese ausschließende Suche die Kombinationen der Suchtypen möglich, beispielsweise der Ausschluß von Satzteilen.

Beinhaltet eine Anfrage eine Suche nach einem Satzteil, so schaltet der MetaCrawler automatisch in den Verifikationsmodus. Dabei lädt er automatisch alle Referenzen und überprüft die Verfügbarkeit. Außerdem kann der MetaCrawler so noch einmal die Relevanz der Dokumente überprüfen und eine genauere Gewichtung für die spätere Präsentation vornehmen. Ohne die Suche nach einem Satzteil bleibt der Verifikationsmodus ausgeschaltet.

Präsentation des Suchergebnisses

Der MetaCrawler sammelt die zurückgelieferten Suchergebnisse der verschiedenen Suchdienste und sortiert die Verweise. Die Duplikate werden entfernt, es werden aber alle Suchdienste festgehalten, die den Verweis gefunden haben. Wenn der Verifikationsmodus gestartet wurde, werden auch die nicht verfügbaren Verweise eliminiert. Außerdem kann er aus dem geladenen Dokument neue Informationen über die Relevanz des Dokumentes bekommen und die Sortierung überarbeiten. Dem Suchenden werden dann die sortierten Verweise auf einer neuen Liste präsentiert. Jeder Verweis besteht aus einem anklickbaren Link zum Dokument und der Dokument-Kurzinformation, falls diese vorhanden ist, dem Sortiergewicht, überprüften Schlüsselwörtern und dem aktuellen URL der Ressource. Dabei wird jedes Wort der Suchanfrage durch Benutzung des Schriftattributes Fettdruck dargestellt.

8.3.2 SavvySearch

SavvySearch ist ein Multi-Service Suchprogramm, das von Daniel Dreilinger entwickelt wurde. Sein Ziel war es, viele verschieden Suchdienste unter einer Schnittstelle zusammenzufassen und verschiedene Verfahren für die Auswahl der benutzten Suchdienste zu entwickeln. Bei einer derzeitigen Unterstützung von 27 verschiedenen Suchdiensten sollten nicht alle in die Suche einbezogen werden. SavvySearch steht dem Nutzer des World Wide Webs unter der URL *http://guaraldi.cs.colostate.edu:2000/form* [2] zur Verfügung.

Interface

SavvySearch bietet das Suchformular in verschiedenen Sprachen an. Derzeit hat der Suchende die Auswahl zwischen 19 verschiedenen Sprachen. Es stehen jedoch nicht alle Funktionen von SavvySearch in jeder der Sprachen zur Verfügung, deshalb wird im Folgenden das Interface in englischer Sprache betrachtet.

Beim Aufruf von SavvySearch wird das einfache Suchformular geladen. Dieses ist einfach zu bedienen, bietet dem erfahrenen Anwender aber nicht alle Einstellmöglichkeiten für eine gezielte Suche. Abbildung 8.2 zeigt das erweiterte Benutzerinterface. Dieses ist um zehn Kontrollfelder erweitert, die in der Abbildung dunkel hinterlegt sind. Durch anklicken des Schaltfeldes „+ sources and types of information“ im einfachen Suchformular werden die Kontrollfelder in das Interface eingefügt. Bei der Nutzung des deutschen Suchformulars sind diese zusätzlichen Kontrollfelder jedoch nicht vorhanden, hier ist nur die Suche mit dem einfachen Suchformular möglich.

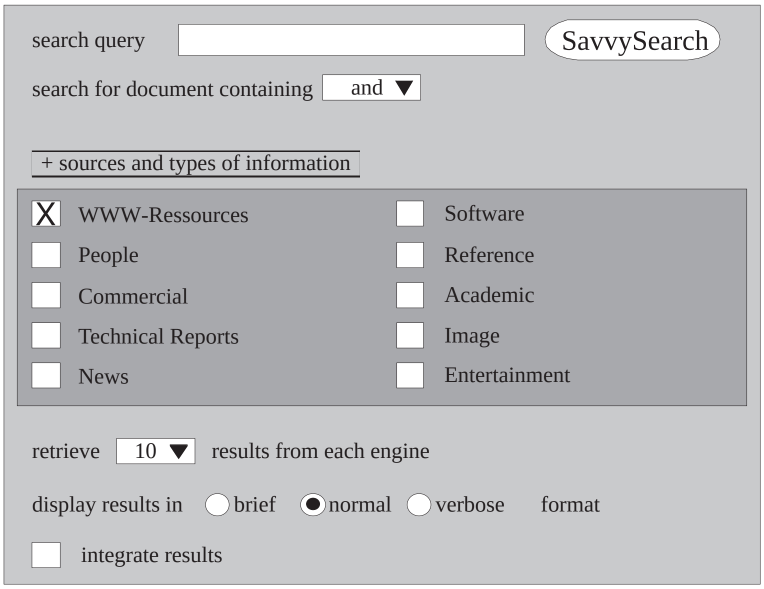


Abbildung 8.2: Das Benutzerinterface von SavvySearch

Ganz oben im Formular ist das Eingabefeld für die Anfrage und das Schaltfeld für den Start der Suche. Im Auswahlmenü darunter kann die Methode der Verknüpfung der Anfrage ausgewählt werden. Es gibt die drei Menüpunkte *und, benachbart* und *oder*, wobei als Defaultwert *und* vorgegeben ist. Im erweiterten Interface stehen dann die zehn Kontrollfelder *WWW-ressources, people, commercial, technical reports, news, software, reference, academic, image* und *entertainment* zur Auswahl. Als Voreinstellung ist hier *WWW-ressources* ausgewählt, der Suchende kann jedoch beliebig viele Kontrollfelder in allen Kombinationen auswählen. Mit dem nächsten Auswahlmenü bestimmt der Anwender die Anzahl der maximal erwünschten Verweise jedes Suchdienstes. Hier ist der Defaultwert zehn Treffer, der Benutzer kann jedoch bis zu 50 Verweise je Suchdienst zulassen. In der folgenden Zeile kann ausgewählt werden, ob das Suchergebnis kurz, normal oder ausführlich dargestellt werden soll. Unten am Benutzerinterface kann über ein Kontrollfeld ausgewählt werden, ob die Suchergebnisse aller Suchdienste zusammengefaßt oder einfach nach Suchdiensten geordnet präsentiert werden sollen.

Anfragesprache

Die Anfragesprache von SavvySearch ist nicht sehr ausdrucksstark. Im Eingabefeld für die Anfrage können mehrere Worte oder Wortstücke eingegeben werden, nach denen gesucht werden soll. Es können Dokumente gesucht werden, die mindestens eins der Worte, alle Worte oder alle Worte benachbart enthalten. Die Art der Verknüpfung wird durch das Auswahlmenü des Suchformulars ausgewählt, eine richtige Anfragesprache gibt es eigentlich nicht. Deshalb ist es auch nicht möglich, die verschiedenen Suchmöglichkeiten miteinander zu kombinieren oder anzugeben, daß ein Suchstring nicht in dem Dokument vorkommen darf.

Auswahl der Suchdienste

SavvySearch geht bei der Auswahl der Suchdienste folgendermaßen vor. Der Suchende muß die Anfrage um eine kleine Information erweitern, und zwar die Kategorie der gesuchten Ressource. Basierend auf der Auswahl einer oder mehrerer Kategorien kann mit Hilfe der internen Gewichtsmatrix eine Rangfolge der Eignung der Suchdienste für diese Anfrage aufgestellt werden. Unter Beachtung dieser Reihenfolge kann durch ein Suchdienst-Auswahlschema die wahrscheinlich beste Auswahl für die Suche getroffen werden.

Um das Suchergebnis ständig zu verbessern, wird nach jeder Suche eine Anpassung der Gewichtsmatrix durchgeführt. Bei der Präsentation der Suchergebnisse werden dem Nutzer zusätzlich zwei Schaltflächen angeboten, mit denen er durch einfaches Anklicken positives oder negatives Feedback über sein spezielles Suchergebnis geben kann. Mit einem Gewichtsanpassung Schema wird die Gewichtsmatrix neu angepaßt; im Falle eines positiven Feedbacks werden die betroffenen Gewichte angehoben, bei negativem Zuspruch werden die Gewichte verkleinert.

Schemata zur Suchdienstauswahl und Gewichtsanpassung

Bei der Entwicklung von SavvySearch mußten erst einmal Algorithmen entwickelt werden, um die Gewichtsmatrix anzupassen und um die beste Auswahl der Suchdienste zu treffen. Beide Probleme wurden angegangen, indem mit einer einheitlich initialisierten Matrix angefangen wurde, auf die verschiedene Schemata zur Optimierung der Matrix angewandt wurden [1]. Es wurden insgesamt vier verschiedene Verfahren untersucht, die durch zwei charakteristische Werte unterschieden werden können. Der eine besteht in der Auswahl der Suchdienste für eine bestimmte Anfrage; der andere im Verfahren zur Manipulation der Gewichtsmatrix. Diese Verfahren sollen im Folgenden genauer betrachtet werden.

- **Schema 0** wählt einfach zufällig die Suchdienste aus. Es wird keine Anpassung der Gewichtsmatrix vorgenommen, das Feedback des Anwenders wird in keiner Form genutzt. Die Anzahl der zu benutzenden Suchdienste wird ebenfalls zufällig ausgewählt bis zur Höchstanzahl von sieben Stück. Dieses rein zufällig arbeitende Schema wurde eingeführt, um einen Vergleich mit den intelligenteren Verfahren ziehen zu können. Es sollte eine schlechtere Performance von Schema 0 gegenüber den anderen Verfahren festgestellt werden können.

- **Schema 1** wählt die n besten Suchdienste aus, streng nach der Summe der Gewichte der ausgewählten Kategorien. Das Feedback in diesem Schema wird durch Addition oder Subtraktion der entsprechenden Gewichte um einen konstanten Faktor zur Gewichtsanpassung verwendet. Anschließend wird jeder Vektor normiert, so daß die Summe der Elemente eins ergibt. Bei dieser Art der Gewichtsanpassung wird das Anwenderfeedback als Steigung benutzt.

- **Schema 2** benutzt den gleichen Ansatz zur Suchdienstauswahl wie Schema 1. Es wird lediglich ein anderes Verfahren zur Gewichtsanpassung verwendet. Anstatt hier einen konstanten Faktor zur Veränderung der Gewichte zu wählen, wird ein exponentiell fallender Wert benutzt. Dieses Schema trägt der Idee Rechnung, das ein anfängliches Feedback mehr Auswirkungen haben sollte, als ein späteres. Zu diesem Zeitpunkt ist die Matrix bereits durch eine große Zahl von Rückmeldungen gut angepaßt. Die Charakteristik wird durch eine Multiplikation der Konstante aus Schema 1 mit e^{-x} erzielt, wobei x eine Zahl aus dem Intervall (0,5) ist. Die Idee zu diesem Verfahren stammt vom abnehmenden Parameter des simulated annealing Search.

- **Schema 3** benutzt dasselbe Verfahren zur Gewichtsanpassung wie Schema 2, aber ein anderes Verfahren zur Suchdienstauswahl. Hier wird bevor die Sortierung durchgeführt wird ein exponentiell fallender, völlig zufälliger Wert dem Gewicht des Suchdienstes hinzuaddiert. In der sich so ergebenden Reihenfolge werden die n besten Suchdienste ausgewählt. Es wurde das gleiche Intervall von e^{-x} gewählt. Die Idee des Verfahrens besteht darin, am Anfang bei der Initialisierungsphase die Suchdienste komplett zufällig auszuwählen. Es wird dann allmählich immer weiter dazu übergegangen, anhand der durch das Feedback angepaßte Gewichtsmatrix die Auswahl zu treffen. Es ist zu beachten, daß der Zufallswert in die Suchdienstauswahl und nicht in die Gewichtsanpassung der Matrix eingeht. Andernfalls könnte die Gewichtsanpassung verfälscht werden.

Mit jedem der vier Verfahren wurde eine Studie durchgeführt, es wurden jeweils über 750 Anfragen, über 50 Feedbackmeldungen und über 160 Gewichtsanpassungen betrachtet. Für die Gewichtsanpassung wurde folgende Frage verwendet: Ist das Suchergebnis für deine Anfrage nützlich? Der Suchende konnte zwischen den zwei Auswahlfeldern ja oder nein auswählen nachdem er die durch SavvySearch gefundenen Ressourcen betrachtet hat.

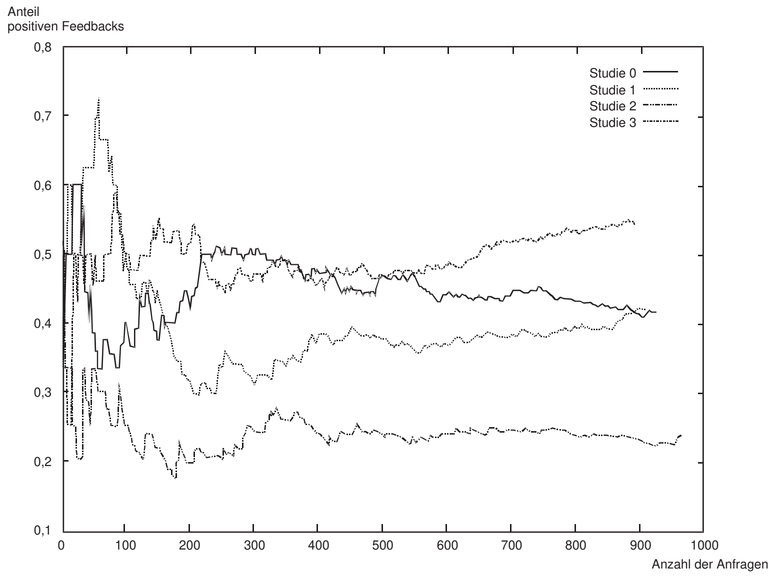


Abbildung 8.3: Vergleich der vier Schemata

Abbildung 8.3 zeigt den prozentualen Anteil des ja Feedbacks im Verhältnis zur Anzahl der Anfragen der vier Schemata. Bei der Betrachtung von Schema 0 fällt auf, daß es nicht unbedingt schlechter abschneidet als die anderen. Es liegt vielmehr fast die ganze Zeit der Untersuchung auf Platz 2. Bei diesem Schema ist es allerdings schwer, die abfallende Tendenz in der Bewertung zu erklären. Da die Auswahl der Suchdienste völlig zufällig erfolgt, erwartet man eigentlich eine konstante Gerade. Vielleicht läßt sich die abfallende Tendenz mit der Gewöhnung der Suchenden an Schema 3 erklären, durch das ein besseres Suchergebnis erwartet wird.

Schema 1 bewegt sich im Mittelfeld der Schemata. Es allerdings liegt fast die ganze Zeit hinter dem völlig zufälligen arbeitendem Schema 0, erst ganz am Ende der Untersuchung überholt es dieses Schema. Die ganze Zeit ist allerdings ein Aufwärtstrend festzustellen. Man kann vermuten, daß bei einer längeren Betrachtung die Anpassung der Gewichtsmatrix immer weiter verbessert wird.

Dem Betrachter fällt ins Auge, das Schema 2 äußerst schlecht bewertet wird. Nur ca. 25% der Suchenden bewerteten das Suchergebnis als nützlich, alle anderen Schemata sind deutlich besser. Die beste Erklärung für dieses Phänomen scheint darin zu liegen, daß in der Anfangsphase, als jedes Feedback große Auswirkungen auf die Gewichtsmatrix hat, bestimmte Suchdienste negatives Feedback erhalten haben. Obwohl sie für die Suche in einer bestimmten Kategorie sehr gut geeignet waren, wurde das Gewicht als sehr niedrig festgesetzt. Da bei diesem Schema keine zufällige Komponente in die Suchdienstauswahl mit einfließt, wurde dieser Suchdienst so lange nicht mehr für die Suche in dieser Kategorie benutzt, bis alle anderen schlechter geeigneten Suchdienste hinter ihm angeordnet wurden. Auf diese Weise kann natürlich nicht sehr viel positives Feedback gegeben werden.

Über die Dauer dieser Studie ist Schema 3 eindeutig am besten geeignet. Nach Schwankungen in der anfänglichen Initialisierungsphase steigen die positiven Rückmeldungen ständig an. Der große Vorteil von diesem Schema besteht darin, das selbst bei negativem Feedback ein Suchdienst nie völlig ausgeschlossen wird. Besonders bei Kategorien, die mehrere identische Gewichte für verschiedene Suchdienste besitzen, wird so eine gute Mischung bei der Auswahl der Suchdienste getroffen.

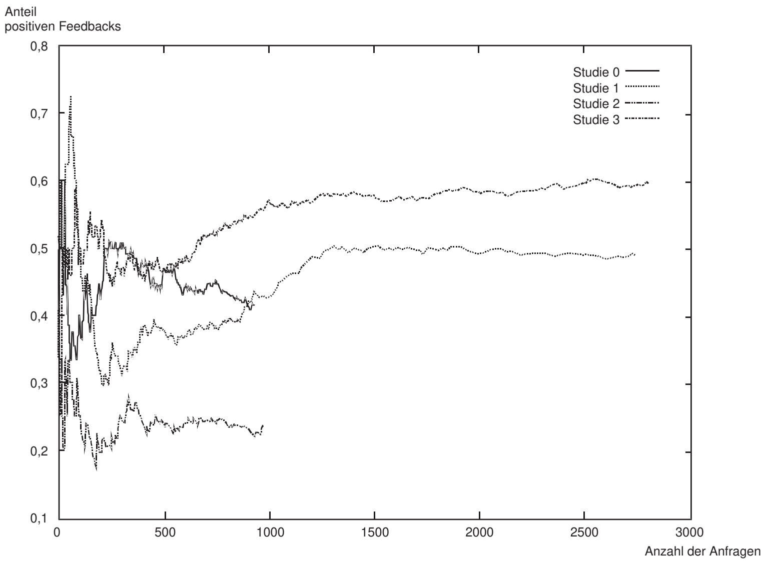


Abbildung 8.4: Vergleich der Schemata

Abbildung 8.4 zeigt eine erweiterte Untersuchung von Schema 1 und Schema 3, bei der über 1700 Anfragen mit einbezogen wurden. Diese Untersuchung soll die Annahmen untermauern, daß über einen langen Zeitraum hinweg die Suchdienstauswahl immer besser erfolgen kann. Tatsächlich steigt die Zufriedenheit mit der Suchdienstauswahl von Schema 3 ständig an, bis sie bei über 60% stagniert. Auch Schema 1 steigt eine Zeitlang an, bis bei ca. 50% positivem Feedback eine nahezu konstanter Wert eingehalten werden kann. Dauerhaft scheint sich Schema 3 jedenfalls von den vier Schemata am besten dazu eignen, die Suchdienstauswahl und die Anpassung Gewichtsmatrix durchzuführen.

Präsentation der Suchergebnisse

Der Anwender kann auswählen, ob die Suchergebnisse zusammengefaßt werden sollen oder nicht. Standardmäßig werden die Suchergebnisse nicht zusammengefaßt. Das ergibt den Vorteil, das SavvySearch die Ergebnisse schneller präsentieren kann. Sobald ein Suchdienst das Suchergebnis zurückgeliefert hat, wird es dem Benutzer präsentiert. Es muß nicht erst auf den langsamsten Suchdienst gewartet werden, allerdings können keine Duplikate entfernt werden. Außerdem können die Suchergebnisse nur nach der Sortierung des benutzten Suchdienstes geordnet werden.

Werden die Ergebnisse zusammengefaßt, so dauert die Suche etwas länger. Dafür werden jedoch die Duplikate der Verweise eliminiert und es kann eine bessere Reihenfolge des Suchergebnisses erzielt werden. Die Suchergebnisse werden jetzt entsprechend der Gewichtsmatrix für die Suchdienstauswahl sortiert. Dabei gibt es jedoch zwei Verfeinerungen. Wenn mehrere Suchdienste Verweise auf die gleiche Ressource liefern, werden die Gewichte addiert. Außerdem werden die von manchen Suchdiensten gelieferten Gewichtungen des Suchergebnisses normiert und zu 10% den Gewichten der Gewichtsmatrix zugerechnet. In der sich auf diese Weise ergebenden Reihenfolge wird das Suchergebnis jetzt präsentiert.

Den Umfang der gelieferten Information kann der Anwender im Suchformular auswählen. Bei der Präsentation im kurzen Format werden nur der Name des Dokumentes und der erfolgreiche Suchdienst aufgelistet. Wählt der Suchende das normale Format für die Präsentation, so listet SavvySearch zusätzlich jeweils die Dokument-Kurzinformation mit auf. Im ausführlichen Präsentationsformat werden alle Informationen zur jeweiligen Ressource aufgelistet, die SavvySearch zur Verfügung stehen.

Literaturverzeichnis

[1] Daniel Dreilinger. Integrating heterogeneous www search engines. 1995.

[2] Daniel Dreilinger. Savvysearch homepage. *URL http://guaraldi.cs .colostate.edu:2000*, 1996.

[3] Erik Selberg und Oren Etzioni. Multi-service search and comparison using the metacrawler. *URL http://metacrawler.cs .washington.edu:8080/papers/www4/html/Overview*, 1995.

[4] Erik Selberg und Oren Etzioni. Metacrawler homepage. *URL http:// metacrawler.cs.washington.edu:8080/home.html*, 1996.

Kapitel 9

WebWatcher: Lernen von Benutzergewohnheiten

Andreas Eckert

Übersicht

Diese Arbeit beschreibt einen Assistenten, genannt WebWatcher, der den Benutzer bei der Informationssuche im World Wide Web durch Ratschläge unterstützt. Zuerst wird die Benutzung des Programms anhand eines Beispiels erläutert. Dabei soll nicht nur die Bedienung gezeigt werden, sondern auch ein Einblick in die Verarbeitungsmethodik vermittelt werden. Danach folgt die Betrachtung des Assistenten unter dem Aspekt des *Maschinellen Lernens*, wobei Methoden aus diesem Bereich aufgeführt werden, die besonders geeignet erscheinen. Abschließend wird versucht, die Güte der Ratschläge mit Hilfe einiger Tests zu bewerten und zu verbessern.

9.1 Einleitung

Mit Einführung des World Wide Web (WWW) erhalten immer mehr Benutzer Zugang zu einer Informationsfülle, aus der sie ohne fremde Hilfe kaum noch bestimmte Informationen filtern können. Es gibt zwar bereits Index- und Inhaltsverzeichnisse (z. B. Alta Vista und Yahoo), die die Suche im Netz erleichtern, jedoch beschränken diese sich auf eine reine Kartographierung des WWW und sind nicht in der Lage, dem Benutzer interaktiv Ratschläge zu geben oder selbständig auf die Suche nach Informationen für den Anwender zu gehen. Diese Lücke soll von sogenannten Agenten oder Assistenten gefüllt werden, zu denen auch *WebWatcher* [1, 2] gehört.

Nachdem der Benutzer das Ziel seiner Suche angegeben hat, wird er vom WebWatcher-System unterstützt, indem es ihm vorschlägt, welchen Hyperlink er als nächstes wählen sollte; ferner kann der Anwender eine Liste von Hyperlinks abfragen, die mit der aktuellen Seite in einem inhaltlichen Zusammenhang stehen. Dabei lernt der Agent sowohl durch Beobachtung der Reaktion des Benutzers auf seine Ratschläge als auch durch Erfolg oder Mißerfolg der Suche.

Bei WebWatcher handelt es sich um ein Forschungsprojekt der *School of Computer Science* der *Carnegie Mellon University*. Es befindet sich noch in der Entwicklungsphase und wird ständig verbessert und erweitert. Dadurch kann es unter Umständen vorkommen, daß das System nicht erreichbar ist oder nicht antwortet.

9.2 Wie arbeitet WebWatcher?

Diese Frage soll durch ein Beispiel, in dem die Benutzung von WebWatcher gezeigt wird, beantwortet werden. Um das System zu starten, muß man sich auf einer Seite[1] befinden, die einen Hyperlink auf das *WebWatcher Front Door* enthält. Folgt man diesem Hyperlink, erhält man ein Formular, in dem man seine e-mail Adresse angeben kann und das Ziel der Suche beschreiben muß. Siehe hierzu auch [4]. Danach befindet man sich wieder auf der Seite, von der man gekommen ist, die jetzt aber von WebWatcher überwacht und erweitert wird. Die Seite wird durch die folgenden Ergänzungen modifiziert: (1) Am Anfang der Seite wird ein Menü eingefügt, mit dem man zusätzliche Funktionen abrufen oder die Suche beenden kann. (2) Unter dem Menü werden einige Hyperlinks eingefügt, von denen WebWatcher glaubt, daß sie für den Benutzer interessant sind, da sie mit der Seite, von der der Benutzer ins System gesprungen ist, in einem inhaltlichen Zusammenhang stehen. (3) Das System markiert alle Hyperlinks auf der ursprünglichen Seite, von denen es glaubt, daß sie zum Ziel führen, durch *Augen* vor und hinter dem Hyperlink. Dabei gibt die Größe der Augen an, wie sicher WebWatcher mit seinem Ratschlag ist. (4) Alle Hyperlinks auf der Seite zeigen auf das WebWatcher-System und haben ihr ursprüngliches Ziel als Parameter. Dadurch weiß WebWatcher, welchen Hyperlink der Benutzer gewählt hat, und welche Seite er als nächstes modifizieren muß. Der Anwender hat

[1] z. B. http://www.ai.univie.ac.at/oefai/ml/ml-ressources.html

jetzt die Wahl, ob er sich auf das System verläßt und einem Ratschlag folgt, oder ob er einen Hyperlink wählt, der zwar nicht empfohlen wurde, der den Benutzer aber interessiert. D.h. der Benutzer hat immer die Kontrolle über das System; er muß den Ratschlägen nicht folgen, sondern kann seine eigenen Suchkriterien anwenden. Solange der Anwender die Suche nicht abbricht, wird WebWatcher auf jeder weiteren Seite das Menü einfügen und Ratschläge geben. Eine Sequenz dieses Zyklus ist in den Abbildungen 9.1 bis 9.3 zu sehen.

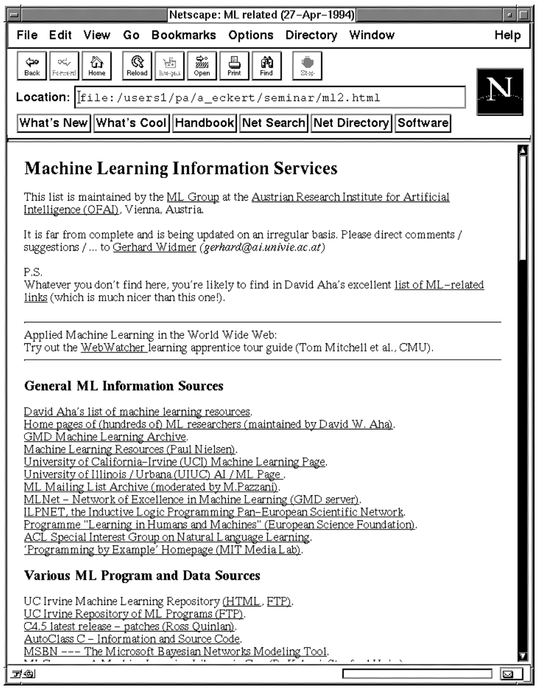


Abbildung 9.1: Originalseite

Im Beispiel gibt der Benutzer als Ziel `Machine Learning` an (Abbildung 9.2). Das System überarbeitet jetzt die Originalseite unter dem Aspekt Machine Learning und bietet dem Anwender eine Übersicht von Hyperlinks (Abbildung 9.3, zweiter Absatz), die mit der aktuellen Seite inhaltlich zusammenhängen; dann markiert es die Hyperlinks auf der Originalseite, von denen es glaubt, daß sie mit Machine Learning zu tun haben, durch Einfügen der WebWatcher-Augen (Abbildung 9.3 z. B. `list of ML-related links`). Durch diese Hervorhebung weist das System auf seine Ratschläge hin. Folgt der Benutzer dem Hyperlink `list of ML-related links`, so gelangt er auf dessen modifizierte Seite, auf der WebWatcher wieder neue Ratschläge hervorgehoben hat. Die Suche wird so lange fortgesetzt, bis der Benutzer sie durch Drücken von `Exit: Goal Reached` oder `Exit: Goal Not Found` beendet.

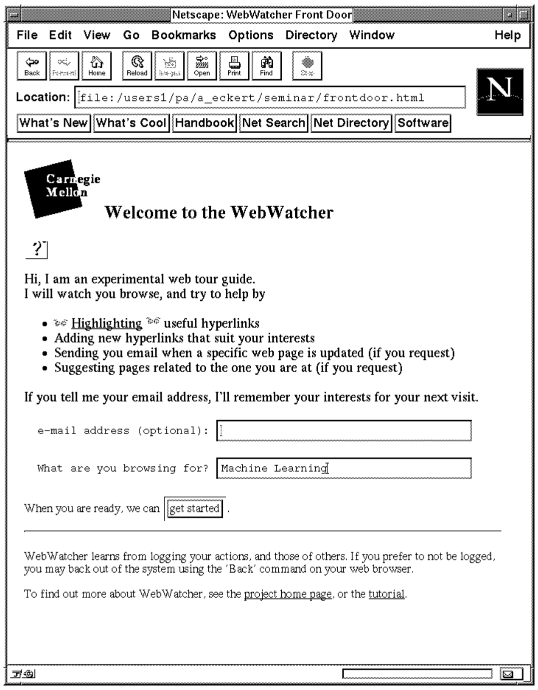


Abbildung 9.2: WebWatcher Front Door

Aus Sicht des Benutzers verhält sich WebWatcher wie ein Agent, der über spezielles Wissen über die Suche außerhalb der aktuellen Seite verfügt. Das System gibt Ratschläge, welchen Hyperlink der Benutzer wählen sollte, wobei dieser aber die völlige Kontrolle über die Suche hat und die Ratschläge des Agenten ignorieren kann. Das ist wichtig, da es vorkommen kann, daß WebWatcher die Zielvorgabe nicht richtig versteht oder falsch deutet.

Aus Sicht des WebWatchers stellt sich die Situation etwas anders dar. Beim ersten Aufruf des Systems erhält es die Rücksprungadresse der Seite, die es aufruft, kodiert in der URL als Parameter. Als Rücksprungadresse wird die URL der Originalseite bezeichnet; diese wird benötigt, damit WebWatcher den Benutzer, nachdem er das Ziel angegeben hat, auf die Originalseite zurückbringen und diese erweitern kann. Das Menü besteht für WebWatcher lediglich aus einigen Hyperlinks, die zurück auf das System zeigen und ihm zusätzlich einen Parameter übergeben. Durch Auswertung dieser Parameter kann der Agent entscheiden, welche Funktion als nächstes ausgeführt werden muß. Die zusätzlichen Hyperlinks, die unter dem Menü stehen, werden von der Funktion *Related* ausgewählt. Diese Funktion kann auch über das Menü auf jeder anderen WebWatcher-Seite angestoßen werden. Auf *Related* wird im Abschnitt 9.3 auf Seite 146 näher eingegangen. Die Funktion *User-Choice?* ermittelt die Hyperlinks auf der Seite, die mit einer hohen Wahrscheinlichkeit mit dem Ziel zusammenhängen. Hyperlinks mit einer hohen

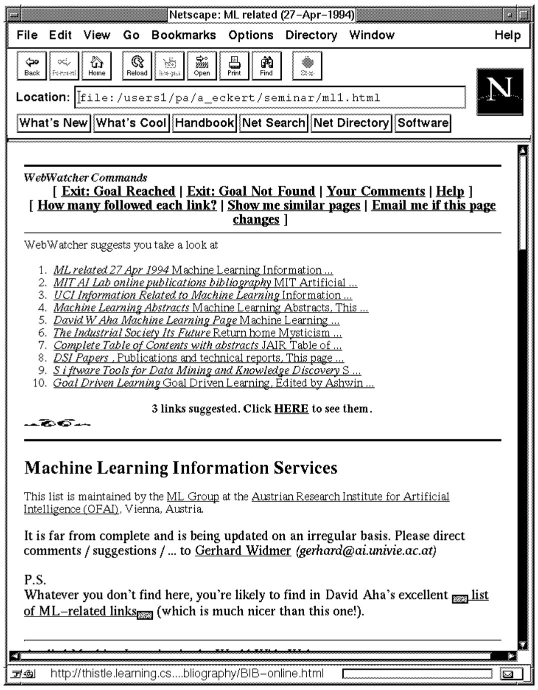


Abbildung 9.3: Originalseite (Abb. 9.1) mit Ergänzungen

Wahrscheinlichkeit erhalten Augen als Markierung, solche mit einer geringen Wahrscheinlichkeit erhalten keine Augen. Auch diese Funktion wird im Abschnitt 9.3 näher besprochen. Schließlich werden alle Hyperlinks durch erweiterte Hyperlinks ersetzt, die auf das WebWatcher-System zeigen und Parameter, wie z.B. das Ziel, enthalten. Dem Benutzer wird dann die modifizierte Seite gezeigt, und es wird eine Logdatei eröffnet, in der die Suche und ihr Ergebnis protokolliert werden. Diese Logdatei wird später zum Lernen durch das System ausgewertet. Während das System auf die nächste Eingabe durch den Benutzer wartet, lädt es bereits die Seiten, die es gerade empfohlen hat und sucht jeweils den Hyperlink mit der höchsten Wahrscheinlichkeit. Wählt der Anwender einen Hyperlink, so lädt der Agent die zugehörige Seite aus dem Netz oder, falls er sie bereits geladen hat, aus seinem internen Speicher. Dabei erweitert er die neue Seite wieder, und übergibt sie dem Benutzer.

WebWatcher protokolliert und modifiziert so lange, bis der Benutzer einen der beiden `Exit`-Hyperlinks auswählt. Dann vermerkt das System in der Logdatei, ob die Suche für den Benutzer erfolgreich war oder nicht, schließt die Logdatei und übergibt dem Anwender die letzte Seite ohne Ergänzungen.

9.3 Maschinelles Lernen

Qualität und Quantität des erlernten Wissens sind die wesentlichen Einflußfaktoren auf das Ergebnis von WebWatcher. Da sich das World Wide Web sowohl in seiner Struktur als auch in seinem Inhalt verändert, müßte dieses Wissen ständig von Hand erweitert und angepaßt werden. Deshalb sucht man nach Methoden, die dieses Wissen selbständig erlernen und verwalten können.

9.3.1 Die Funktion *Related*

Für WebWatcher ist das World Wide Web ein gerichteter Graph, dessen Knoten die Seiten und dessen Kanten die Hyperlinks repräsentieren. D. h. der Graph enthält eine Kante von P nach Q genau dann, wenn auf der Seite P ein Hyperlink l_Q existiert, der auf die Seite Q zeigt. Der Graph enthält zwei Arten von Wissen. Zum einen ist Wissen in den Seiten in Form von Text, zum anderen stecken aber auch in den Kanten Informationen. Das Wissen in den Seiten wird mit der Funktion *UserChoice?* ausgewertet, während *Related* die Informationen in den Kanten benutzt.

Der Agent speichert den Graphen als Adjazenzmatrix. In Tabelle 9.1 ist ein Ausschnitt aus der Matrix gegeben. Eine 1 bedeutet, daß von der Zeilenseite

Seite	WWatcher	LearnLab	ILPNet	$\cdots$
Tom	1	1	0	
Dayne	1	1	0	
Thorsten	1	1	0	
Projects	1	1	0	
Katharina	0	0	1	
MLResour	1	0	1	
$\vdots$				$\ddots$

Tabelle 9.1: Ausschnitt aus der Adjazenzmatrix

ein Hyperlink auf die Spaltenseite zeigt; eine 0 bedeutet analog, daß kein Hyperlink auf die Spaltenseite zeigt. D. h., daß z. B. auf der Seite von *Tom* ein Hyperlink auf die *WWatcher*-Seite existiert, aber auf der Seite von *Dayne* kein Hyperlink auf die *ILPNet*-Seite ist. Für die *WWatcher*-Seite könnte man aus dem Graphen folgende Informationen gewinnen:

Tom, Dayne und Thorsten finden die Seite interessant, denn sie haben einen Hyperlink von ihren persönlichen Seiten auf die WWatcher-Seite. Außerdem *ist die Seite über ein Projekt an der CMU und hat etwas mit Maschinellem Lernen zu tun.*

Die Aufgabe besteht nun darin, von einer gegebenen Seite (z. B. *WWatcher*) die Seiten zu finden, die ähnliche Informationen haben. *Related* ist also eine Funktion, die eine Seite auf eine Menge von Seiten abbildet:

$$Related : Seite \longrightarrow \{\ddot{a}hnliche\ Seiten\}$$

Related nimmt an, daß zwei Seiten ähnlich sind, wenn weitere Seiten jeweils Hyperlinks auf beide Seiten besitzen. Das bedeutet also, daß zwei Seiten ähnlich sind, wenn ihre Spalten in der Adjazenzmatrix ähnlich sind. In der Beispielmatrix ist die *LearnLab*-Spalte der *WWatcher*-Spalte wohl am ähnlichsten. Es gilt also:

$$LearnLab \in Related(WWatcher)$$

Die Funktion gibt die Hyperlinks der n ähnlichsten Spalten aus.

Um Spalten auf Ähnlichkeit zu kontrollieren, überprüft man, wie gut das Aufreten eines bestimmten Hyperlinks auf einer Seite das Auftreten eines anderen Hyperlinks auf derselben Seite voraussagt. Im Beispiel prüft man also: wie gut sagt das Auftreten eines Hyperlinks auf die *LearnLab*-Seite das Auftreten eines Hyperlinks auf die *WWatcher*-Seite voraus? (Wie groß ist die sogenannte *Mutual Information*[2], daß auf einer beliebigen Seite, auf der ein Hyperlink auf *LearnLab* ist, auch ein Hyperlink auf *WWatcher* ist?) Sei D die Menge aller Zeilen der Adjazenzmatrix; weiter sei l_i ein Hyperlink auf die Seite P_i. In D ist die *Mutual Information* von l_j bezüglich l_i:

$$I(D, l_i, l_j) = E(D, l_i) - \frac{m_+}{m} * E(D_+, l_i) - \frac{m_-}{m} * E(D_-, l_i)$$

Dabei ist D_+ die Menge aller Zeilen, in denen $l_j = 1$ ist, während D_- die Menge aller Zeilen mit $l_j = 0$ ist. $m = card(D)$ ist die Zahl der Zeilen der Adjazenzmatrix, $m_+ = card(D_+)$ und $m_- = card(D_-)$. $E(D, l_i)$ ist die Entropie von D bezüglich l_i.[3] Im Beispiel berechnet man also:

$$\begin{gathered} I(\{Tom, \ldots, MLResour\}, l_{WWatcher}, l_{LearnLab}) = \\ E(\{Tom, \ldots, MLResour\}, l_{WWatcher}) \\ -0,67 * E(\{Tom, \ldots, Projects\}, l_{WWatcher}) \\ -0,33 * E(\{Katharina, MLResour\}, l_{WWatcher}) \end{gathered}$$

[2] eine Art Ähnlichkeitsmaß. Siehe hierzu [3].

[3] Das ist eine Größe aus der Informationstheorie und wird zur Ermittlung des mittleren Informationsgehalts von D bezüglich l_i verwendet.

Obwohl *LearnLab* mehr Ähnlichkeit mit *WWatcher* hat als *ILPNet*, ist $I(D, l_{WWatcher}, l_{LearnLab}) = I(D, l_{WWatcher}, l_{ILPNet})$. Das liegt daran, daß l_{ILPNet} invers zu $l_{LearnLab}$ ist. Um solche Fehler auszuschließen, werden nur solche l_j untersucht, die nicht invers zu l_i sind. Diese Bedingung wird von dem Attribut $NonComp(D, l_i, l_j)$ auf folgende Weise berechnet:

$$NonComp(D, l_i, l_j) \Leftrightarrow \frac{p_+}{m_+} \geq \frac{p_-}{m_-}$$

p_+ ist die Zahl der Zeilen in D_+, in denen $l_i = 1$ gilt. p_- ist die Zahl der Zeilen in D_-, in denen $l_i = 1$ gilt. Da $NonComp(D, l_{WWatcher}, l_{ILPNet})$ FALSE[4] ist, wird die *ILPNet*-Seite nicht untersucht.

Der folgende Algorithmus implementiert die Funktion *Related*:

`Input: Seite` P_{in}

$\forall$ `Hyperlinks` l_i `mit` $NonComp(D, l_{in}, l_i)$ `= TRUE`

`berechne` $I(D, l_{in}, l_i)$

`Output: die` l_i `mit den` n `besten` *Mutual Information* .

9.3.2 Die Funktion *UserChoice?*

Diese Funktion soll auf einer WWW-Seite die Hyperlinks herausfinden, die den Benutzer zum Ziel seiner Suche führen. Sie sollte dazu für jeden Hyperlink der Seite die Wahrscheinlichkeit berechnen, daß der Benutzer diesen Hyperlink wählt. Dabei werden Hyperlinks mit hoher Wahrscheinlichkeit mit WebWatcher-Augen markiert. Es müßte also folgende Abbildung realisiert werden:

$$LinkUtility : Page \times Goal \times User \times Link \longrightarrow [0, 1]$$

Dabei ist *Page* die aktuelle Seite, *Goal* das vom Benutzer angegebene Ziel, *User* der Benutzer und *Link* der Hyperlink, der gerade untersucht wird. Der Wert von *LinkUtility* ist die Wahrscheinlichkeit, daß dieser Hyperlink auf der aktuellen Seite den Benutzer auf kürzestem Weg zu seinem Ziel führt.

Das System arbeitet nicht mit der Funktion *LinkUtility*, sondern mit der einfacheren Funktion *UserChoice?*.

$$UserChoice? : Page \times Goal \times Link \longrightarrow [0, 1]$$

[4] $\frac{1}{2} \not\geq \frac{4}{4}$

Diese Funktion läßt sich einfacher realisieren, da sie nicht von *User* abhängt und auch nicht den kürzesten Weg sucht. Auch der Weg, den der Benutzer bisher zurückgelegt hat, wird nicht berücksichtigt. Der Wert von *UserChoice?* ist die Wahrscheinlichkeit, daß ein beliebiger Benutzer mit dem Ziel *Goal* auf der Seite *Page* den Hyperlink *Link* wählt. Trotz dieser einfacheren Funktion lassen sich gute Ergebnisse erzielen (vgl. Abschnitt 9.4). Für *UserChoice?* lassen sich auch einfacher die Lernbeispiele beschaffen. Diese Lernbeispiele werden automatisch von der Logdatei erzeugt, die immer, wenn der Benutzer einen neuen Hyperlink wählt, ein Lernbeispiel für jeden Hyperlink auf der aktuellen Seite speichert. Es wird also abhängig vom Ziel, für jeden Hyperlink gespeichert, ob er gewählt wurde oder nicht.

Damit die Funktion *UserChoice?* aus den Lernbeispielen lernen kann, muß zuerst eine Repräsentation von $Page \times Goal \times Link$ gefunden werden, die mit verfügbaren maschinellen Lernmethoden kombinierbar ist und das Wissen effizient auswertbar macht. Während Seiten, Hyperlinks und Ziele textbasiert und oft unstrukturiert sind, erwarten die meisten maschinellen Lernmethoden als Repräsentation strukturierte Daten wie z. B. einen Attribut-Vektor. *UserChoice?* benutzt zur Repräsentation von $Page \times Goal \times Link$ einen Attribut-Vektor, der aus ungefähr 530 Attributen besteht. Dieser Attribut-Vektor setzt sich aus folgenden vier Unter-Attribut-Vektoren zusammen:

1. *Unterstrichene Worte im Hyperlink:* 200 Attribute werden benutzt, um die unterstrichenen Worte im Hyperlink zu kodieren. Diese Attribute repräsentieren die 200 Worte, die den größten Informationsgehalt besitzen.

2. *Worte im Satz, der den Hyperlink enthält:* Weitere 200 Attribute kodieren das Auftreten von 200 ausgewählten Worten im Satz, der den Hyperlink enthält.

3. *Worte in den Überschriften, die mit dem Hyperlink verbunden sind:* 100 Attribute kodieren die wichtigsten Worte in den Überschriften, unter denen der Hyperlink steht. Darunter fallen auch Überschriften von Überschriften, solange der Hyperlink im Bereich der Überschrift ist.

4. *Worte, die das Ziel definieren:* Diese Attribute repräsentieren die Worte, die der Benutzer zur Definition des Ziels verwendet hat. Es werden alle Worte verwendet, die eingegeben wurden.

Die Wichtigkeit der Worte wurde mit folgender Methode bestimmt: zuerst wurden alle Worte nach ihren Unter-Attribut-Vektoren geordnet, dann wurden sie dort nach ihrer *Mutual Information*[5] geordnet und schließlich die jeweils ersten N Worte zur Repräsentation ausgewählt. Diese vier Unter-Attribut-Vektoren wurden zu einem großen Attribut-Vektor zusammengefaßt.

Um verschiedene Lernmethoden zu erforschen und um den Grad der Erlernbarkeit der Funktion zu bestimmen, wurden vier Lernmethoden implementiert, die im Folgenden kurz beschrieben werden:

- Bei **Winnow** handelt es sich um eine Schwellenwert-Funktion. Die Schwellenwerte der Funktion werden durch wiederholte Iteration der Lernbeispiele angepaßt. Dabei wurden die ursprünglichen 530 Attribute erweitert. Jedes Attribut a wurde durch zwei Attribute a und $\bar{a}$ ersetzt. Dabei war ein Attribut gleich dem Originalattribut, während das zweite Attribut die Negation des Originalattributs war. Nachdem die Lernphase abgeschlossen war, wurden die Schwellenwerte entfernt und die Funktion zur Berechnung von *UserChoice?* verwendet.

- **Wordstat** versucht aufgrund statistischer Abschätzungen vorherzusagen, wie wahrscheinlich es ist, daß *Link* gewählt wird. Für jedes Attribut in der $Page \times Goal \times Link$ Repräsentation werden zwei Zahlen ermittelt: die Anzahl des Auftretens des Attributs in allen Lernbeispielen (*total*) und wie oft das Attribut als positiv ausgewählt wurde (*pos*). Das Verhältnis $pos/total$ gibt die Wahrscheinlichkeit an, daß *Link* ausgewählt wird. Sind diese Attribut-Wahrscheinlichkeiten voneinander unabhängig, so kann man mehrere von ihnen zu einer Attribut-Vektor-Wahrscheinlichkeit zusammenfassen. Seien $p_1, \ldots, p_n$ die Attribut-Wahrscheinlichkeiten, so wird die Wahrscheinlichkeit, daß der Hyperlink gewählt wird, durch $1 - \prod_{i=1}^{n}(1 - p_i)$ berechnet.

- **TFIDF** ersetzt im Attribut-Vektor jedes Wort durch eine Zahl, die wie folgt berechnet wird:

  $$V_i = Freq(Word_i) * [ld(n) - ld(DocFreq(Word_i))]$$

  Dabei ist n die Anzahl der Lernbeispiele, $Freq(Word_i)$ die Anzahl des Auftretens von $Word_i$ im aktuellen Beispiel und $DocFreq(Word_i)$ die Anzahl der Lernbeispiele, in denen $Word_i$ vorkommt. Die Länge des Vektors wird dann auf 1 normalisiert, und der Vektor wird einer der Klassen *positiv* oder *negativ* zugeordnet. Die Zuordnung ist davon abhängig, ob *Link* ausgewählt wurde oder nicht. Um nun die Wahrscheinlichkeit eines neuen Vektors zu berechnen, wird der Kosinus des Winkels zwischen dem neuen Vektor und dem *positiv*-Klassen-Vektor vom Kosinus des Winkels zwischen dem neuen Vektor und dem *negativ*-Klassen-Vektor abgezogen.

[5] hier: Wie gut beschreiben die Worte die Lernbeispiele?

- **Random** wurde implementiert, um zu prüfen, um wieviel die Lernmethoden besser als ein zufällig ausgewählter Hyperlink sind.

Mit diesen Lernmethoden hat man einige Versuche durchgeführt, deren Ergebnisse im nächsten Abschnitt besprochen werden.

9.4 Güte der Ratschläge

Um die Güte der Ratschläge zu bewerten, wurden einige Tests mit jeder Lernmethode durchgeführt. Für jede Lernmethode wurden 30 Lernbeispiele in 29 Trainingsbeispiele und ein Testbeispiel geteilt. Die durchschnittliche Anzahl der Hyperlinks pro Seite war 16 und lag zwischen einem und 300. Die Lernmethode lernte die 29 Trainingsbeispiele und wurde dann mit dem Testbeispiel geprüft. Dieser Vorgang wurde 30mal wiederholt, wobei jedes Lernbeispiel einmal zum Testbeispiel wurde. Die Ergebnisse der 30 Tests wurden gemittelt und sind in Tabelle 9.2 zu sehen. Über den Spalten ist angegeben, wieviele Hyperlinks empfohlen werden durften. In den Feldern ist die Häufigkeit angegeben, mit der einer der empfohlen Hyperlinks gewählt wurde. In der Spalte *1 Link* steht also die Häufigkeit mit der der höchstplazierte Ratschlag gewählt wurde, in der Spalte *2 Links* steht die Häufigkeit mit der der höchstplazierte oder der zweithöchstplazierte Ratschlag gewählt wurde und so weiter. Zu beachten ist, daß alle Lernmethoden wesentlich besser als zufällig erzeugte Ratschläge sind. Z. B.: in 30% der Fälle wählt der Benutzer den von **Winnow** höchstplazierten Ratschlag, während er nur in 6% der Fälle den von **Random** empfohlenen Ratschlag wählt.

Lernmethode	1 Link	2 Links	3 Links	4 Links	5 Links
Winnow	30%	48%	56%	64%	66%
Wordstat	24%	49%	56%	61%	68%
TFIDF	24%	39%	51%	56%	63%
Random	6%	15%	22%	27%	32%

Tabelle 9.2: Güte der Ratschläge

Um die Güte der Ratschläge zu verbessern, wurde ein Schwellenwert für *UserChoice?* eingefügt. Liegt der höchstplazierte Ratschlag unter einer Mindestwahrscheinlichkeit, so wird auf dieser Seite kein Hyperlink empfohlen. Tabelle 9.3 gibt die verbesserten Ergebnisse wieder. Über den Spalten ist angegeben, wie häufig das System einen Hyperlink empfiehlt. Je seltener ein Hyperlink empfohlen wird, umso höher ist der Schwellenwert und damit auch die Genauigkeit. Bei einem sehr hohen Schwellenwert wählt der Benutzer in 53% der Fälle (vorher nur 30%) den höchstplazierten Ratschlag

Lernmethode	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Winnow	53%	40%	38%	42%	40%	40%	36%	33%	31%	30%
Wordstat	6%	8%	11%	18%	26%	26%	26%	26%	24%	24%
TFIDF	35%	26%	30%	27%	28%	26%	28%	25%	24%	24%

Tabelle 9.3: Verbesserung der Ratschläge

von **Winnow**, dieser gibt aber nur noch in 10% der Fälle (vorher immer) einen Ratschlag. Die Werte in der *100%-Spalte* von Tabelle 9.3 und die der *1 Link-Spalte* von Tabelle 9.2 sind gleich, da der Agent bei 100% immer einen Ratschlag geben muß. Die schlechten Werte von **Wordstat** lassen sich damit erklären, daß einige unregelmäßig auftauchende Attribute in den Lernbeispielen waren. Außerdem ist die Unabhängigkeitsannahme zwischen den Attributen für die Lernbeispiele nicht korrekt. Zusätzlich bessere Ergebnisse kann man auch durch Erhöhung der Lernbeispiele erwarten.

9.5 Ausblick

Man testet andere Lernmethoden, um bessere Ergebnisse zu erreichen. Da in den Lernbeispielen immer nach technischen Artikeln gesucht wurde, ist das obige Zahlenmaterial für andere Ziele nicht besonders repräsentativ. Trotzdem ist selbst bei schlechten Ratschlägen eine wesentliche Reduktion der Anzahl von Hyperlinks gegeben. Durch diese Vorverarbeitung führt eine Suche effizienter und schneller zum Ziel. Installiert man das System auf verschiedenen Seiten im Netz, so können diese Kopien sich auf bestimmte Gebiete spezialisieren und dadurch bessere Ratschläge geben. Zusätzlich läßt sich das System verbessern, indem es seinen eigenen Ratschlägen folgt, und dem Benutzer besonders vielversprechende Seiten bereits auf der aktuellen Seite empfiehlt. Dieser könnte dann direkt auf diese Seite springen, und müßte nicht den Weg, den WebWatcher bereits gegangen ist, noch einmal gehen. Zusätzliche Informationen sind auf der WebWatcher Home Page[6] zu finden.

Literaturverzeichnis

[1] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. *1995 AAAI Spring Symposium on Information Gathering from*

[6] URL http://www.cs.cmu.edu/afs/cs/project/theo-6/web-agent/www/project-home.html

Heterogeneous, Distributed Environments, Stanford, March 1995. URL http://www.cs.cmu.edu/afs/cs/project/theo-6/web-agent/www/webagent-plus.ps.Z.

[2] T. Joachims, T. Mitchell, D. Freitag, and R. Armstrong. Webwatcher: Machine learning and hypertext. *Fachgruppentreffen Maschinelles Lernen, Dortmund*, August 1995. URL http://www.cs.cmu.edu/afs/cs/project/theo-6/web-agent/www/mltagung-e.ps.Z.

[3] J. Ross Quinlan. *Programs for Machine Learning.* Morgan Kaufmann Publishers, 1987.

[4] *The WebWatcher Tutorial.* URL http://webwatcher.learning.cs.cmu.edu:8080/html/tutorial.html.

Kapitel 10

Planen für das Sammeln von Informationen

Andreas Zimmermann

Übersicht

Das Internet hat sich zum größten Netzwerk der Welt entwickelt. Es bietet dem Benutzer eine große Vielfalt an Informationen aus jedem nur denkbaren Bereich. Diese Menge an Information ist auf Informationsquellen verfügbar, die über die ganze Welt verteilt sind. Ein Zugriff auf solche verteilten und uneinheitlichen Informationsquellen bringt Schwierigkeiten mit sich. Er soll nicht nur die gewünschten Informationen beschaffen, sondern dies auch möglichst effizient und flexibel erledigen. Aus diesem Grund ist es notwendig das Beschaffen von Information aus diesem riesigen Netzwerk genau zu planen. Hierfür wurde ein Planungssystem namens SAGE entwickelt, das hier vorgestellt werden soll. SAGE vereint Planung, Ausführung und Neuplanung einer Anfrage an ein Netzwerk, um dem Benutzer seine erwünschten Daten sicher und effizient zu beschaffen.

10.1 Einleitung

Ein riesiges Netzwerk, wie es das Internet ist, besteht aus unzähligen zusammengeschalteten Rechnern. Vom Supercomputer bis zum PC sind alle Arten von Computern in diesem Netz zugänglich. Und jeder dieser Rechner bietet dem Benutzer des Netzwerks seine ganz eigene Menge an Informationsmaterial zu jedem erdenklichen Thema.
Das bedeutet, daß zum Einen die Server, die die Informationen anbieten, nicht unbedingt einem einheitlichen Standart folgen, also zum Beispiel Unterschiede in den Datenübertragungs- und Rechengeschwindigkeiten bestehen. Zum Anderen liegen die gesuchten Daten in einem uneinheitlichen Format vor, weil sie zum Beispiel auf unterschiedlichen Arten von Datenbanken liegen. Ein Zugriff auf solche verteilten und heterogenen Informationsquellen bringt somit einige Probleme mit sich. So können beispielsweise Fehler auftreten, während eine solche Anfrage verarbeitet wird, die sofort behoben werden müssen.
Es können auch aus einer Anfrage heraus zusätzlich neue Aufgaben entstehen, die ebenfalls gelöst werden müssen. Solche Anfragen sind mitunter sehr komplex, bestehen also aus einer Vielzahl von auszuführenden Operationen. Aus diesem Grund ist es wichtig, die Vorgehensweise bei einer Anfrage genau zu planen. Mit Hilfe der Möglichkeiten der Künstlichen Intelligenz, von der die Planung ein Teilgebiet darstellt, wird nun ein nahezu optimaler Ablaufplan erstellt, der auszuführen ist.
Durch Planung erreicht man ein hohes Maß an Flexibilität und Effizienz:
Flexibel, weil Fehler durch Neuplanung behoben und neuentstandene Aufgaben ohne allzuviel Aufwand in den vorhandenen Plan eingefügt werden können. Effizient, weil zum Beispiel durch heuristische Suche ein Plan gefunden werden kann, der eine Anfrage in sehr kurzer Zeit abarbeiten kann.

Im Folgenden soll nun besprochen werden, wie das Planen für das Sammeln von Informationen aus verteilten und heterogenen Informationsquellen funktioniert. Des Weiteren wird ein Planungssystem namens SAGE vorgestellt, das für das Arbeiten in Umgebungen wie dem Internet konzepiert wurde. In SAGE sind vier Hauptaufgaben integriert: Planung, Ausführung, Neuplanung und Sensing.
Planung beinhaltet die Erstellung eines Planes, der parallele Ausführung berücksichtigt, um mehrere verteilte Ressourcen gleichzeitig anzusprechen, was in einem großen Netzwerk möglich ist. Die **Ausführung** und das Planen wurden in SAGE miteinander verbunden. Durch **Neuplanung** werden eventuell auftretende Fehler behandelt. **Sensing** bedeutet, daß der Planer während der Laufzeit zusätzliche Information über seine Arbeitsumgebung einholt und sofort verarbeitet.

Anhand dieses Planers soll gezeigt werden, wie das sichere, effiziente und flexible Beschaffen von Information verwirklicht wurde.

10.2 Grundsätzliches über das Planen

In diesem Abschnitt soll erklärt werden, wie sich Probleme mit Hilfe der Planung lösen lassen [1]. Dazu gehört auch, nahezu optimale Ablaufpläne für Prozesse zu erzeugen. Außerdem wird beschrieben, wie die Methoden der Künstlichen Intelligenz die Suche nach einem günstigen Plan unterstützen.

10.2.1 Lineares Planen

Um ein Problem mit einem Computer zu lösen, benötigt man einige Hilfsmittel. Einerseits braucht man natürlich gute Problemlösungsstrategien, aber auch die Frage, wie man ein Problem in einem Computer repräsentieren kann, muß beantwortet werden. Viele Probleme lassen sich in einzelne Teilprobleme aufgliedern, die meistens einfacher zu lösen sind. Lösungen dieser Unterprobleme müssen am Ende zu einer kompletten Gesamtlösung zusammengefügt werden (wobei zusätzliche Schwierigkeiten auftreten können).
Planung bedeutet die Benutzung von Methoden zur

1. Zerlegung eines Problems in Teilprobleme,
2. Bearbeitung dieser Unterprobleme und
3. Zusammensetzen der Teillösungen zur kompletten Problemlösung.

Ein einfaches Planungssystem enthält eine detaillierte Beschreibung des Problems als Anfangszustand, eine Beschreibung des Zielzustandes und erlaubte Hilfsmittel (Operatoren, Aktionen), um den Anfangszustand schrittweise in den Zielzustand zu überführen. Die Anfangssituation sowie der Endzustand werden durch eine Menge von Prädikaten ausreichend beschrieben. Dabei handelt es sich um einfache Aussagen, die mit *erfüllt* oder *unerfüllt* gekennzeichnet sind.
So ist zum Beispiel bei der Erstellung eines Planes zum Bewegen eines Tisches von A nach B das Prädikat „Tisch steht in Platz A“ im Anfangszustand erfüllt, während es im Endzustand nicht mehr erfüllt ist. Genau umgekehrt verhält sich das Prädikat „Tisch steht in Platz B“. Das Prädikat „Tisch hat vier Tischbeine und eine Tischplatte“ muß jedoch sowohl im Anfangs-, als auch im Endzustand erfüllt sein.
Will man nun den Operator „Bewege Tisch von A nach B“ auf den Ausgangszustand anwenden, muß zuerst gewährleistet sein, daß der Operator

überhaupt ausgeführt werden kann. Ist beispielsweise das Prädikat „Tisch ist festgenagelt" erfüllt, muß vorher noch ein Operator „Entferne Nägel" in den Ablaufplan eingefügt werden. Nach der Anwendung eines Operators liegt eine veränderte Situation vor, in deren Beschreibung einige Prädikate eine andere Belegung haben als vorher. Diese Situation muß noch nicht dem Zielzustand gleichen.
Die Bedingungen, die erfüllt sein müssen, um eine Aktion durchzuführen, heißen **Vorbedingungen**; die Prädikate, die nach der Durchführung geändert worden sind, nennt man **Effekte**.
Lineares Planen bedeutet nun, daß das System eine zusammenhängende **total geordnete** Kette von Operatoren finden muß, die sequentiell ausgeführt ein gegebenes Problem lösen. Wenn viele Operatoren zur Verfügung stehen, kann man sich vorstellen, daß es eine sehr große Anzahl Möglichkeiten gibt, einen solchen Ablaufplan zu konstruieren. Einige dieser Pläne sind gut, also sehr effizient und schnell in ihrer Ausführung, andere hingegen sind weniger geeignet. Mit den Verfahren der Künstlichen Intelligenz ist es allerdings möglich, sehr gute Pläne zu finden. Zwei dieser Verfahren *Best-First-Search* und *Means-Ends-Analysis* sollen nun vorgestellt werden.

Best-First-Search

Eine Möglichkeit, einen Plan zu erstellen, wäre, einfach eine zufällige Folge von Aktionen zu wählen und zu testen, ob diese Sequenz das Problem löst. Offensichtlich wird dieses Verfahren nur mit Glück einen guten Ablaufplan erzeugen, weil die Testfunktion nur „ja, das ist eine Lösung" oder „nein, das ist keine Lösung" zurückliefert, sich aber kein Urteil darüber erlaubt, ob die Lösung gut oder schlecht ist. Best-First-Search ist ein Verfahren, das problemspeziefisches Wissen in die Suche nach einem guten Plan miteinbringt. Ausgehend vom Anfangszustand werden zuerst alle Operatoren gesucht, die auf diesen Zustand angewendet werden können. Das geschieht durch Vergleich der Prädikate aus der Zustandsbeschreibung mit den *Vorbedingungen* der einzelnen Operatoren. Aus den *Effekte* der Operatoren entsteht eine Reihe neuer Zustände, die alle durch eine sogenannte **heuristische Funktion** bewertet werden. Diese Funktion setzt sich aus zwei Teilen zusammen, die zu einem Gesamtwert addiert werden:

- Einen Wert darüber, wie groß die Kosten für die Operatoranwendungen von Startpunkt bis zum aktuellen Zustand sind
- Eine Abschätzung der Kosten, die entstehen werden, wenn man vom aktuellen Zustand zum Ziel kommen würde.

Die heuristische Funktion liefert einen Zahlenwert zurück, der angibt, wie nahe ein Zustand am Endzustand liegt. Das Suchverfahren nimmt jetzt den

Zustand mit der besten Bewertung und sucht wieder alle Operatoren, die darauf anwendbar sind. So fährt Best-First fort, bis es einen Endzustand gefunden hat. Die Zustände, die keine Beachtung finden, werden nicht verworfen, sondern in einer Liste gespeichert, die nach der Höhe der Bewertung geordnet wird (somit wird diese Liste zu einer *Prioritätswarteschlange*). Läuft das Verfahren einmal in eine Sackgasse, kann dann mit dem Zustand weitergemacht werden, der an erster Stelle in dieser Liste steht.

Allgemeiner Algorithmus Best-First-Search:

1. Initialisierung

   OPEN: Liste der unbenutzten Zustände (Prioritätswarteschlange)

   CLOSED: Liste der benutzten Zustände

2. Berechne Anfangszustand und schreibe ihn in OPEN

3. Tue, bis Endzustand gefunden wurde oder keine Zustände mehr in OPEN stehen:

   (a) Nimm den besten Zustand von OPEN

   (b) Berechne alle Nachfolgezustände

   (c) Tue für jeden Nachfolgezustand:

      i. Wenn der Zustand noch nicht betrachtet wurde, bewerte ihn und füge ihn in OPEN ein.

      ii. Wenn der Zustand schon einmal betrachtet wurde, wechsle den Ursprungszustand, wenn der neue Pfad besser ist als der vorhergehende. In diesem Fall, aktualisiere die Kosten, um zu diesem Zustand und zu jedem seiner Nachfolgezustände zu gelangen.

Means-Ends-Analysis

Ist bei einem Problem der Anfangszustand und der angestrebte Endzustand explizit gegeben, kann man einen Algorithmus anwenden, der die Unterschiede zwischen diesen beiden Zuständen herausfindet und schrittweise durch Verkleinern dieser Unterschiede versucht, den Anfangszustand in den Endzustand zu überführen.
Das Prinzip dabei ist, zu allererst die Hauptprobleme zu lösen, und dann zu versuchen die kleineren Probleme zu lösen, die entstehen, wenn die Lösungen der Hauptprobleme zu einer Gesamtlösung zusammengesetzt werden. Wurde ein Unterschied festgestellt, muß nach einem Operator gesucht werden, der diesen Unterschied verkleinern kann. Beim Reduzieren eines Problems können zwei Arten von Unterproblemen entstehen:
Erstens kann der Operator vielleicht gar nicht direkt den gefundenen Unterschied verkleinern, sondern nur einen, der diesem ähnlich ist. Er muß demnach erst in diesen ähnlichen Unterschied, dem sogenannten Teilziel, transformiert werden, damit der zur Verfügung stehende Operator anwendbar ist. Es müssen also erst die *Vorbedingungen* des Operators erfüllt sein, damit er überhaupt angewendet werden kann.
Ein zweites Unterproblem kann entstehen, wenn der Operator den Anfangszustand durch Beseitigung des Hauptunterschiedes nicht direkt in den Endzustand umwandeln kann, was weitgehend immer der Fall ist. Im Allgemeinen sind diese entstehenden Unterprobleme leichter zu lösen, als das Hauptproblem.

Hier ist es also wichtig, mit dem Operator seine *Vorbedingungen* und *Effekte* explizit anzugeben. Durch das Means-Ends-Verfahren wird eine Kette von Operatoren gebildet, bei denen die *Effekte* des einen Operators und die aktuelle Zustandsbeschreibung die *Vorbedingungen* seines Nachfolgers erfüllen. Zusätzlich können den gefundenen Unterschieden Prioritäten zugewiesen werden, damit sich das System auch zuerst auf den wichtigsten Unterschied konzentriert.

<u>Allgemeiner Algorithmus Means-Ends-Analysis:</u>

Means-Ends(CURRENT, GOAL)

1. Vergleiche CURRENT mit GOAL.
   Wenn keine Unterschiede $\Longrightarrow$ ENDE

2. Nehme den wichtigsten Unterschied und reduziere ihn durch folgendes Verfahren solange, bis Erfolg oder ein Fehler eintritt:

   (a) Wähle einen bisher noch nicht benutzten Operator OP, der auf den jetzigen Unterschied angewendet werden kann.

Wenn kein Operator vorhanden ist $\Longrightarrow$ FEHLER

(b) Versuche OP auf CURRENT anzuwenden.
Erstelle die Beschreibungen von zwei Zuständen:

OP-START: Zustand, in dem die *Vorbedingungen* von OP erfüllt werden.

OP-RESULT: Zustand, der entstehen würde, wenn OP auf OP-START angewendet werden würde.

(c) Wenn (FIRSTPART $\Longleftarrow$ Means-Ends(CURRENT, OP-START) und (LASTPART $\Longleftarrow$ Means-Ends(OP-RESULT, GOAL) erfolgreich sind, signalisiere Erfolg und gebe das Resultat zurück, indem FIRSTPART, LASTPART und OP konkateniert werden.

Best-First-Search und Means-Ends-Analysis stellen zwei verschiedene Prinzipien der heuristischen Suche dar. Best-First-Search versucht, wenn möglich immer den besten Plan zu finden, arbeitet dafür ziemlich langsam. Means-Ends-Analysis hingegen wird, weil es rekursiv abläuft, sehr schnell einen Plan finden; allerdings ist dieser Plan nicht immer sehr effizient.

10.2.2 Nichtlineares Planen

In manchen Problemgebieten ist es möglich, mehrere Aktionen gleichzeitig auszuführen. Das geschieht zum Beispiel, wenn für eine Aufgabe mehr als ein Roboterarm zur Verfügung steht, so daß mehrere Teile zur selben Zeit umherbewegt werden können. Auch in der Problemumgebung der Informationsbeschaffung von verteilten Systemen können mehrere Anfragen gleichzeitig an verschiedene Systeme gerichtet werden.

Das Erstellen eines Ablaufplanes, der die mögliche parallele Ausführung von Operatoren berücksichtigt und die dadurch entstehenden Schwierigkeiten handhabt, ist die Aufgabe eines nichtlinearen Planers. Dabei werden mehrere Teilziele, die an einem Punkt im Planungsprozeß auftreten können, unabhängig voneinander bearbeitet und zwar an der Stelle im Planungsprozeß, wo sie aufgetreten sind. Es entsteht ein Plan mit **partiell geordneten** Operatoren.

Durch die getrennte Ausführung der Operatoren in verschiedenen Abschnitten des Planes, kommt es vor, daß Konflikte unter manchen Operationen entstehen. Als Konflikte bezeichnet man unerwünschte Interaktionen zwischen einzelnen Teilplänen oder Operatoren, die die reibungslose Ausführung des Planes gefährden. Solche Interaktionsprobleme treten dann auf, wenn der *Effekt* des einen Operators die *Vorbedingungen* des anderen im Parallelzweig geplanten Operators ändert, so daß dieser nicht mehr ausgeführt werden kann.

Um diese Konflikte zu lösen, führt man entweder eine zeitliche Reihenfolge unter den gefährdeten Operatoren ein (Linearisierung des Planabschnittes), oder man fügt einen Operator dazwischen ein, der die *Vorbedingungen* des zweiten Operators wiederherstellt.

Man sieht also, daß man durch lineares Planen sehr sichere Pläne erhält, die ganz frei von Konflikten sind, während nichtlineare Pläne wesentlich schneller ausgeführt werden können, da mögliche Parallelität ausgenutzt wird.

10.3 Der Planer SAGE

Ein System, das Planung durchführt, wie sie im vorangegangenen Abschnitt beschrieben wurde, ist der Planer SAGE [3]. Er wurde entwickelt, um speziell in der Umgebung eines großen Netzwerks zurecht zu kommen. Die Grundlage für dieses Planungssytem bildet ein nichtlinearer Planer namens UCPOP. SAGE plant Zugriffsanfragen auf verteilte und uneinheitliche Informationsquellen (Ressourcen) und führt sie aus [2] [4] .

10.3.1 Planen für das Sammeln von Informationen

Um Informationen aus einem Netzwerk zu beschaffen, muß eine Anfrage drei Hauptaufgaben erfüllen:

1. Auffinden (locate) der benötigten Daten auf den verschiedenen Informationsquellen,
2. Übertragen (retrieve) dieser Daten auf das lokale System und
3. Zusammenfügen (join) der Daten zum Ergebnis der Anfrage.

Für diese drei Aufgaben gilt es nun einen Plan zu erstellen, der die einzelnen Schritte einer Zugriffsanfrage effizient gestaltet. Im Startzustand, bei dem mit der Planung begonnen wird, ist beschrieben, welche Informationsserver zur Zeit verfügbar sind und welcher dieser Server welche Informationsquellen anbietet. Der Endzustand legt genau fest, welchen Bedingungen die angeforderten Daten zu genügen haben und wohin das Ergebnis geschickt werden soll (z.B. Output).
Um vom Startzustand in den Endzustand zu gelangen, werden Operatoren gebraucht. SAGE verwendet für die Erstellung eines Planes drei verschiedene Arten von Operatoren:

- Operatoren zum Auswählen einer Quelle, von der die Daten geholt werden sollen,
- Operatoren, die die Datenverarbeitung steuern und
- Operatoren, die festlegen, wo die Operatoren ausgeführt werden und in welcher Reihenfolge.

Der Unterschied zu einem normalen Planungsprozeß, wie er oben beschrieben wurde, und dem Planen einer Anfrage besteht darin, daß die Anfragen zur Realzeit beantwortet, und somit auch die Pläne zur Realzeit erstellt werden müssen. Außerdem soll nicht nur irgendeine Lösung für ein Problem gefunden werden, sondern auch noch eine möglichst gute. Das prinzipielle Konzept von SAGE, nämlich Planung, Ausführung, Neuplanung und Sensing soll nun im Folgenden näher betrachtet werden.

10.3.2 Planung in SAGE

Das grundlegende Ziel eines Planers, der die Möglichkeit hat, auf verteilte Ressourcen zuzugreifen, sollte sein, diese auch getrennt voneinander gleichzeitig anzusprechen. Dieser Vorteil der parallelen Verarbeitung bringt eine sehr große Zeiteinsparung mit sich. Für die Planung bedeutet das, Unterpläne zu erzeugen, die Unteranfragen an die verschiedenen Server verwalten. Dabei können zwei Probleme auftreten:
Erstens darf bei einer parallelen Ausführung kein Operator das Ergebnis eines anderen beeinflussen, zweitens dürfen zwei Operatoren nicht die gleiche Informationsquelle zur selben Zeit benutzen (Ressourcenkonflikt).
Zur Lösung dieses Problems wurde den in SAGE benutzen Operatoren eine Ressourcendeklaration hinzugefügt, so daß sie folgendes Schema haben:

1. Name des Operators
2. Benutzte Variable
3. Benutzte Ressourcen
4. Vorbedingungen
5. Effekte

Mit jedem Neueintrag eines Operators in den bestehenden Plan wird dann geprüft, ob Ressourcenkonflikte mit anderen Operatoren des Planes entstehen, und diese werden danach durch Umordnung der Aktionsreihenfolge aufgelöst.

Daten können von verschiedenen Orten übertragen und an verschiedenen Orten verarbeitet werden; die Operatoren eines Plans können in verschiedener Reihenfolge ausgeführt werden. Das bedeutet, daß es unzählige Möglichkeiten gibt, einen Zugriffsplan zu erstellen. Es entsteht ein entsprechend großer Suchraum, der nach einem guten Plan durchforstet werden muß.
Um den Suchraum zu verkleinern und die Suche zu vereinfachen, müssen geeignete Funktionen, die die Kosten für die Ausführung einer Zugriffsanfrage abschätzen, und Heuristiken eingesetzt werden. Sie sollen nicht nur den zu durchsuchenden Raum von möglichen Plänen einengen, sondern auch dabei helfen, Teilpläne untereinander vergleichbar zu machen, um den passendsten zu finden. Folgende Werte sollten in eine Kostenabschätzungsfunktion eingebracht werden:

- Kosten für das Zugreifen auf die verschiedenen Informationsquellen (Zugriffszeit auf die Datenbank)
- Kosten für die Übertragung von Suchergebnissen (Übertragungsgeschwindigkeit der einzelnen Netzverbindungen)
- Kosten für das Kombinieren verschiedener Daten (JOIN-Operation)
- Kosten für das Ausnutzen möglicher Parallelität (Erstellen von Unteranfragen)

Heuristik und Bewertungsfunktion dürfen nur soviel Einfluß auf die Suche haben wie nötig. Sie sollen möglichst viele Pläne schon im voraus eliminieren, so daß die Suche effizient wird, aber nicht zu viele, da sonst möglicherweise sehr gute Pläne verloren gehen könnten.

Jetzt kann mit einem geeigneten Suchverfahren (wie oben beschrieben) damit begonnen werden, einen Plan mit hoher Qualität zu finden.

10.3.3 Ausführung in SAGE

In SAGE ist die Ausführung eines Planes Teil der Planung. Ziel ist es, einen kompletten und ausgeführten Plan zu erstellen. Dazu wird in SAGE jede einzelnen Aktion mit einer Marke versehen, die anzeigen soll, ob sie unausgeführt (**unexecuted**) ist, sich in Ausführung befindet (**executing**) oder ob sie bereits ausgeführt wurde (**executed**). Dabei gelangt eine Operation erst in den Status *unexecuted*, wenn ihre *Vorbedingungen* alle erfüllt und alle möglichen Konflikte beseitigt sind. Ist eine Operation ausführbar, wird sie als Prozeß gestartet, der im Hintergrund abläuft, und bekommt die Marke *executing* zugewiesen. Durch diese Marke weiß der Planer, daß diese Operation weder abgebrochen, noch abgeändert werden darf. Wurde die Aktion erfolgreich beendet, bekommt der Planer eine Meldung und ihre Marke

ändert sich in *executed*.
Das Planungssystem sucht in regelmäßigen Abständen diese Marken bei ausgeführten Operationen, um den Plan in Bezug auf die Ausgabe auf den neusten Stand zu bringen. Die durch das Beenden einer Aktion neuentstandenen Bedingungen werden einer Liste zugefügt, die alle noch offenen und zu erfüllenden Bedingungen enthält. Jetzt kann der Planer Aktionen ausführen, für die die vorangegangenen Operationen schon erfolgreich abgearbeitet wurden.

Das Abarbeiten der Aktionen im Hintergrund bewirkt, daß der Planer durchgehend laufen und arbeiten kann. Sobald ein Operator beendet wurde oder ein Fehler aufgetreten ist, kann der Plan bei Bedarf abgeändert und aktualisiert werden. Die Planung erfolgt also im Kontext des gerade laufenden Plans. Dadurch wird der ganze Prozeß der Planung sehr flexibel und so können die Fehlerbehandlung, die Handhabung neuentstandener Aufgaben und das Sensing einfach durchgeführt werden, wie in den folgenden Abschnitten gezeigt werden wird.

10.3.4 Neuplanung nach Auftreten eines Fehlers

Wie schon erwähnt, kann die Ausführung eines Operators neue Aufgaben oder Ziele mit sich bringen, die in der weiteren Planung berücksichtigt werden müssen, um zu einem erfolgreichen Abschluß des Planes zu kommen. Solche Aufgaben oder Ziele können zum Beispiel Fehler sein, die zum Abbruch einer Aktion geführt haben, oder einfach nur neue Ereignisse, die durch das erfolgreiche Abschließen einer Operation zusätzlich entstanden sind.
Nun will man natürlich auf Grund eines Fehlers nicht den gesamten Plan verwerfen und noch einmal einen neuen entwerfen, sondern nur soviel wie nötig am bestehenden Plan ändern. Das bedeutet, daß nur der Teilplan neu geplant werden muß, in dem der Fehler aufgetreten ist, und die Teilpläne, die durch diesen Fehler beeinflußt wurden.
Das wird erreicht, indem der Planer den Suchraum nach einem modifizierten Plan durchsucht, der den fehlgeschlagenen Teilplan nicht mehr enthält. Der Plan wird dann auf einen Zustand zurückgesetzt, der vor dem fehlgeschlagenen Teilplan liegt, und der neugeplante Teil gestartet. Da in der Umgebung der Informationsbeschaffung mehrere mit der gleichen Information existieren, heißt Neuplanung hier, daß die fehlerhafte Teilanfrage noch einmal gestellt wird, nur an einen anderen Server gerichtet. In dieser Umgebung ergeben sich drei Arten von Fehlern:

- Der Server oder die Datenbank ist zur Zeit nicht verfügbar
- Die Anfrage wurde falsch formuliert

- Falsche Daten werden zurückgesendet

Die ersten beiden Fehlerarten sind wie gezeigt relativ einfach durch Umformulierung der Anfrage zu beheben. Die dritte Art setzt voraus, daß genaue Vorstellungen darüber existieren, wie die angeforderten Daten auszusehen haben. Dann ist es möglich Ergebnis und Vorstellung zu vergleichen und Konsequenzen daraus zu ziehen. Der große Vorteil von SAGE ist es, die gesamte Neuplanung während der Ausführung durchzuführen. So können Aktionen, die sich in der Ausführung befinden, weiterlaufen, während in einem anderen Teil des Planes neugeplant wird.

10.3.5 Sensing in SAGE

Bei der großen Vielfalt an Information, die ein großes Netzwerk bietet, besteht die Möglichkeit, bestimmte Informationen für den weiteren Ablauf des Planes zu gebrauchen. Sensing bedeutet, daß Teile des Resultates einer Anfrage gespeichert werden, um sie an anderer Stelle im Plan wieder zu verwenden. Dazu werden den Operatoren Laufzeitvariabeln hinzugefügt, die das Ergebnis einer Anfrage zwischenspeichern sollen. Diese Variablen werden in den *Effekte* der Aktionen aufgeführt. Mit diesen gespeicherten Daten aus einer Informationsquelle kann nun

- eine Anfrage an eine andere Informationsquelle formuliert werden
- eine neue geeignetere Informationsquelle ausgewählt werden.

Eine Konsequenz hieraus ist allerdings, daß solange mit der weiteren Abarbeitung des Planes gewartet werden muß, bis die benötigte Information zur Verfügung steht. Die Fähigkeit, mit dem Wissen aus einer bereits ausgeführten Anfrage eine neue Anfrage zu formulieren, wurde in SAGE durch Erstellen zweier zusätzlicher Operatoren verwirklicht.
Der erste Operator, **execute-query**, holt sich einfach nur Daten vom lokalen System und bindet sie an Variabeln. Die einzige *Vorbedingung* dieser Aktion ist, daß die Daten auch auf dem lokalen System erhältlich sein müssen.
Use-gathered-info, der zweite Operator, prüft, ob eine Anfrage so zerlegt werden kann, so daß Datenteile in einer weiteren Anfrage Verwendung finden. Die Anfrage wird dann abgeändert und eine Unteranfrage abgesplittet. Um das Ergebnis der Unteranfrage benutzen zu können, muß sie zuerst ausgeführt werden. Durch die Laufzeitanbindung der Variabeln ist das Ergebnis sofort in die modifizierte Anfrage eingesetzt, so daß diese gleich abgeschickt werden kann.
Sensing hilft die Datenmengen, die unter den Systemen hin- und hergeschoben werden, zu reduzieren. Dadurch wird ebenfalls die Zugriffszeit verkürzt, was zusätzliche Effizienz bringt.

10.4 Schlußwort

Wie man gesehen hat, werden durch das Planungssystem SAGE alle Fähigkeiten und Vorteile eines Planers für das Sammeln von Informationen benutzt.
Einerseits kann im Rahmen der Planung heuristische Information mit eingebracht werden, um die Anfragen möglichst effizient zu gestalten (*Lernen* als Teilgebiet der Künstlichen Intelligenz). Außerdem ermöglicht das Prinzip der Planung das Integrieren von Planung und Ausführung, wodurch drei Vorteile entstehen:
Erstens können Ausführungsfehler besser behandelt werden. Zweitens kann man so besser mit unvollständiger oder inkonsistenter Information zurechtkommen. Drittens bringt das Zusammenfügen von Planung und Ausführung sehr viel mehr Flexibilität mit sich.
Die explizite Auflistung der benutzten Ressourcen in einem Operator ist äußerst wichtig, um Ressourcenkonflikte aufzuspüren und somit einen Plan weitgehend parallel auszuführen. Diese Neuerung ergibt eine zusätzliche Zeitersparnis.
Die Möglichkeit der Planung, *Vorbedingungen* als Prädikate zu definieren, hilft bei der Suche nach den passenden Operatoren, um sie in einen bereits vorhandenen Plan einzufügen. Außerdem kann dies bei der Aufteilung einer Anfrage in mögliche Teile nützlich sein.

Literaturverzeichnis

[1] Joachim Hertzberg. *Planen.* Mannheim; Wien; Zürich: BI-Wissenschaftsverlag, 1989.

[2] J. Koch M. Jarke. Query optimazation in database systems. In *ACM Computing Surveys*, volume 16 of *2*, pages 111–152, 1984.

[3] University of Southern California. *Integrating Planning and Execution for Information Gathering*, Marina del Rey, CA 90292. Craig A. Knoblock.

[4] C.-N. Hsu C. A. Knoblock Y. Arens, C.Y.Chee. Retrieving and integrating data from multiple information sources. In *International Journal on Intelligent and Cooperative Information Systems*, volume 2, pages 127–158.

Kapitel 11

MACRON: Kooperatives Sammeln von Informationen

Matthias Lang

Übersicht

Die Komplexität der angebotenen Informationen in unserer Welt erfordert schon jetzt und mehr noch in der Zukunft ein zielgerichtetes und systematisches Vorgehen beim Gewinnen von Informationen.

Dieser Teil des Seminars basiert auf dem Cooperative Information Gathering (CIG) Paradigma, welches speziell für komplexe Informations-Sammel-Systeme konzipiert wurde. Useranfragen bewirken, daß zum Teil vorgegebene Ablaufpläne komplettiert werden und anhand derer in der Regel mehrere verschiedene Agenten in Aktion treten, um die (Teil-)Aufgaben anzugehen. Die Agenten kooperieren dabei auf vielfache Art.

MACRON ist eine konkrete Instanz dieses Ansatzes, befindet sich aber noch im Anfangsstadium seiner Entwicklung. MACRON versucht die Möglichkeiten der Agenten sinnvoll zu kombinieren und Wechselbeziehungen zu berücksichtigen. Zudem werden bei der Erzeugung der Suchpläne Rahmenbedingungen wie Resourcenbeschränkungen oder Kostenfragen berücksichtigt. MACRON stellt eine Komplettarchitektur für solche Suchprobleme dar.

Dieses Dokument basiert in wesentlichen Teilen auf einer Veröffentlichung von Decker, Lesser und anderen [8].

11.1 Einleitung

Die Komplexität der angebotenen Informationen in unserer Welt erfordert ein zielgerichtetes und systematisches Vorgehen beim Gewinnen von Informationen. Dabei geht es mehr um das eigentliche Gewinnen, sprich Erzeugen, von Antworten, als um das mit Information Retrieval bezeichnete Gebiet der Abrufung von einmal abgelegtem Wissen.
Grundlage dieses Kapitels ist das Cooperative Information Gathering (CIG) Paradigma [12], welches speziell entwickelt für Informations-Sammel-Systeme konzipiert wurde. Useranfragen bewirken, daß zum Teil vorgegebene Rahmen für Ablaufpläne komplettiert werden, Aufgaben in Teilaufgaben zerlegt werden und anhand derer Agenten mit speziellen Fähigkeiten in Aktion treten. Diese Aufgabenverteilung verlangt nach einer Organisation, welche die verschiedenen Aktivitäten koordiniert. Außerdem müssen die Teilantworten zu einer gemeinsamen Antwort auf die Useranfrage weiterverarbeitet werden. Das umfaßt das Konkatenieren, Abwägen und Auswerten der Teilergebnisse und schließlich auch die Erzeugung und Darstellung der (Teil-)Antworten und der Gesamtantwort für den Anwender.

Was sind überhaupt Agenten?
Agenten sind im wesentlichen Programme. Sie können auch mit wissensbasierten Methoden arbeiten, eine bestimmte Aufgabe erfüllen und auch mit anderen Agenten kommunizieren [5].

Eine konkrete Instanz dieses Ansatzes ist MACRON (Multi-agent Architecture for Cooperative Retrieval ONline). Der Multiagenten-Ansatz bietet die Möglichkeit, mit spezialisierten Agenten in heterogenen Umgebungen zu agieren. Ferner können die Agenten mit wissensbasierten Methoden arbeiten und somit *lernen*. Als Quellen kommen Newsgruppen, WWW-Seiten, Datenbanken und Archive von Verlagen oder Sonstiges in Betracht, prinzipiell spielt die Art der Quelle keine Rolle. Ein Vorteil des Mutliagenten-Ansatzes ist die hohe Flexibilität. Kommen neue Arten von Quellen hinzu, werden einfach nur neue spezialisierte Agenten in die Architektur eingebunden, während der organisatorische Überbau erhalten bleibt. Außerdem können Suchprozesse dynamisch sein, d.h. Userinteraktionen zur Laufzeit lassen davon nicht betroffene Agenten weiterarbeiten und den Anwender trotzdem den Prozeß lenken. Aber auch Agenten selbst können Suchpläne zur Laufzeit beeinflussen, z.B. wenn Zeitlimits überschritten werden. Solche Anforderungen an Flexibilität, Dynamik und Lernfähigkeit verbunden mit dem Beachten von Resourcenbeschränkungen und Qualitätsanforderungen bedarf eines komplexen Managements. Die aus der Hauptanfrage generierten Teilanfragen oder Aufgaben sind zudem in der Regel nicht voneinander unabhängig, d.h. (Zwischen-)Resultate von Agenten wirken sich auf andere

Agenten aus. Die Agenten stehen in ständiger Wechselwirkung zueinander bzw. kooperieren bei ihrer Suche. Teilantworten können das weitere Vorgehen von anderen Agenten beeinflussen. Zudem können die Agenten die Wahl ihrer lokalen Suchpläne von den Strategien oder Quellen anderer aktiver Agenten abhängig machen. Das Nutzen gemeinsamer Quellen kann auch dazu führen, daß Reihenfolgen der Aktivitäten unter den Agenten abgesprochen werden. Durch schrittweise Verfeinerung der globalen Anfrage gelangt man zu verschieden abstrakten Sichten auf das Problem, was sich auch in einer Agentenhierarchie [Abb. 11.1] widerspiegelt.

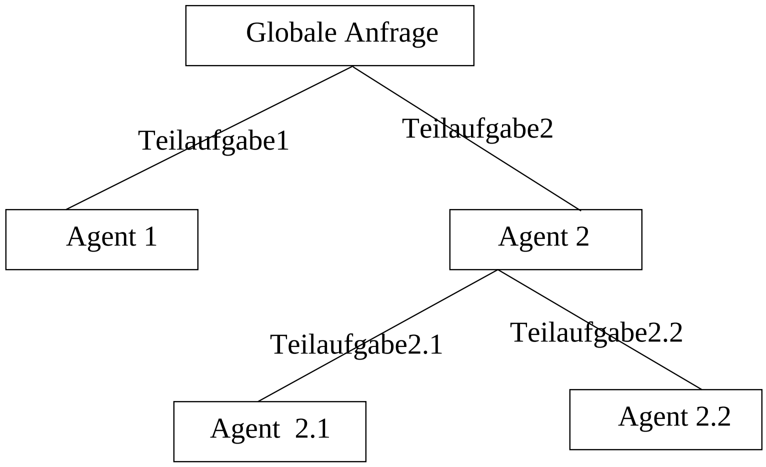


Abbildung 11.1: Hierarchie von Agenten

Das Prinzip der schrittweisen Verfeinerung ist ganz natürlich. Sucht man selbst nach Informationsmaterial zu einem bestimmten Thema, z.B. MACRON, so könnte man sich folgende Teilaufgaben vorstellen.

- Suchen in Zeitschriften und Journalen
- Suchen in der Bibliothek unter diesem Stichwort
- Suchen mit externen Suchmaschinen im WWW, wie Alta Vista oder Lycos.

Diese drei Tätigkeiten sind relativ autonom durchzuführen. Man selbst würde natürlich eines nach dem anderen machen, also sequentiell vorgehen, man könnte diese Teilaufgaben aber auch an andere Leute (Agenten) weiterdelegieren, also parallel durchführen.

Vorteile der Multiagenten-Suche

- Agenten mit konkurrierenden Strategien bei gleichen Teilaufgaben können die Qualität einer Antwort verbessern.
- Mutliagenten-Architekturen unterstützen paralleles Abarbeiten von Subproblemen und somit Effizienzsteigerung.
- Agenten können in Netzwerken vorort an der Quelle die Daten untersuchen, man nennt dies eine verteilte Problemlösungsstrategie. So werden nur die relevanten Informationen an den Ausgangsrechner transferiert, statt unnötig viele Daten an eine zentrale Verarbeitungseinheit zu kopieren.
- Sie bieten Modularität, Robustheit, Verfeinerung abstrakter Anfragen, leichtere dynamische Anpassung an neue Umgebungscharakteristika und andere Vorteile von verteilten Systemen wie bessere Ausnutzung von Rechenleistung.
- Ferner könnten Teile des Such-Systems ersetzt oder gewartet werden, ohne daß andere Komponenten davon betroffen wären.
- Agenten können transparent für andere, höher angesiedelte Agenten und schließlich auch für den Anfrager arbeiten und damit verschieden abstrakte Sichtweisen unterstützen.
- Passive Quellen wie Datenbanken können sinnvoll genutzt werden, wenn man nur eine intelligente Schnittstelle dazu bereitstellt [2]. Dies ist dann ein hochspezialisierter Agent.

11.2 MACRON

MACRON setzt das CIG-Konzept um. Seine Architektur nutzt die Fähigkeiten von spezialisierten Agenten, kontrolliert die Wechselwirkungen zwischen den Teilaufgaben, behebt Inkonsistenzen zwischen den Teilresultaten, bearbeitet oder vermeidet Redundanz und nutzt die Resourcen intelligent. Die Agenten arbeiten weitgehend autonom, haben aber einen festen Rahmen, in den sie eingebunden sind.

11.2.1 Architektur und Organisation

Jede Multiagenten-Architektur, also auch MACRON, ist im wesentlichen einfach eine Zuordnung von Aufgaben an bestimmte Agenten. Diese Rollenzuweisung weist jedem Agenten sein Aufgabengebiet, seine Resourcen und die Art und Weise der Darstellung und Übermittlung seiner Ergebnisse zu.

Aus organisatorischer Sicht gibt es zwei Mengen von Agenten. Es sind dies Funktionale Einheiten und Anfragen bearbeitende Einheiten. Funktionale Einheiten sind Mengen von Agenten mit gleichem Tätigkeitsbereich, sie haben z.B. gleiche Typen von Informationsquellen (Datenbanken, Newsgruppen). Für die Dauer einer Suchanfrage werden aus den benötigten Funktionalen Einheiten die einzelnen Agenten rekrutiert, wobei die Auswahl wiederum nach bestimmten Kriterien erfolgt wie *zur Verfügung stehende Zeit* oder *erwartete Qualität* der Antwort. Die Auswahl trifft ein bestimmter Agent jeder Funktionalen Einheit, nämlich der Funktionale Manager. Er kennt die Besonderheiten und speziellen Fähigkeiten aller Mitglieder seiner Gruppe und kann so entscheiden, wann er welche seiner Agenten den Anfrage bearbeitenden Agenten zur Verfügung stellt.
Auf Menschen übertragen wären Programmierer und Hardware-Designer in je einer Funktionalen Klasse und bei neu anstehenden Projekten würden aus beiden Gruppen Leute ausgewählt, die, entweder aufgrund ihrer speziellen Erfahrung oder weil sie gerade nicht beschäftigt sind oder sehr schnell arbeiten, besonders geeignet sind. Werden neue Agenten in das Suchsystem integriert, genügt es dies dem Funktionalen Manager bekannt zu machen, d.h. diese Organisationsform erlaubt es, Agenten auszutauschen, zu ergänzen oder zu streichen ohne den organisatorischen Überbau zu verändern. Bei neuen Projekten bzw. Anfragen werden dann Projektteams gebildet. Sie bestehen im Beispiel aus Hardware- und Softwaredesignern sowie einem Projektleiter. In diesem Kontext sind es die Anfragen bearbeitenden Einheiten. Sie bestehen aus einem Anfrage-Manager Agenten und einer Menge von Agenten, die aus den Funktionalen Gruppen nach oben erläutertem Verfahren rekrutiert werden. Der Manageragent der Suchanfrage-Einheit erstellt einen abstrakten globalen Suchplan, global in dem Sinn, daß er alle Aspekte der Useranfrage abdeckt und von den benötigten Funktionalen Managern Agenten anfordert. Zudem überwacht er den Ablauf des Plans zur Laufzeit. Das schließt ein, daß der Anwender auf dem aktuellen Stand gehalten wird und auch die Möglichkeit erhält, den Prozeß zu beeinflussen. Bei MACRON wird die Schnittstelle zum User durch einen Web-Browser mit Eingabefeldern realisiert. Wenn soviele Agenten aktiv sind und zumindest manche wie der Funktionale Manager und der Anfrage-Manager miteinander kommunizieren, muß es einen einheitlichen Standard über den gegenseitigen Austausch geben. Bei MACRON kommunizieren die Agenten in der Sprache KQML (Knowledge Query and Manipulation Language). Die Sprache basiert auf dem ARPA-Standard. ARPA-Rechnernetze wurden ursprünglich in den USA vom Verteidigungsministerium betrieben. Näheres zu KQML kann in der Literatur nachgelesen werden [3]. Außerdem wird eine gemeinsame Representation der Anweisungen benutzt, die auf TAEMS (Task Analysis, Environment Modeling, and Simulation) basiert. TAEMS bietet die Möglichkeit, Abhängigkeiten zwischen Aufgaben zu spezifizieren und diese so beim Planen und auch zur Laufzeit zu berücksichtigen [11]. Wenn Agenten mitein-

ander kommunizieren sollen, dann müssen sie auch einen Überblick haben, welche Agenten es gibt und welche Adresse sie haben. Diese Information ist in dem OCM (Organizational Chart Manager) abgelegt. Der OCM ist einfach ein Speicher in dem über jeden Agenten Statusinformationen abgelegt sind. Neue Agenten oder gar neue Funktionale Einheiten können zu jeder Zeit in ein laufendes System eingebunden werden, indem sie einfach im OCM eingetragen werden.

Beispiel

Ein Beispiel für einen möglichen Ablauf einer Suchanfrage wäre *Suche Informationen über MACRON*. Diese globale Useranfrage wird vom Anfrage bearbeitenden Agenten verfeinert zu *Suche mit Web-Search-Engines*, *Suche in der Bibliothek nach Büchern* und zu *Suche in Zeitschriften und Journalen*. Für diese Teilaufgaben gibt es jeweils Funktionale Einheiten und somit Funktionale Manager, mit denen der Anfragemanager in Kontakt treten muß. Die Funktionalen Manager zerlegen die Aufgaben weiter bis hin zu elementaren Aufgaben und rekrutieren jeweils die passenden Agenten ihrer Gruppe. Dabei kann es wiederum zur Bildung von Hierarchien kommen, d.h. die Aufgabe wird auf verschiedenen Abstraktionsebenen bearbeitet. Elementare Aufgaben wären z.B. *benutze FTP um ein File hierher zu kopieren* oder *kopiere die Einträge x,y aus der Datenbank z.*

11.2.2 Alternative Organisation der Agenten

Der MACRON-Ansatz gruppiert also seine Agenten nach funktionalen Gesichtspunkten, ausgenommen die Anfrage bearbeitenden Agenten. Für jede neue Suchanfrage werden neue Anfrage bearbeitende Einheiten zusammengestellt. Dabei wird berücksichtigt, welche Agenten zur Zeit frei sind und welche individuellen Eigenschaften sie haben. Solche Projektteams wie man sie bei Menschen nennen würde, sind flexibel an die jeweilige Anforderung anpassbar, dynamisch auch zur Laufzeit änderbar (z.B. beim Überschreiten von Zeitlimits werden schnellere Agenten aktiviert), und die zur Verfügung stehenden Resourcen können intelligent genutzt werden.

Statische Anfrage bearbeitende Einheiten

Alternativ könnte man auch von vornherein feste Anfrage bearbeitende Einheiten, also feste Projektteams bilden, um so die organisatorische Arbeit zu reduzieren. In jedem festen Team müssten dann Agenten von jeder möglichen Funktionalität vorkommen, damit auch jede Anfrage bearbeitet werden kann. Dies führt aber zu Nachteilen.

- In aller Regel wird nicht jede Anfrage jede Funktionalität erfordern, d.h. es werden Agenten gebunden, die eigentlich nichts tun haben.
- Durch das Fehlen eines Agenten ist das ganze Team blockiert. Es könnte sein, daß ein Agent aus Wartungsgründen oder Sonstigem temporär nicht erreichbar ist, dann kann die Suchanfrage nicht bearbeitet werden, obwohl es vieleicht Agenten mit der selben Funktionalität gibt.
- Setzt man wissensbasierte Methoden ein, also lernfähige Agenten, dann werden sich die Agenten im Laufe der Zeit unterscheiden, auch wenn sie ursprünglich gleiche Funktionalitäten hatten. Dies ist insofern ein Problem, als daß es das Ziel sein muß, qualitativ die besten Antworten auf eine Anfrage zu liefern. Diese ist aber mit festen Teams in der Regel nicht erreichbar, sondern nur durch Benutzen der jeweils besten Agenten für ein Problem, und diese sind i.a. nicht in der selben Anfrage bearbeitenden Einheit.

Agenten-Pool

Um die Nachteile des statischen Ansatzes zu beheben, kann man auch das andere Extrem betrachten. Alle Agenten sind in einem großen Pool, und Anfrage bearbeitende Agenten wählen bei jeder neuen Anfrage ein Team aus. Dieser Ansatz ist flexibel, dynamisch und chaotisch. Wenn ein Anfrage bearbeitender Agent ein möglichst gutes Team zusammenstellen will, muß er Information über jeden einzelnen Agenten haben, über seine speziellen Fertigkeiten, seine bereits gewonnenen Ehrfahrungen und sein Bedarf an Resourcen. Da dies bei großen Suchsystem schwer zu verwalten scheint, kommt nur in Frage, daß die Agenten selbst Statusinformationen bereithalten, die der Manager abrufen kann, wenn er ein Team organisiert. Das setzt aber voraus, daß die Agenten sich selbst untereinander vergleichen können, also brauchen sie Informationen über alle anderen Agenten, und wir haben wiederum das selbe Problem.
Bei MACRON verwaltet der Funktionale Manager die Statusinformationen über einen Teil der funktionalen Agenten, nämlich die in seiner Gruppe. So kann er entscheiden, wann welcher Agent rekrutiert wird.

11.3 Agentenklassen bei MACRON

MACRON benutzt drei Klassen von Agenten. Es sind die DECAF-Agenten, primitive Datensammelagenten und solche für die Benutzerschnittstelle.

11.3.1 DECAF-Agenten

Die Architektur der DECAF-Agenten (Distributed Environment Centered Agent Framework) [Abb. 11.2] ist die selbe wie die, die benutzt wurde, um das GPGP (Generalized Partial Global Planning) zu testen [9, 7, 6]. DECAF-Agenten bestehen aus mehreren Komponenten. Eine verfügt über das organisatorische Wissen, also über die aktuelle hierarchische Aufgaben-Struktur. Ferner gibt es Komponenten, die Planen, Entscheidungen treffen und die Abläufe überwachen. Eine andere ist für die Kommunikation mit anderen Agenten zuständig.

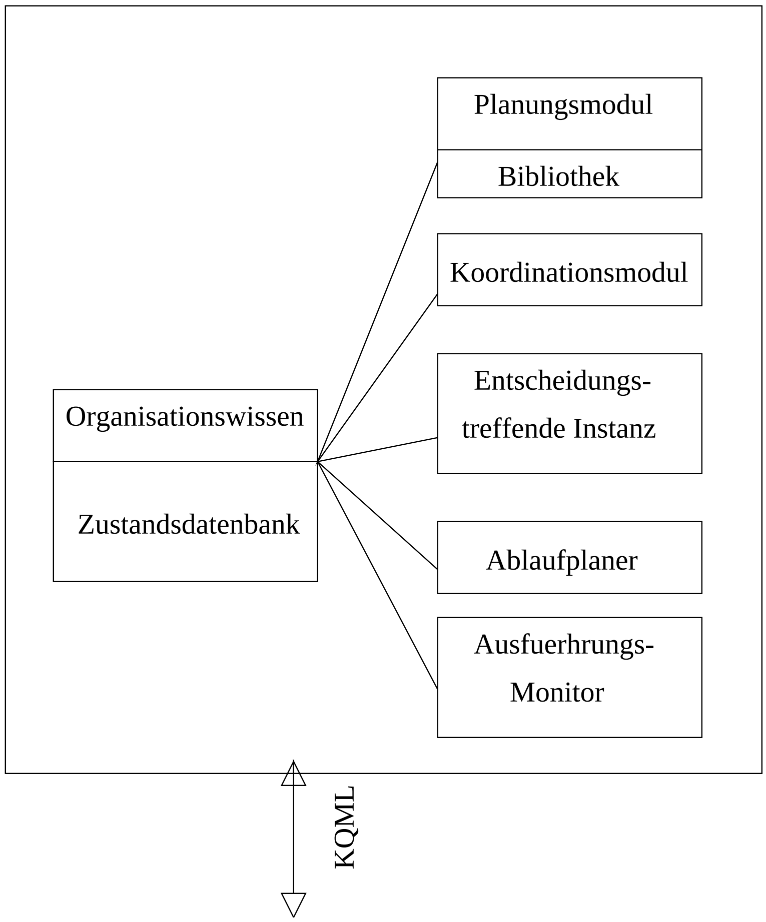


Abbildung 11.2: DECAF-Agenten Architektur

Zustandsdatenbank

In der Zustandsdatenbank werden alle aktuellen Zustände der zu erledigenden Aufgaben und laufenden Prozesse abgelegt. Außerdem sind hier auch die Beziehungen der Teilaufgaben untereinander und zu denen anderer Agenten vermerkt. Dazu müssen natürlich auch Daten über die Organisation einer aktuellen Suchanfrage vorhanden sein. Sie sind als Untersektion ebenfalls in der Zustandsdatenbank. Die Zustandsdatenbank kann von allen anderen Komponenten des Agenten gelesen und auch von einigen beschrieben werden. Das Planungsmodul erzeugt beispielsweise die Anweisungshierarchie. Das Koordinationsmodul erkennt wechselseitige Beziehungen zu anderen Agenten, erzeugt neue Anweisungen und Randbedingungen. Der Lokale Ablaufplaner ordnet die Aktivitäten zeitlich ein.

Planungsmodul

Das Planungsmodul ist für die Zerlegung oder Verfeinerung der Aufgaben zuständig. Es legt also die hierarchische Struktur der zu erledigenden Aufgaben in der Zustandsdatenbank ab. Die Komponente benutzt dazu eine Bibliothek, woraus fertige Pläne für vordefinierte Aufgaben abgerufen oder neue Pläne generiert werden können. Die Funktionalen Agenten haben natürlich auf ihre spezielle Funktion ausgelegte Bibliotheken. Bei der Erstellung eines solchen Plans werden den Anweisungen Attribute zugewiesen, um die Suchanfrage sinnvoll zu bearbeiten, zu steuern und zu überwachen.

Koordinationsmodul

Das Koordinationsmodul erkennt bestimmte Muster im Anweisungsbaum der Zustandsdatenbank und vermerkt, wo Wechselbeziehungen zu anderen Teilaufgaben oder gar anderen Agenten bestehen. Es erzeugt daraufhin zusätzliche Anweisungen und legt auch erste Vereinbarungen fest, wann z.B. eine Aktion relativ zu einer anderen ausgeführt wird. Die verschiedenen Koordinationsmechanismen werden im Abschnitt *Koordinationsmechanismen* auf Seite 178 genauer untersucht.

Lokaler Ablaufplaner

Der Lokale Ablaufplaner liest die Anweisungsstruktur der Zustandsdatenbank und erzeugt daraus unter Berücksichtigung der lokalen und auch der nicht-lokalen Vereinbarungen mehrere mögliche Ablaufpläne. Diese legen dann die Reihenfolgen und zeitlichen Abläufe des Suchprozesses fest. Die

Syntax solcher Ablaufpläne ist im wesentlichen die von der schon erwähnten Sprache TAEMS. Mit einem heuristischen DTT (Design-To-Time) Ablaufplaner [4] wird dann daraus nach bestimten Kriterien der endgültige Ablaufplan bestimmt. Bei MACRON werden von der DTT-Komponente sogar mehrere Alternativen ausgearbeitet, wobei jede versucht, bestimmte Kriterien zu optimieren. Die Kriterien sind zum Beispiel Geschwindigkeit, Qualität oder Kosten. Manche Anfragen sollen besonders schnell bearbeitet werden auch wenn dabei die Qualität leiden muß. Machmal möchte man eine möglichst genaue Antwort oder eine die nichts kostet, denn es gibt Informationsquellen, die Gebühren verlangen. Andere wollen, daß alle Aspekte einer Suchanfrage bearbeitet werden. Beim Erstellen der Pläne greift der DTT-Ablaufplaner auf statistische Daten zurück.

Entscheidungstreffende Instanz

Die Entscheidungstreffende Komponente wählt aus den Alternativen vom Lokalen Ablaufplaner den jeweils zu einem bestimmten Zeitpunkt besten Plan. Wie schon erwähnt, realisieren die verschieden Alternativen des Lokalen Ablaufplaners optimierte Lösungen bzgl. eines Kriteriums. Es kann aber zur Ablaufzeit Veränderungen im Suchraum geben, oder verschiedene Aspekte einer Anfrage sollen nach verschiedenen Kriterien behandelt werden. Deshalb kann die Entscheidungstreffende Komponente zur Laufzeit zwischen den verschiedenen Plänen wechseln [13]. Die Entscheidungstreffende Instanz entlastet somit den Lokalen Ablaufplaner.

Ausführungsmonitor

Der Ausführungsmonitor überwacht den Suchprozeß. Wenn er mit statistischen Daten versorgt wird, setzt er sich selbst Marken und überprüft so, ob eine Aktion ihren Vorgaben hinterherhinkt. Bei Bedarf veranlaßt er eine neue Generierung der Ablaufpläne. Fehlerhafte Aktionen werden ebenfalls von dieser Komponente erkannt.

Koordinationsmechanismen

Das CIG-Konzept [12] sieht eine verteilte Problemlösung vor. Das erfordert jedoch Absprachen zwischen den Agenten, denn die Teilaufgaben haben mitunter wechselseitige Abhängigkeiten verschiedener Arten. Die Lösung mancher Teilaufgaben kann das Lösen anderer erleichtern oder sogar erst ermöglichen. Aufgaben können sich auch zum Teil überlappen. Diese Beziehungen zwischen den Teilproblemen werden vom Koordinationsmodul aufgedeckt und in der Anweisungsstruktur in der Zustandsdatenbank festgehalten und auch bei Bedarf anderen Agenten mitgeteilt. Der dafür von Lesser entwickelte Ansatz heißt FA/C (Functionally Accurate, Cooperative) und sieht den Austausch von Informationen über den aktuellen Stand ihrer Aktivitäten zwischen den Agenten vor. Damit steigert sich die Effizienz, weil unter anderem auch ungewollte Redundanz vermieden wird. Es gibt verschiedene Kooperationsmechanismen.

- Alle Anweisungen in der Anweisungsstruktur, die andere Agenten betreffen könnten, aber von denen angenommen wird, daß sie diesen nicht bekannt sind, werden ihnen mitgeteilt.
- Resultate werden anderen Agenten mitgeteilt, wenn angenommen wird, daß sie diesen helfen könnten.
- Mechanismen, die Teilaufgaben an andere Agenten übertragen, wenn sie zu deren Funktionalität passen.
- Mechanismen, die Redundanz aufdecken und behandeln.

11.3.2 Informationssammelnde Agenten

Diese Agenten sind die, die eigentlich Daten sammeln. Sie erledigen primitive Tätigkeiten wie Datentransfer, sie füllen, z.B. wenn sie externe Suchmaschinen wie Altavista oder Lycos benutzen, HTML-Eingabeformulare aus, stellen Anfragen an externe odere sonstige Datenbanken, sie suchen in Texten nach Schlüsselworten und sammeln aus HTML-Dokumenten vorkommende URLs. Sie sind je für eine der Aufgaben stark spezialisiert und arbeiten nicht mit wissensbasierten Methoden. Prinzipiell sind solche Suchenvorgänge nicht an bestimmte Informationquellen wie das WWW gebunden sondern können in jeder beliebigen Umgebung agieren. Es bedarf dann lediglich neuer Sammleragenten, die die neuen Quellen nutzen können. Die Gesamtarchitektur und die Organisation sind vom Suchumfeld unabhängig. Das heißt, diese Sammleragenten bieten den DECAF-Agenten eine abstrakte Sicht auf die Informationsquellen. Die Standardkommunikationssprache ist, wie bei allen anderen Agenten bei MACRON, KQML.

11.3.3 Agenten für die Benutzerschnittstelle

Weil der Nutzer die Möglichkeit haben soll manipulierend in einen Suchprozeß einzugreifen, muß ihm auch ein verständlicher Überblick angeboten werden. Das stellt aber auch Anforderungen an die Darstellung aller relevanten Zustandsgrößen, und die Manipulationen müssen auch in den Suchprozeß eingebunden werden. Bei MACRON wird dem Anwender die Anweisungsstruktur, also die Hierarchie der Teilaufgaben, angezeigt. Dadurch kann er eine Aktion anwählen, die er näher betrachten möchte. Er hat auch die Möglichkeit, die Parameter der Anweisungen neu zu belegen und dies in den Prozeß einzubringen. Andere Manipulationen kann er nicht vornehmen.

11.4 Aktueller Stand der Implementation

Die Entwicklung von MACRON befindet sich noch im Anfangsstadium. Es gibt einen einfachen Prototypen [10][Abb. 11.3]. Dieser nutzt einen Web-Browser als Userschnittstelle mit HTML Eingabefeldern. Für jede Anfrage wird ein Anfrage-Agent aktiviert. Dieser kontaktiert einen Planer, welcher den Suchplan erstellt und die elementaren Aufgaben an die Sammleragenten verteilt. Bis jetzt werden als Quellen Djanews, Macinfo, INQUERY und andere WWW-Quellen benutzt. Der Planer ist auf das Auffinden von Macintosh Produkten ausgelegt.

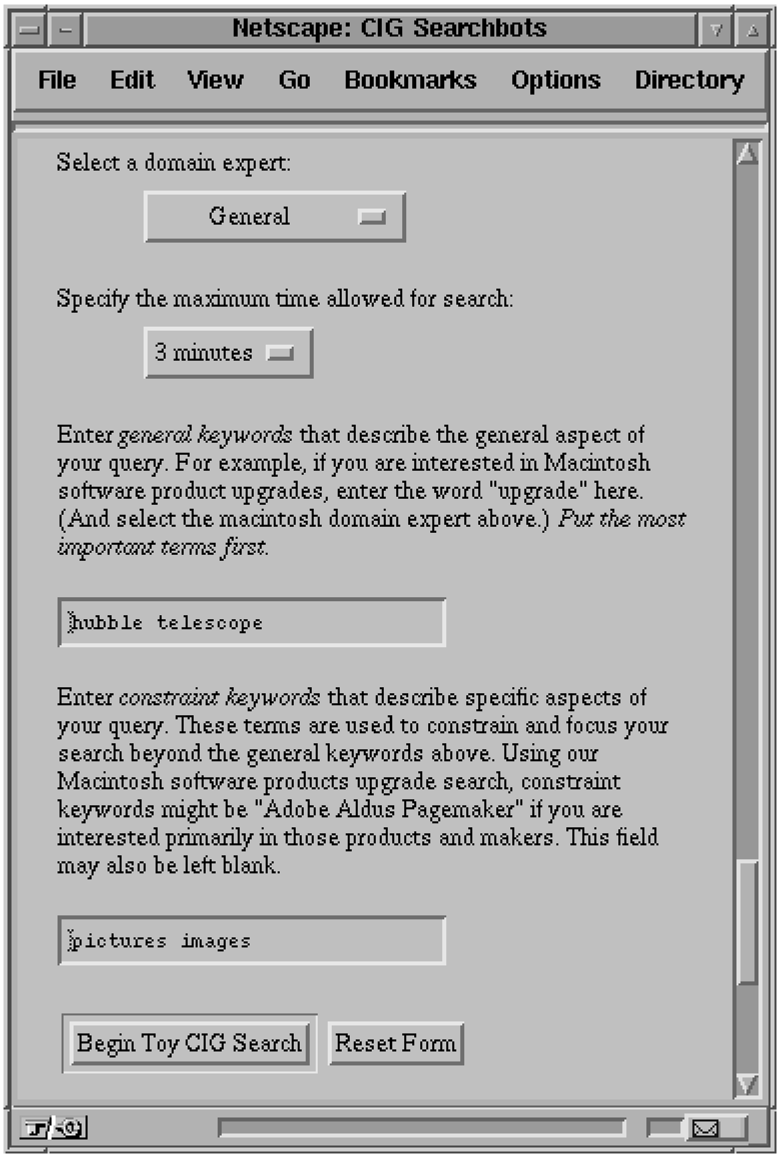


Abbildung 11.3: Einfacher Prototyp

11.5 Fazit und Ausblick

Bisherige Informationssammelsysteme benutzen entweder keine wissensbasierten Methoden oder beachten die Abhängigkeiten zwischen den Teilaufgaben und damit auch zwischen den Agenten nicht oder nur unzureichend. Es gibt ein System von Knoblock [1], das sowohl wissensbasierte Methoden einsetzt, als auch wechselseitige Beziehungen berücksichtigt. Dieses System

optimiert jedoch seine Suchablaufpläne nicht nach Kriterien wie Geschwindigkeit oder Qualität der Ergebnisse, auch Kostenminimierung spielt bei seinem Ansatz keine Rolle. MACRON geht auf all diese Mängel ein, wie jedoch der Stand der Implementierung zeigt, sind noch nicht alle Punkte des Konzepts umgesetzt.
Sicher wird ein solcher Ansatz den herkömmlichen passiven Strategien überlegen sein, weil sich erstens nicht alle Fragen, die einmal gestellt werden könnten, in einer Datenbank ablegen lassen, so daß man die Antworten nur noch abrufen muß, und zweitens wäre dies der falsche Ansatz, da sich die angebotenen Informationen ständig verändern und die gleiche Anfrage zu verschiedenen Zeiten zu unterschiedlichen Antworten führen muß. Ein anderer Ansatz als der mit der verteilten Problemlösung ist nicht realistisch. Ein aktiver Agent kann nicht alle der ganz verschiedenartigen anfallenden Arbeiten verrichten. Also bleibt nur die verteilte Problemlösung mit mehreren spezialisierten Agenten.
Was meiner Meinung nach Problematisch erscheint ist das Interagieren des Anwenders zur Laufzeit. Die Belegung von Parametern zu Anfang, dies können Angaben über die gewünschte Qualität oder Zeitlimits sein oder was sonst noch sinnvoll erscheint, müsste eigentlich ausreichen, um die Suche nach eigenen Vorstellungen zu gestalten. Interaktionen des Users bringen zeitliche Verzögerung, und zudem ist der User unter Umständen auch überfordert, vor allem bei Laienanwendern. Dies gilt nicht, wenn die Anfragen umfangreich sind und Spezialisten mit dem Tool umgehen. Solche Fachleute können aufgrund ihrer Erfahrung ein Suchsystem steuern und lenken. Benutzermanipulationen können also sinnvoll sein. Es kommt aber darauf an in welcher Umgebung das System laufen soll. Wenn man Manipulationen des Benutzers ausschließt, wird die Organisation des Systems stark vereinfacht, was durchaus auch eine Rolle spielt, denn es hat keinen Sinn beliebig komplexe Konzepte umzusetzen, die gegenüber einfacheren keinen oder kaum Gewinn bringen. Es bleibt immer auch die Kosten-Nutzen Frage zu beachten.

Literaturverzeichnis

[1] Y. Arens C. A. Knoblock. An architecture for information retrieval agents. In *Working Notes of the AAAI Spring Symposium on Software Agents*, 1994.

[2] Y. Arens C. A. Knoblock and C. Hsu. Cooperating agents in information retrieval. In *2nd International Conference on Cooperating Information Systems*, 1994.

[3] Tim Finin. Knowledge query and manipulation language. Technical report, University of Maryland Baltimore County, 1995. http://www.cs.umbc.edu/kqml/.

[4] Lesser Garvey. Design-to-time scheduling with uncertainty. Technical report, University of Massachusetts, 1995.

[5] H. Ghenniwa. Folien zu einem vortrag. Technical report, University of Waterloo Canada, 1995. http://sail.uwaterloo.ca/ sd422/Presentations/pr1.ps.

[6] V. R. Lesser K. S. Decker. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1992.

[7] V. R. Lesser K. S. Decker. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems*, 1995.

[8] M. V. Nagendra Prasad Keith Decker, Victor Lesser and Thomas Wagner. Macron: An architecture for multi-agent cooperative information gathering. Technical report, University of Massachusetts, 1995.

[9] Victor Lesser. Generic coordination strategies for agents. Technical report, University of Massachusetts, 1995. http://dis.cs.umass.edu/research/gpgp.html.

[10] Victor Lesser. A simple macron demonstration. Technical report, University of Massachusetts, 1995. http://dis.cs.umass.edu/research/searchbots.html.

[11] Victor Lesser. Taems: A framework for task analysis, environment modeling, and simulation. Technical report, University of Massachusetts, 1995. http://dis.cs.umass.edu/research/taems.html.

[12] Prasad Oates, Lesser. Cooperative information gathering: A distributed problem solving approach. Technical report, University of Massachusetts, 1994.

[13] Alan Garvey Victor Lesser, Keith Decker. A negotiation-based interface between a real-time scheduler and a decision-maker. Technical report, University of Massachusetts, 1994.

Kapitel 12

Eine Einführung in lernende Assistenten am Beispiel von CAP

Marcus Flehmig

Übersicht

Die Anpassung von Programmen auf die persönlichen Bedürfnisse und Eigenarten eines Anwenders wird durch die Zunahme an Möglichkeiten der Einstellungen und Funktionen der Programme ein immer umfangreicheres Unterfangen. Das zu erleichtern kann eine mögliche Aufgabe von lernenden Assistenten sein. Man kann Programme um die Möglichkeit, von der Art der Benutzung desselben auf bestimmte Eigenarten und Vorlieben des Anwenders zu schließen, erweitern, um ihn so produktiv in seinem Arbeitsfluß zu unterstützen. Der Calender-Apprentice CAP ist ein Beispiel für ein solches Programm. Er ist in der Lage aus Termineingaben auf bestimmte Eigenschaften des Termins zu schließen und später Vorschläge zur Termingestaltung zu machen. Es wird neben der Methodik, die sich hinter CAP verbirgt, auch auf Erfahrungswerte aus der Praxis eingegangen.

Vorwort

Die vorliegende Abhandlung basiert auf einer Abhandlung von Tom Mitchell mit dem Titel "Experience With a Learning Personal Assistant" [1]. Die Gliederung des Originals ist meiner Meinung nach sehr sinnvoll und wurde deshalb vollständig übernommen. Der Text versteht sich darüber hinaus nicht als eine einfache Übersetzung, sondern als eine Zusammenfassung und einführende Hinleitung an das Thema: Softwareassistenten, lernende Programme und CAP.

12.1 Einleitung

Computersysteme und die damit genutzten Programme werden immer komplexer und umfangreicher. Umfangreicher nicht nur im Sinne immer weiteren Hinzufügens von Funktionen und Einstellmöglichkeiten, sondern auch umfangreicher im Sinne von räumlicher Ausdehnung - in der Form von Netzwerken. Computernetzwerke finden neben dem universitären oder kommerziellen Bereich immer mehr Verbreitung in privaten Haushalten. Homebanking, Teleshopping, Mail, News oder Net-Surfing sind Begriffe, die heute vielen geläufig sind. Die sich dahinter verbergenden Tätigkeiten werden erst durch Computernetzwerke, wie z.B. dem InterNet, ermöglicht.

Allerdings steigt durch die Bedienvielfalt auch der Aufwand der Konfiguration, der Anpassung an die persönlichen Vorlieben und Eigenheiten eines Benutzers. Programme ermöglichen zwar oft, sogar Feinheiten bis ins Detail einzustellen, aber der Aufwand, diese Einstellungen zu pflegen, übersteigt die initiale Arbeit nicht selten. Man nehme als Beispiel einen WWW-Browser. Eine Eigenheit eines Benutzers könnte z.B. sein, daß er jede URL(universal resource location), die er im Internet besucht hat, nach einer Verzweigung von dieser zu einer anderen Adresse, in sein persönliches Merkbuch aufnimmt. Natürlich kann der Programmierer für alle Eigenarten und Gewohnheiten, die er kennt, eine Konfigurationseinstellung anbieten. Die entstehende Vielfalt führt dann zu dem oben beschriebenen Anwachsen des Aufwandes. Ein Programmassistent kann hier unterstützend einwirken und es finden sich noch eine Vielzahl von weitern Einsatzmöglichkeiten für netzorientierte Softwareassistenten [9] [15] [16] [7] [17] [2] [4].

Die Frage, die sich also stellt ist: Wie kann man den Konfigurationsaufwand verringern, bzw. wie kann man den Benutzer durch guten Entwurf eines Softwareassistenten am besten unterstützen? Da sich ein Programmassistent dem Benutzer durch Lernen von Erfahrungswerten anpassen soll, kann man daher die Bemühungen, wie folgt, umreißen:

- Man stellt zunächst eine Schnittstelle zur Verfügung, die es dem Anwender erlaubt, seine Aufgaben komfortabel zu meistern
- Wenn das System benutzt wird, behandelt man jede Eingabe eines Benutzers als eine Information, die in den Lernprozeß einfließt.
- Aus den Informationen lassen sich dann allgemeine Regeln ableiten, die benutzt werden können, um den Anwender effektiv zu unterstützen.

Mitchell nennt diese Art von Programmassistent "Learning Apprentice" [8]. Dieser Learning Apprentice stellt quasi einen interaktiven Assistenten dar, der die Eigenarten seines Benutzers beobachtet. Er kann z.B. eine wie oben beschriebene Gewohnheit erkennen, lernen und selbständig eine entsprechende Aktion ausführen. Das bedeutet eventuell einen anfänglichen Mehraufwand bis das System auch funktioniert, der aber immer weiter schwinden kann. Das Konzept steht und fällt nämlich damit, wie treffend die Entscheidung des Assistenten, bzw. wie genau sie an der Vorstellung des Benutzers ist. Daher gibt es Forschungen, die untersuchen, wie man auf anderem Wege als oben beschrieben eine Wissensbasis für einen Learning Apprentice aufbauen kann [8] [10] [5]. Man kann den Lernprozeß z.B. explizit starten und nimmt so nicht jede Eingabe als Information auf. Diese halbautomatische Methode nennt man auch "Learning by Demonstration" [2] [6] [17].

Neben Erleichterungen im Umgang mit dem Programm, wie oben beschrieben, kann man sich im nächsten Schritt vorstellen, daß Softwareassistenten auch inhaltlich auf den Programmfluß einwirken, z.B. indem in einer Zahlungsanweisung zu einer bestimmten Bank automatisch die entsprechende Bankleitzahl eingetragen wird, die dem System aber nicht von vornherein bekannt sein muß. Bei einem "Online"-Kalender könnte man sich vorstellen, daß die Dauer oder die Teilnehmer eines Termins vorgegeben werden. Ferner sind auch abstraktere Eingaben denkbar. Was aber überhaupt ist ein "Online"-Kalender? Allgemein gesagt nutzt ein "Online"-Kalender im Gegensatz zu einem "Offline"-Kalender die Möglichkeiten eines Computernetzes. Ein konventioneller Kalender, oder eben "Offline"-Kalender, verwaltet die Termine eines Benutzers. Terminabsprachen finden verbal statt und werden dann im Kalender vermerkt. Die Funktionalität gegenüber einem Papierkalender ist nur unwesentlich höher. Interessanter wird es, wenn man ein Computernetz nutzen kann. Terminabsprachen finden über das Netz statt. Man plant einen Termin, lädt andere Personen dazu ein, die online entscheiden können, ob sie teilnehmen oder nicht. Zusagen werden sofort auf beiden Seiten vorgemerkt. Absagen werden ebenso automatisch vermerkt. Kurz, es können Mitteilungen über das Netz versand werden. Der "Calender-Apprentice", kurz CAP, ist ein "Online"-Kalender. Er ermöglicht den Versand bzw. Empfang von Mitteilungen über das Netzwerk, er bietet Kalenderfunktionen und ist in der Lage aus den Eingaben des Benutzers zu lernen, um daraus eine Wissensbasis zu erstellen. Er erfüllt damit formal alle oben genannten Anforderungen an einen Learning Apprentice.

12.2 Der Calender-Apprentice: CAP

Der Kalenderassistent bietet einen interaktiven Zugang zu einem Terminkalender mit e-mail-Unterstützung. Der Terminkalender kann bearbeitet werden, indem der Benutzer Termine hinzufügt, löscht, verschiebt, kopiert, absagt oder als vorläufig bzw. fest einstuft. Ferner ist er in der Lage, Einladungen, bzw. Absagen oder Erinnerungen über das Computernetz via e-mail, als elektronische Post, zu verschicken, die dann wiederum von anderen Benutzern evtl. als fest oder vorläufig eingestuft, mit Notizen zurückgeschickt oder einfach nur zur Kenntnis genommen werden können. Auch ist es möglich einen Terminkalender auf vielfältige Weise zu Papier zu bringen und die Art der Anzeige zu verändern, um unter anderem einen Wochen- oder Monatsüberblick zu bekommen.

Es stellt sich hier nun wieder die Frage: Wie kann man den Benutzer in seiner Tätigkeit unterstützen? Mitchell führt einen Vergleich mit einer Sekretärin, bzw. Sekretär, um dieser Frage näher zu kommen. Man stelle sich also eine Büroassistenz vor. Unter anderem fällt ihr vielleicht auch die Aufgabe der Terminplanung zu. Zu Antritt ihrer Arbeit kennt sie keinerlei Präferenzen des Betreffenden. Es ist unvermeidlich zu jedem Termin alle notwendigen Informationen explizit zu erfragen. Erst mit der Zeit, nach einer gewissen Einweisung und Einarbeitung, kann die Assistenz selbständig Termine vereinbaren. Sie hat gelernt wann, wo und für wie lange ein bestimmtes Treffen in Abhängigkeit von den Teilnehmer oder der Art des Treffens stattfindet. Sie kann erkennen, welche Termine wichtiger als andere sind, um sie zu verschieben. Dieses Wissen aber verändert auch den Dialog zwischen der Person und ihrer Assistenz. Es können abstrakte Aufgaben, wie schon kurz erwähnt, gestellt werden. Es muß nicht mehr konkret und explizit jede Information einzeln gegeben werden, sondern es werden Vereinbarungen möglich, wie z.B. ein Termin mit einer bestimmten Person in der nächsten Woche. Die Assistenz ist in der Lage Zeitpunkt, Ort und Dauer festzulegen, den Termin mit der anderen Person abzustimmen und evtl. je nach Wichtigkeit andere Termine abzusagen oder zu verschieben, mit dem Ziel, den Auftraggeber zu unterstützen und zu entlasten.

CAP wurde mit der Vorstellung entwickelt, ein ähnliches Verhalten zu zeigen. Es soll sich von einem einfachen, passiven Eingabewerkzeug zu einem wissensbasierten Assistenten entwickeln, der jedem Benutzer individuelle Unterstützung gibt, indem er durch einen intelligenteren Dialog den Benutzer entlastet. Um dies zu erreichen, wurden Lernmethode entwickelt und mit

Hilfe maschinell Gelerntem ist CAP in der Lage, dem Benutzer bei der Terminvereinbarung, Dauer, Ort oder Datum vorzuschlagen. Denkbar ist auch ein halbautomatisches Verfahren basierend auf der erlernten Wissensbasis, um Absagen über das Netz zu verschicken.

12.2.1 Die Systemeigenschaften

Die Benutzeroberfläche von CAP ist in den Editor GNU-EMACS integriert. EMACS ist prädestiniert für eine solche Aufgabe, da er neben einer textorientierten Schnittstelle eine Programmierschnittstelle in LISP bietet. Im allgemeinen fragt CAP bei einer Neueingabe eines Termins nach der Art des Termins, den Teilnehmern, dem Datum, der Dauer, der Uhrzeit, dem Ort und ob der Termin schon fest ist oder vorläufig. Je nach Wissensstand schlägt CAP gewisse Daten vor, die übernommen oder überschrieben werden können. Die Vorschläge stammen aus zuvor gelernten Regeln, die mit Hilfe der schon eingegebenen und daraus abgeleiteten Daten ausgewählt werden. Jedesmal wenn der Benutzer einen Vorschlag annimmt oder verwirft wird ein Datensatz mit gewissen Trainingsdaten gespeichert, die in einem späteren Lernprozeß ausgewertet werden.

Ein typischer Datensatz könnte wie in Bild 12.1 aussehen. Durch die Gliederung kann man auf die unterschiedliche Herkunft der Daten schließen. Die Daten aus dem ersten Absatz stammen direkt aus den Eingaben des Benutzers, die aus dem mittleren Abschnitt aus der Programmumgebung. Die Daten des letzten Abschnittes werden aus Hintergrundwissen gewonnen, welches in einer zentralen Datenbank abgelegt sein kann und beim ersten Benutzen abgefragt und gespeichert wird oder lokal, z.B. als Teil der Konfiguration, abgelegt ist.

Aus diesen Daten leitet CAP Regeln ab, um Dauer, Ort, Tag, oder Uhrzeit vorzuschlagen. Ein Beispiel für diese abgeleitete Regeln findet man in Bild 12.2 Für jede Regel wird protokolliert, wie es zu ihrer Entstehung kam, linkes Zahlenpaar, und wie effektiv sie in der Anwendung ist, rechtes Zahlenpaar. Die Regeln werden in einer Liste nach diesen Zahlen sortiert gespeichert. Da die Zahlen eine Art Güte widerspiegeln, werden sie immer wieder aktualisiert und die Liste evtl. neu sortiert. Die oberste Regel erzeugt im Falle einer Übereinstimmung mit entsprechenden Eingabedaten den Vorschlag. Ob der Vorschlag übernommen wird oder nicht, geht wieder in die Bewertung ein. Die Daten werden aber nicht sofort ausgewertet, sondern zwischengespeichert und über Nacht verarbeitet. Vorschläge zu einem bestimmten Datum gehen nicht direkt aus den Regeln hervor, da diese Information viel zu speziell wäre, sondern werden aus Regeln den Wochentag betreffend abgeleitet. Angenommen das heutige Datum wäre D und eine Regel sage aus, daß ein Treffen an einem Mittwoch stattfinden solle, so würde

request-5-27-1992-48:
attendees: thrun
event-type: meeting
date: (29 5 1992)
time: 1400
duration: 30
location: weh5309
confirmed?: yes

displayed-week: (25 5 1992)
action-time: 2915977709
action-date: (27 5 1992)
previous-request: request-5-27-1992-13
previous-prompt: confirmed=yes

position-of-attendees: project-scientist
previous-attendees-meeting: request-5-20-1992-1
next-attendees-meeting: none
lunchtime?: no
number-of-attendees: 1
cmu-attendees?: yes
attendees-in-toms-group?: yes
known-attendees: yes
day-in-week: Friday
end:time: 1500
busyness-of-attendees: 2
single-attendee?: yes

Abbildung 12.1: Ein nach Eingabe eines Termins erzeugter Datensatz

CAP entweder den ersten Mittwoch nach D oder den Mittwoch aus der Woche, die gerade angezeigt wird, je nachdem welcher Termin später liegt, vorschlagen. Die Uhrzeit eines Termins wird direkt aus den Regeln geschlossen. Ist zu dieser Zeit allerdings schon ein Termin vorgesehen, so schlägt CAP eine Zeit vor, die möglichst nah an der ursprünglichen liegt.

12.2.2 Die Lernmethoden

Allabendlich verarbeitet CAP die für den Lernprozeß gespeicherten Daten, um seinen Bestand an Regeln zu erweitern und zu pflegen. Der Lernprozeß benutzt dafür einen Algorithmus, der dem ID3 [11] [12] sehr ähnlich ist. ID3 liefert einen Entscheidungsbaum als Ergebnis, in dem jeder Pfad von der Wurzel zu einem Blatt eine Regel darstellt(siehe auch Kapitel 12.5). Um die allgemeine Anwendbarkeit der Regel zu erhöhen, können auch Vorbedingungen entfernt werden, wenn hierdurch die Leistungsfähigkeit der Regel gesteigert werden kann. Der Lernprozeß benutzt als Eingabe die in Bild 12.1 gezeigten Daten und produziert die in Bild 12.2 dargestellten Regeln. Jeden Abend lernt CAP 40 bis 50 Regeln und zwar jeweils für die Dauer, den Ort, die Zeit und den Wochentag. Darin enthalten sind etwa 5 bis 20 neue Regeln, die zu den bestehenden Regeln hinzugenommen werden. Die Regeln, die sich nicht von den vorhandenen unterscheiden, werden verworfen. Die neuen Regeln werden in Listen gespeichert. Es existiert für jedes der Attribut Dauer, Ort, Zeit und Wochentag eine eigene Liste. In diese werden die Regeln abgelegt, die Duplikate evtl. entfernt und die Liste wird nach Leistungsfähigkeit der Regeln, wie oben erwähnt, sortiert. Der Lernprozeß wird in Bild 12.3 nochmals verdeutlicht.

Bei der Entwicklung eines lernenden Softwareassistenten tauchen einige Fragen auf. Die erste, die sich stellen ließe, wäre, welche Lernmethode man bevorzugen solle. Bei der Entwicklung von CAP wurden zwei Methoden in Betracht gezogen: die der Entscheidungsbäume [11] und die der neuronalen Netze [13]. Es stellte sich in Experimenten heraus, daß das Ergebnis beider Methoden gleichwertig war [4] und man entschloß sich deshalb für die Entscheidungsbäume. Denn einerseits ist es wichtig, daß in Bereichen in denen Menschen mit maschinell Gelerntem umgehen sollen, Regeln leicht zu verstehen sind, um sie manuell nachbearbeiten zu können. Wie man in Bild 12.2 anschaulich sieht, sind die von CAP gelernten Regeln unmittelbar verständlich, wohingegen man bei neuronalen Netzen mit Gewichtungsfunktionen arbeiten muß, die nur schwer interpretierbar sind. Andererseits sind Regeln, die mit Hilfe von Entscheidungsbäumen erzeugt werden, stückweise zerlegbar. Ergebnisse aus neuronalen Netzen bilden ein monolithisches System und sind nicht modular. Das System ist in der Lage mit Hilfe der Regeln aus Bild 12.2 über die Leistungsfähigkeit jeder Regel zu wachen, so

If position-of-attendees is grad-student, and
single-attendee? is yes, and
sponsor-of-attendees is mitchell;
Then duration is 60.
Training: 6/11 Test: 31/38

If position-of-attendees is faculty, and
department-of-attendees is scs, and
number-of-attendees is 2;
Then location is weh5220.
Training: 6/6 Test: 13/16

If position-of-attendees is grad-student;
Then location is weh5220.
Training: 21/21 Test: 56/59

If seminar-type is theo;
Then day-of-week is monday.
Training: 6/6 Test: 8/8

If department-of-attendees is edrc, and
day-of-week is friday,
Then time is 0830.
Training: 5/5 Test: 18/19

If course-name is 16-741;
Then time is 0930.
Training: 35/35 Test: 59/60

Abbildung 12.2: Einige erlernte Regeln. Jeden Abend lernt CAP aus den gespeicherten Daten etwa 5 bis 20 neue Regeln. Regeln werden nach ihrer Leistungsfähigkeit sortiert, die sich aus den beiden Zahlenpaaren ergibt. Das linke Zahlenpaar aus dem ersten Beispiel bedeutet, daß die Regel im Lernvorgang in 6 von 11 Fällen korrekt war, und im weiteren Verlauf, also bei der Eingabe von neuen Terminen in 51 von 86 Fällen angewendet, bzw. benutzt wurde.

daß z.B. ineffektive Regeln entfernt und taugliche Regeln beibehalten werden können

Die zweite Frage, die aufgeworfen wird, ist, wie hoch die Anzahl der zusammen zu verarbeitenden Trainingsdatensätze sein darf. Eine hohe Anzahl führt wünschenswerterweise zu statistisch sehr sicheren Regeln. Erhöht man die Anzahl aber immer weiter, so vergrößert man auch den zeitlichen Betrachtungsrahmen und zieht so eventuell auch alte und damit oft irrelevante Daten mit in die Betrachtung ein. Beispielsweise kommen bestimmte Termine nach Abschluß eines Projekts nicht mehr zustande. Es ist einleuchtend, daß es nicht sinnvoll ist, Datensätze, die vor einem solchen Abschluß gespeichert wurden, weiter in den Lernprozeß einzubeziehen. In einem Feldversuch mit CAP fand man heraus, daß eine Anzahl von 180 Trainingsdatensätzen sinnvoll erscheint, was in dem Versuch etwa einem zeitlichen Rahmen von zwei Monaten entsprach. In einer schnell wechselnden Umgebung kann es daher sehr nützlich sein, wenn Methoden zur Verfügung ständen, die auch aus einer geringen Anzahl von Datensätzen eine sinnvolle Regel ableiten könnten.

1. Aktualisieren der Leistungsdaten für jede Regel, um die Verarbeitungsgeschwindigkeit zu erhöhen

2. window-examples nimmt die letzten 180 eingegebenen Termine auf

3. training-examples besteht aus 120 zufällig aus window-examples ausgewählten Datensätzen

4. test-examples ist gleich window-examples ohne training-examples

5. Für jedes Attribut aus Dauer, Ort, Wochentag, Zeit führe folgende Punkte aus:

    - Erzeuge mit Hilfe des Algorithmus ID3 einen Entscheidungsbaum für Daten aus training-examples
    - Bilde aus jedem Pfad des Entscheidungsbaums eine Regel
    - Entferne alle Vorbedingungen, die die Leistungsfähigkeit der Regel, angewendet auf window-examples, nicht beeinträchtigen
    - Protokolliere für jede neue Regel wie oft ein Beispiel aus test-examples darauf zutrifft und wie oft nicht
    - Sortiere die neue Regel nach den Kriterien aus dem vorangegangen Punkt in die entsprechende Liste ein.

Abbildung 12.3: Der Lernprozeß, der jeden Abend gestartet wird.

Eine weitere Frage, beschäftigt sich mit dem Umfang der Attribute, die beim

Lern- bzw. Aufzeichnungsprozeß betrachtet werden. CAP könnte einen Termin potentiell durch hunderte von Attributen beschreiben, z.B. wären Attribute wie "the department of the attendees of the meeting before the previous meeting with the same attendees" (also ein komplexer langer Ausdruck als ein Attribut) denkbar. Einerseits ist es sicher sinnvoll, viele Attribute zu betrachten, um möglichst allgemeingültige Regeln zu erhalten. Andererseits allerdings muß man auch die Anzahl der Trainingsdatensätze erhöhen, wenn man die Anzahl der zu betrachtenden Attribute erhöht, um zu Ergebnissen mit der gleichen Zuverlässigkeit zu kommen. Die eigens dafür entwickelte Methode "Greedy Attribute Selection" [3] löst diesen Konflikt. Sie wählt automatisch für kommende Lernprozesse die Attribute aus, die einen möglichst großen Erfolg versprechen, indem der mit diesem Attribut erzielte Lernerfolg in der Vergangenheit betrachtet wird. Hierdurch wird der Lernprozeß dynamisch an verschiedene Benutzer und verschiedene Arten von Regeln, z.B. für den Ort, die Dauer etc., angepaßt.

Schließlich muß man sich aber auch dem Problem stellen, wie neue Regeln zu der Menge der bereits bestehenden hinzugefügt werden können. Angenommen der Lernprozeß wird nur auf eine bestimmte Menge aktueller Daten angewendet und es gäbe erfolgreiche Regeln, die schon früher hergeleitet worden sind, aber nicht aus den aktuellen Daten herleitbar sind. Wie können dann neue Regeln mit den schon gelernten Regeln zusammengefügt werden? CAP bedient sich hierbei der Leistungsfähigkeit der Attribute, ausgedrückt durch die oben erwähnten Zahlenpaare. Je nachdem wie erfolgreich eine Regel ist, gemessen daran wie häufig ein durch sie erzeugter Vorschlag angenommen wurde, wird sie bei der Zusammenführung berücksichtigt werden. Die Regeln werden dann wieder nach diesem Kriterium sortiert und ein netter Nebeneffekt dieser recht intuitiven Vorgehensweise ist, daß sogar ältere Regeln, die sich aber als sehr erfolgreich erwiesen haben, neben neuen Regeln ganz oben in der Liste rangieren und alte ineffektive Regeln sich sehr bald ganz unten wieder finden und herausgenommen werden können.

12.3 Erfahrungen aus dem täglichen Einsatz

Der Kalenderassistent CAP wurde von mehreren Personen im täglichen Einsatz erprobt. Insgesamt wurde CAP von sechs Personen mit unterschiedlicher Intensität eingesetzt. Die Erfahrungen von über 16 Monaten zweier Probanden soll in diesem Abschnitt genauer betrachtet werden. Beide arbeiten im universitären Umfeld und besitzen einen geschäftigen Terminkalender. Die betrachteten Daten stammen aus der täglichen Arbeit mit CAP.

Eine mögliche Art, zu untersuchen, wie leistungsfähig sich CAP in der Praxis verhält, ist, aufzuzeigen wie oft ein von CAP gemachter Vorschlag ohne Veränderung vom Benutzer übernommen wird. In Bild 12.4 wird das Verhältnis in Prozent für die vier Attribute: Ort, Dauer, Wochentag und Zeit für beide Personen graphisch dargestellt. In der Grafik symbolisiert die durchgezogene Linie die Genauigkeit der gelernten Regeln und die gepunktete Linie zeigt im Vergleich dazu die Genauigkeit eines dynamisch berechneten, also immer aktualisierten, Defaultwertes an. Die Genauigkeit für ein bestimmtes Datum berechnet sich aus der Genauigkeit der Vorschläge der nachfolgenden 60 Terminvereinbarungen. Die beobachteten Zeiträume sind jeweils unter den Diagrammen angegeben.

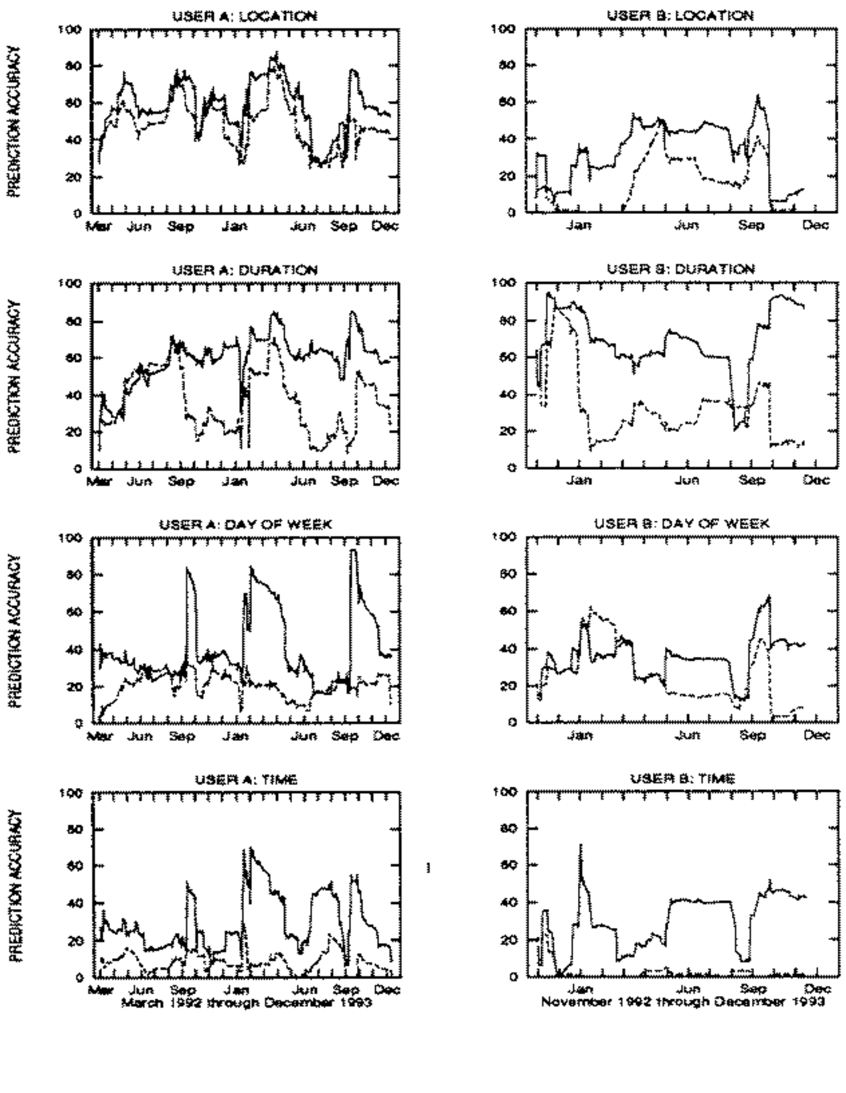


Abbildung 12.4: Genauigkeit der Vorschläge für verschiedene Attribute

Betrachtet man Bild 12.4 so kommt man zu dem Ergebnis, daß das System in der Lage ist bestimmte Eigenarten, bzw. Vorlieben des Benutzers zu erkennen und zu lernen, um korrekte oder zumindest bis zu einem gewissen Grad korrekte Vorgaben zu machen. Allerdings schwankt die Genauigkeit eines Vorschlags von Attribut zu Attribut und auch von Benutzer zu Benutzer, z.B. zwischen 69% mittlerer Genauigkeit für die Dauer und 31% für die Uhrzeit, beides bei Benutzer B. Bei einem wenig genutzten Kalender ist es allerdings ohnehin schwierig, eine genaue Voraussage zu machen, da es viele Möglichkeiten geben kann, wenn sich der Benutzer z.B. willkürlich für eine Zeit entscheidet. Die durchschnittliche Genauigkeit für alle Attribute und beide Benutzer liegt bei etwa 47%, wohingegen der einfache Defaultwert nur eine Genauigkeit von ca. 24% erreicht.

Aber die Genauigkeit schwankt nicht nur von Attribut zu Attribut und von Benutzer zu Benutzer, sondern auch von Zeitraum zu Zeitraum. Hier erkennt man die systemimmanente Dynamik und die Notwendigkeit der ständigen Aktualisierung der Regeln, bzw. der Daten. Beide Benutzer sind ja Universitätsangestellte, deren Zeitplan auch von der Vorlesungszeit abhängt. Die geringste Genauigkeit wird offensichtlich beim Wechsel von einem Semester zum anderen erreicht, z.B. im September. Man sieht bei Semesteranfang einen tiefen Einschnitt bei der Genauigkeit, nämlich gerade dann, wenn alte Regeln durch neue zu ersetzen sind. Durch die dynamische Lernfähigkeit von CAP werden diese Einschnitte aber meistens wieder ausgeglichen, da CAP neue Regeln lernt, die die neuen Anforderungen des Benutzers widerspiegeln, und die alten Regeln "vergißt".

Doch obwohl offensichtlich ist, daß CAP etwas sinnvolles gelernt hat, ist es dennoch enttäuschend, daß die Genauigkeit der Vorschläge nicht höher liegt. Den Ansprüchen an eine interaktive Eingabehilfe kann CAP zwar gerecht werden, aber ein Ziel der Entwicklung war, den Benutzer so weit zu entlasten, daß bestimmte Vorgänge vollständig automatisiert werden können. Um die Genauigkeit zu erhöhen, ist es denkbar, die Lernmethode, die Anzahl der Trainingsdatensätze oder die Anzahl der Attribute zu verändern. Diese Methoden können aber keine einschneidenden Veränderungen bewirken, wie oben bereits kurz erwähnt wurde. Wenn sich CAP aber nur auf Terminvereinbarungen beschränkt, die mit hoher Wahrscheinlichkeit unverändert angenommen werden, so könnte zumindest ein Teil der Vereinbarungen automatisiert werden. Allerdings kann andererseits natürlich dann nicht mehr für alle Terminvereinbarungen eine Vorgabe gemacht werden, die vollständig vom Benutzer angenommen wird. Vielmehr kann für die regelmäßigen Termine eine solche Voraussage getroffen werden. Wenn CAP also in der Lage wäre, regelmäßige Termine anhand der Wahrscheinlichkeit, daß alle Vorgaben übernommen werden, zu erkennen, so hätte der Benutzer ein Werkzeug, das ihn zumindest in einem Teil seiner Arbeit wesentlich entlasten würde. CAP würde analog zu einer Büroassistenz, wie oben beschrieben, nur noch schwierigere Fälle zum Benutzer durchreichen und Termine, die mit hoher Wahrscheinlichkeit vollständig vom Benutzer akzeptiert werden, eigenständig vereinbaren.

Es ist somit noch zu untersuchen, auf welche Weise CAP zwischen Fällen unterscheiden kann, in welchen es auf seine Wissensbasis vertrauen kann und in welchen nicht. Man betrachte dazu folgendes Experiment: Alle Regeln zu dem Attribut Ort, die an einem bestimmten Tag, hier dem 30 September 1992, gelernt worden sind, werden nach ihrer Genauigkeit, wie oben beschrieben, sortiert. Man betrachtet zuerst nur die erste, also oberste Regel und bestimmt, wie oft diese Regel bei den 60 Trainingsdatensätzen eine richtige Vorgabe macht. Dann nimmt man die beiden obersten und dann die obersten drei und fährt so fort. Zuletzt trägt man die Ergebnisse in ein Diagramm ein und erhält das Bild 12.5. Die "1" in der Graphik sagt somit aus, daß in 33% der Trainingsdatensätze die erste Regel zu 95% zutrifft. Nimmt man nun alle 52 Regeln zusammen, so erreicht man eine Deckung von 100%, aber erreicht dabei nur eine Genauigkeit von 67%. Mit dem Begriff Deckung ist demzufolge der Prozentsatz gemeint, für den eine Aussage in Form einer Vorgabe, gemacht werden kann. Das Diagramm zeigt also, daß CAP sehr wohl unterscheiden kann, wann es seine Wissensbasis anwenden kann, indem es die Genauigkeit einer Menge von Regeln bezogen auf nachfolgende Trainingsdatensätze beobachtet. Wenn CAP unter Berücksichtigung dieser Information nur dann eine Vorgabe macht, wenn eine gewisse Wahrscheinlichkeit erreicht wird, so läßt sich die mittlere Genauigkeit auf Kosten der Deckung erhöhen.

In Bild 12.6 schließlich ist dargestellt, wie sich das Verhältnis von Wahrscheinlichkeit und Deckung ändert, wenn man annimmt, daß CAP nur einen Vorschlag macht, wenn eine Wahrscheinlichkeit von N% erreicht wird. N ist gleichbedeutend mit der Genauigkeit der zurückliegenden 60 Vorgaben. Es wurden acht Punkte ermittelt, wobei man N zwischen 0% und 95% variierte und die Genauigkeit und Deckung einzeichnete. Der rechte Punkt gibt die Genauigkeit und Deckung unter Einbeziehung aller Regeln an und zwar gemittelt für den gleichen Zeitraum wie in Bild 12.4. Im Verlauf nach links wird N größer. Man sieht, wie sich die Genauigkeit der Vorschläge für jedes Attribut erhöhen läßt, wenn man Abstriche in der Deckung macht.

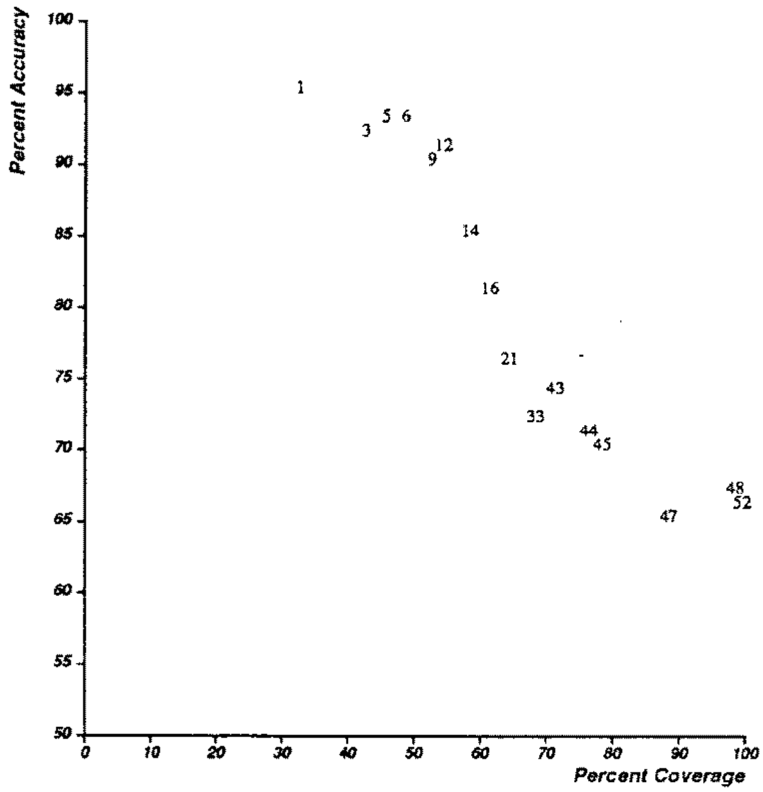


Abbildung 12.5: Zusammenhang von Genauigkeit und Deckung für einen einzelnen Tag

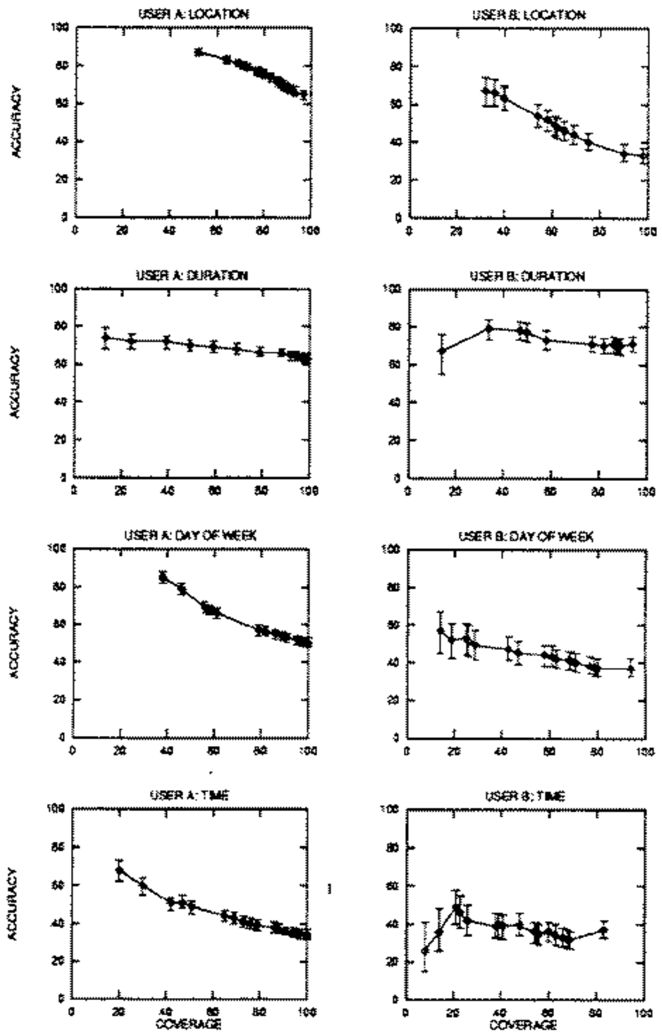


Abbildung 12.6: Zusammenhang zwischen Genauigkeit und Deckung für den Zeitraum aus Bild 12.4

12.4 Schlußfolgerung und Ausblick

Zusammenfassend kann man über CAP sagen:

- es ist möglich, durch Analyse von Eingabedaten benutzerspezifische Vorlieben im Eingabeverhalten mit einer Treffsicherheit zu lernen, die signifikant höher liegt als die Treffsicherheit anderer einfacherer Verfahren, wie z.B. der Defaultwertberechnung.
- Der Benutzer ist durch die Art der Speicherung der Regeln in der Lage, sie zu verstehen. Er kann so die Regeln auswerten, neue Regeln anfügen oder bestehende abändern.
- CAP kann als interaktive Eingabehilfe eingesetzt werden; aufgrund der beschränkten Treffsicherheit sind autonome Handlungen allerdings nicht möglich. Daher unterstützt CAP bisher auch kein automatisches Absagen von Terminvereinbarungen.
- Man kann die Genauigkeit der Regeln erhöhen, indem CAP nur dann einen Vorschlag macht, wenn die Wahrscheinlichkeit, daß dieser Vorschlag übernommen wird, sehr hoch ist. Die Wahrscheinlichkeit gewinnt man durch Beobachtung des bisherigen Erfolgs der Regel. Da ohnehin keine vollständige Automation möglich ist, kann man den Aufgabenbereich teilen: CAP verwaltet den Teil automatisch, für den die Treffsicherheit der Regeln sehr hoch ist, also z.B. die regulären Termine, und gibt die komplexeren Aufgaben an den Benutzer weiter.
- In einer sich ständig ändernden Umgebung, ist es für ein effektives Lernen sinnvoll, die Leistungsfähigkeit jeder einzelnen Regel ständig zu messen, die Menge der Regeln kontinuierlich zu aktualisieren, mit neuen Regeln zu vermischen und nach der gemessenen Leistungsfähigkeit zu sortieren. Die Modularität der Regeln ist hierfür sehr wichtig.

Neben diesen Erfahrungen aus dem Einsatz von CAP, sind aber auch andere Aspekte, die ein Softwareassistent erfüllen sollte, wichtig. Die Benutzerschnittstelle des Kalenders muß auch für Personen interessant sein, deren Eigenarten dem System noch nicht bekannt sind, denn um überhaupt Regeln aufstellen zu können, muß es möglich sein, daß System auch ohne Wissensbasis komfortabel zu bedienen. CAP ist mit einer einfachen textorientierten Schnittstelle ausgerüstet, mit deren Hilfe es dennoch unkompliziert möglich ist, CAP als bloße unintelligente Eingabehilfe für Termine zu gebrauchen. Vollständige und völlig korrekte Vorschläge werden nicht gefordert. Vollständig und völlig korrekt bedeutet eine 100% Genauigkeit bei 100% Deckung. Ferner sollte die Lernphase weitaus kürzer sein als die Anwendungsphase. Bei den Universitätsangestellten betrug die Lernphase etwa ein bis zwei Monate und die Anwendungsphase vier Monate. So kommt man auf die Dauer eines Semesters. Wichtig ist aber auch die Verfügbarkeit der Informationen aus den Attributen, d.h. es ist notwendig, daß das System auch die Informationen erhält, die es benötigt und verwalten soll. In CAP sind die Informationen z.B. zu den Attributen Dauer oder Uhrzeit verfügbar, wohingegen keine weiteren Informationen zum Ort verfügbar sind. Es ist denkbar, daß unter gewissen Umständen ein Raum mehrfach von mehreren CAP-Instanzen belegt werden kann. Andere Softwareassistenten denen es gut stehen würde, solchen Ansprüchen zu genügen, wären ein Newsreader und Email-Programm und daher wird an beidem bereits geforscht.

Um die Leistungsfähigkeit eines Softwareassistenten zu erhöhen, muß es ein weiteres Bestreben sein, die Lernphase zu verkürzen. Denkbar ist, abstraktere Regeln zu lernen, quasi Metaregeln, also Regeln über Regeln, z.B. "Jede Seminarbesprechung in einem Seminar wird immer am gleichen Ort abgehalten.". [14] Dadurch würden sich die Regeln vom Lernkontext abheben und beim ersten Auftreten eines Termins für ein Seminar stünden Informationen zur Verfügung, die nicht erst durch viele ähnliche Termine gelernt werden müßten. Allgemeingültigere Regeln könnten auch auf anderem Wege gebildet werden, wenn man nämlich die Trainingsdatensätze von mehreren Benutzern zusammenfaßt und auf diesem Pool von Daten den Lernalgorithmus anwendet. Dieses Verfahren nennt sich Kooperatives Lernen. Kooperatives Lernen kann die Lernphase deutlich verkürzen.

Ein andersartiger Gesichtspunkt zur Steigerung der Leistungsfähigkeit betrifft die oben erwähnte Verfügbarkeit. Dem Softwareassistent muß der Zugang zu anderen Online-Quellen ermöglicht werden, z.B. zu einer Raumbelegungsdatei, einer Personaldatenbank oder zu anderen Softwareassistenten. Die einzelnen Instanzen eines Softwareassistenten könnten zusammenarbeiten, indem sie Informationen austauschen. In CAP könnte der Raumkonflikt unter anderem dadurch gelöst werden, daß die CAP-Instanzen die Belegung der Räume austauschen. Das ist potentiell ja möglich, da CAP ein Online-Kalender ist, und die Rechner auf denen das System läuft somit vernetzt sind. Durch die Benutzung von Online-Quellen steigt evtl. auch wieder die Anzahl der Attribute und daher besteht das Risiko, den Lernprozeß zu verschlechtern. Also sollte zusätzliche Forschung in diese Richtung mit berücksichtigt werden.

Insgesamt aber hat die Entwicklung von CAP gezeigt, daß es sehr wohl möglich ist, einen lernenden Softwareassistenten zu entwerfen, der dem Benutzer einen Teil seiner Arbeit mit Hilfe des erlernten Wissens abnehmen kann. Die Entwicklung hat die Schwierigkeiten und Möglichkeiten, aber auch die Unzulänglichkeiten eines solchen Systems aufgezeigt und die Ergebnisse werden eine Reihe weiterer lernender Softwareassistenten hervorbringen.

12.5 Der Algorithmus ID3

Ein Entscheidungsbaum stellt eine Reihe von Fragen an ein Objekt oder eine Instanz, um es zu klassifizieren. ID3 erstellt von oben nach unten einen Entscheidungsbaum anhand von Beispielen mit Hilfe des folgenden Algorithmus:

```
Grow-tree(training-samples)
  If Terminate-condition(training-samples)
    Return an appropriately labeled leaf node
  Else
    Let F = Choose-best-feature(training-samples)
    Let N be a decision node testing F
    For each value V of F in training-samples
      Let subsamples = all instances I, such that F(I) = V
      Attach a branch on N to Grow-tree(subsamples)
    Return N
```

Eine häufige Abbruchbedingung beispielsweise ist abzubrechen, wenn alle Instanzen eines subsamples in der gleichen Klasse sind. Wenn der Algorithmus den Baum von oben nach unten aufbaut, muß er immerzu ein neues Attribut für den nächsten Knoten auswählen. Dies erfolgt durch den Versuch die Vieldeutigkeit in einer Trainingsmenge zu verringern, oder anders ausgedrückt die Reinheit innerhalb der Trainingsmenge zu erhöhen. Gemessen wird die Reinheit an der Entropie der entsprechenden Menge. Der Vollständigkeit sei die Entropie hier für den 2-Klassen-Fall angegeben:

$$E(T) = -(c_1/n)log(c_1/n) - (c_2/n)log(c_2/n). \tag{12.1}$$

T ist hierbei das Beispiel mit n Instanzen, c_1 ist die Anzahl der Instanzen, die zu der Klasse 1 gehören und c_2 ist der analoge Wert für die Klasse 2. Gehören zu jeder Klassen gleich viel Instanzen, so ist die Entropie gleich 1 und sie geht gegen Null je mehr sich die Anzahl unterscheidet. Die Entropie eines Trainingsdatensatzes läßt sich aus einer bestimmten Anzahl von Klassen berechnen und das Inverse hiervon gibt die Reinheit an. Der Attributtest aus ID3 teilt eine Beispielsmenge, im Algorithmus training-sample, bzw. subsample genannt, in mehrere Mengen, berechnet die Entropie einer jeden Menge, addiert diese Werte, gewichtet mit dem Kehrwert der Kardinalität der jeweiligen Mengen, auf und subtrahiert diese nicht negative Zahl von der Entropie der Ausgangsmenge. Das Ergebnis nennt sich Informationsgewinn. ID3 wählt zur weiteren Verarbeitung das Attribut mit dem höchsten Informationsgewinn aus.

Literaturverzeichnis

[1] Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott, David Zabowski: Experience With a Learning Personal Assistant, Carnegie Mellon University, 1994

[2] S. Bocionek, M. Sassin: Dialog-based Learning(DBL) for Adaptive Interface Agents and Programming-by-demonstration Systems Tech. Rept. CMU-CS-93-175, Carnegie Mellon University, 1993

[3] R. Caruana, D. Freitag: Greedy Attribute Selection, to appear in the Proceedings of the Eleventh International Conference on Machine Learning

[4] L. Dent, J. Boticario, J. McDermott, T. Mitchell, D. Zabowski: A Personal Learning Apprentice, Proceedings of the International Joint Conference an AI, July 1992

[5] A. Kay: Computer Software, Scientific American 251, 3 (1984),53-59

[6] Y. Kodratoff, G. Tecuci: DISCIPLE: An Interactive Approach to Learning Apprentice Systems, Tech. Rept. UPS-293, Laboratoire de Recherche en Informatique, Universite de Paris-SUD, 1986

[7] R. Kozierok, P. Maes: Intelligent Groupware for Scheduling Meetings, Submittetd to CSCW-92, 1992

[8] T. Mitchell, S. Mahadevan, L. Steinberg: LEAP: A Learning Apprentice for VLSI Design, Proceedings of the Ninth International Joint Conference on Artificial Intelligence, August, 1985

[9] Y. Nakauchi, T. Okada, Y. Anzai: Groupware that Learns, Proceedings of the IEEE Pacific Rim Communications, Computers and Signa Processing, May, 1991

[10] N. Negroponte: The Architecture Machine: Towards a More Human Environment, MIT Press, 1970

[11] J.R. Quinlan: Induction of Decision Trees, Machine Learning 1, 1 (1986), 81-106

[12] J.R. Quinlan: Generating Production Rules from Decision Trees, Proceedings of the International Joint Conference on AI, August, 1987

[13] D. E. Rumelhart, G.E. Hinton, R.J. Williams: Learning Internal Representations by Error Propagation in Parallel Distributed Proccessing, Rumelhart, McClelland and the PDP Research Group, Ed., MIT Press, 1986, pp. 318-362

[14] S. Russel, Analogical and Inductive Reasoning, Ph.D. Th., Computer Science Department, 1986

[15] N. Sadeh: Look-ahead techniques for micro-opportunistic job shop scheduling, Ph.D. Th., Robotics Institute, Carnegie Mellon University, 1991

[16] J.C. Schlimmer: Redstone: A Learning Apprentice of Text Transformations for Cataloging, Unpublished

[17] B. Sheth, P. Maes: Evolving Agents for Personalized Information Filtering, Proceedings of the Ninth IEEE Conference on AI for Applications, 1993

Kapitel 13

Implementierungstechniken für Internet-Agenten

Andreas Lorenz

Übersicht

Bei der Arbeit eines Agenten im Internet kann es leicht zu Verärgerungen kommen – schließlich ist solch ein Agent nicht allein im Netz. Aus diesem Grund gibt es für den Programmierer des Agenten eine Vielzahl von Dingen zu berücksichtigen. Zum Einen unterschiedet sich das Internet von anderen Softwareumgebungen zum Beispiel durch seine starke Dynamik und die extreme Auswirkung von Fehlern. Läuft der eigene Agent „Amok“, so kann dies eine Vielzahl trauriger Benutzer des Internets nach sich ziehen. Durch Agenten verursachter Frust entsteht jedoch nicht nur aufgrund fehlerhafter Implementierungen, sondern auch durch egoistische Arbeitsweise und der daraus resultierenden Blockierung von Ressourcen. Dies ist ein Grund, weshalb Agenten nicht überall willkommen sind und von einigen Servern gebeten werden, fernzubleiben. Aufgabe des Programmierers bleibt es, die hier vorgestellten Konstrukte zur Vermeidung solcher Ärgernisse in seiner Implementation zu verwirklichen.

13.1 Wozu Agenten im Internet?

Die weltweite Vernetzung durch das Internet bietet die Möglichkeit, schnell und kostengünstig auf Informationen zuzugreifen, welche über den ganzen Erdball verteilt sind. Doch die Unmengen an verfügbaren Informationen hat auch ihre Schattenseiten: ein Überblick über das Informationsangebot ist nicht mehr möglich, die Mailbox quillt über und etliche vielversprechende Adressen sind auf Haftnotizen rund um den ganzen Monitor verteilt. Eine gezielte Suche nach Informationen von Hand ist nicht mehr sinnvoll; kaum ein Anwender wird die Zeit aufbringen können, sich von Link zu Link durch's WWW zu „hangeln". Hier kommt der Ruf nach Automatisierung auf den Plan. Und gerade da setzen die meisten Agenten im Internet, vorwiegend als Robot, Wanderer oder Spider bezeichnet, mit ihren Fähigkeiten an. Sie handeln autonom im Auftrag des Anwenders oder auch eines anderen Agenten und sind so selbständig in der Lage, die gewünschten Informationen zu suchen und bereitzustellen. Natürlich sind Agenten fähig, komplexere Aufgaben zu meistern, wie zum Beispiel Flugüberwachung durch gemeinsames Handeln mehrerer Agenten in Multi-Agenten-Systemen oder kaufmännische Tätigkeiten auf elektronischen Märkten durch Anwendung mobiler Agenten. Im Internet werden jedoch die meisten Agenten noch immer für Suche und andere Routineaufgaben, wie Filtern von Mailboxen oder automatische Beantwortung von E-Mail eingesetzt. [6]

13.2 Agentenorientierte Programmierung

Agentenorientierte Programmierung kann als eine spezielle Art von Objektorientierter Programmierung angesehen werden. Agenten werden von Agenten-Programmen kontrolliert. Diese beinhalten auch die Grundlagen der Kommunikation, sowohl mit den Servern als auch mit anderen Agenten. In objektorientierter Sicht haben wir ein System von Modulen, welche in der Lage sind, miteinander zu kommunizieren. Die einzelnen Module behandeln dann die erhaltenen Nachrichten individuell und reagieren darauf. Bei agentenorientierter Programmierung haben wir ein Gerüst, welches einen Zustand festlegt. Dieser Zustand wird als *mental state* bezeichnet. Er wird durch Können und Fähigkeiten des Agenten gegeben. Von ihm aus werden die einzelnen Module, eben die Agenten, kontrolliert. Diese verrichten dann die eigentliche Arbeit. Sie sammeln Informationen, fragen sowohl bei Servern als auch bei anderen Agenten nach, laden und speichern, führen Berechnungen durch und assistieren auf diese Art einem User oder auch einem Artgenossen. Wie in Tabelle 13.1 zu erkennen ist, stimmen jedoch die Funktionsweisen bei beiden Arten der Programmierung überein und lassen so die Betrachtung der agentenorientierten Programmierung als Spezialfall

Tabelle 13.1: OOP vs. AOP

	OOP	AOP
Grund-Einheit	Object	Agent
Parameter zur Definition des Zustandes	keine Vorgaben	Wissen, Fähigkeiten, Auswahlmöglichkeiten
Funktionsweise	Erhalt von Nachrichten	und Aufruf von Methoden
Nachrichtentyp	keine Vorgaben	Informationen Angebote, Rückfragen
Vorraussetzungen an Methoden	keine	Ehrlichkeit, Konsistenz

der Objektorientierten zu. Dies ist ein wichtiger Grund für die Programmierung von Agenten mit Hilfe objektorientierter Konstrukte, wodurch sich gerade im Internet, angetrieben durch die wachsende Verbreitung von JAVA, vielfältige Möglichkeiten ergeben. [10]

13.3 Implementierungskonstrukte für Agenten im Internet

13.3.1 Anforderungen an Robots speziell im Internet

Agenten können in den verschiedensten Programmiersprachen implementiert werden. Doch egal ob PERL, LISP, C, C++ oder andere Sprachen: bestimmte Konstrukte sind allen Agenten-Implementationen gemeinsam.

Zu allererst handelt es sich auch bei Agenten um Software. Daher müssen sie den allgemeinen Anforderungen an Software genügen, wie

- Korrektheit
- Vollständigkeit
- Effizienz und
- Zuverlässigkeit. [9]

Hierbei bleibt jedoch zu beachten, daß es sich in diesem Fall um aktive Software in einer bestimmten Umgebung, dem Internet, handelt. Dies hat zur Folge, daß einige Punkte nicht wie gewohnt behandelt werden können. Als erstes ist hier die Effizienz zu nennen. Normalerweise beinhaltet sie den

Wunsch, Ergebnisse mit geringem Einsatz an Hardwareressourcen sowie an Zeit zu erzielen. Besonders der Zeitfaktor spielt dabei eine sehr sensible Rolle. Denn so nützlich Agenten sein können, so sehr werden sie zu einer Plage, wenn sie auf bereits überlasteten Servern zusätzlich hohen Kommunikationsverkehr auslösen. Sicher kann ein Search-robot hunderte von WWW-Dokumenten pro Minute durchsuchen, doch wird die immens hohe Belastung der Kommunikationswege des WWW-Servers dessen Administrator garantiert verärgern. Deshalb sollte jeder Robot-Programmierer darauf achten, seinen Robot nicht zu hastig zu Werke gehen zu lassen. Dies zu bewerkstelligen ist mit Sicherheit nicht einfach und erfordert vom Programmierer eine gewisse Selbstdisziplin. [4] Doch es gibt noch mehr zu beachten. Zum Beispiel stellt HTTP ein *Accept field* zur Verfügung. In diesem Feld kann spezifiziert werden, welche Art von Daten vom Agenten verarbeitet werden können. Werden nur Textdateien gesucht, so kann mit Hilfe dieses Feldes ein cleverer Server die Übertragung aller andereren Dateien vermeiden. Die Vermeidung unnötiger Kommunikation ist natürlich nicht auf den Server beschränkt. Der Programmierer eines Agenten sollte in seiner Implementation bereits bewerkstelligen, alle Links auf derartige Daten zu ignorieren. Wenn ein Robot nur Textdateien behandeln kann, so besteht keinerlei Notwendigkeit, Dateien in .ps, .zip oder .gif Formaten überhaupt zu beachten. Desweiteren ist es oft sinnvoll, ein Rückschreiten beim Laden von Dokumenten zu vermeiden. Hierbei ist bei HTML-Dokumenten besondere Vorsicht vor Referenzen wie <a href=http://irgendwo/pub> geboten. Viele Seiten unterlassen ein abschließendes „/“ zur Kennzeichnung eines Directories, um Sub-URL's einfacher zusammensetzen zu können. Für einen Agenten, welcher sich an derartigen Referenzen „entlanghangelt“, entsteht jedoch die Gefahr, zu bereits behandelten Ausgangspunkten zurückzugelangen und im schlimmsten Fall in eine Schleife zu geraten. Aus diesem Grund ist es häufig sinnvoll, sich die bereits aufgesuchten Adressen zu merken, um so mehrfachen Zugriff oder gar Endlosschleifen vermeiden zu können.

Mit Hilfe dieser Ratschläge läßt sich der Verkehr auf dem Internet zwar nicht vermeiden, aber doch wenigstens auf das notwendige Minimum verringern. Leider funktioniert dies nur, wenn die Implementierung auch absolut fehlerfrei ist. Besonders wichtig ist dabei, daß der Agent nicht einfach nur läuft, sondern auch wirklich die Aufgaben – und zwar nur diese Aufgaben – erfüllt, wozu er erschaffen wurde. Der Programmierer eines Agenten muß bedenken, daß Fehler in seiner Software eine große Menge von Internet-Benutzern in Mitleidenschaft ziehen. Ein „Amoklaufender“ oder in einer Endlosschleife hängender Agent ist dabei mindestens so gefährlich wie ein Abstürzender, besonders wenn er durch unkontrollierte Kommunikation den Netzbetrieb behindert. Aus diesem Grund sollte der erste Test eines neuen Robots auf dem lokalen Server erfolgen, um negative Effekte wenigstens begrenzen zu können. Nach den ersten Testläufen sollte man sich auch die Mühe machen, die erhaltenen Resultate und die erreichte Performance zu analysieren, und zu beurteilen, ob der Robot ein ausgewogenes Gleichgewicht zwischen gewünschter Effizienz und vertretbarer Performance erzielt. Ist die lokale Testphase erfolgreich abgeschlossen, so soll der entstandene Agent auch auf anderen Servern im Internet aktiv werden. Da es sich hierbei um „intelligente“ Software handelt und nicht etwa um Viren, ist es meist seitens des Programmierers sinnvoll, die Verantwortung für das Tun seines Robots zu übernehmen. Dazu gehört, daß sich sowohl der Agent, als auch sein Programmierer bei den Servern ausweisen. In HTTP stehen dazu zwei Felder zur Verfügung:

1. User-Agent: Hier wird der Name des Agenten eingetragen.
2. From: Der Programmierer gibt in diesem Feld seinen Namen oder besser seine E-Mail Adresse an.

Desweiteren sollte vor dem eigentlichen Ablauf des Robots eine kurze Mitteilung an *comp.infosysteme.www.providers* geschickt werden. Hier ist eine Liste aller aktiven Internet-Agenten verfügbar.
Diese drei Punkte bringen für beide Seiten – Server und Agent – einige Vorteile. Der Server ist in der Lage, die Software zu identifizieren und braucht keine Gegenmaßnahmen zu ergreifen, was wiederum den Ablauf des Agenten nicht behindert. Der Programmierer des Agenten hat außerdem noch den Vorteil, daß er im Falle eines Fehlers oder sonstigen ungewollten Aktionen seiner Software durch den entsprechenden Server-Administrator darüber informiert werden und so seine Software verbessern kann. Das Zusammenspiel von Server und Agent ist also nicht zu unterschätzen. Um die Hilfe des Servers noch steigern zu können, kann ihm mitgeteilt werden, welche Ziele der Agent verfolgt. Hierzu unterstützt HTTP ein *Referer field* , in welchem der Server über Sinn und Zweck informiert werden sollte.

13.3.2 Standard for Robot-Exclusion

Die Beachtung der bisher vorgestellten Tips stellt eine bestmögliche Nutzung des Internet mit Hilfe von Agenten dar. Doch Robots bleiben eine Art zweischneidiges Schwert. Richtig eingesetzt können sie ihren Anwendern eine Menge Arbeit abnehmen und sogar den Netzverkehr reduzieren. Unvorsichtig eingesetzt produzieren sie jedoch unnötigen Datenverkehr, der das Netz belastet und Ressourcen bindet. Gerade Server mit stark begrenzten Ressourcen oder mit begrenzter Bandbreite können dadurch sehr schnell aus dem Gleichgewicht geraten. Auch Server-Betreiber, welche für ihren Betrieb volumenabhängige Tarife zahlen müssen, sehen es lieber, wenn Robots von ihren Servern fernbleiben. Aus diesen Gründen stoßen Robots nicht unbedingt überall auf Gegenliebe. Deshalb haben sich die Mitglieder der *Robot*

Mailing List auf eine Empfehlung zum Ausschluß von Robots geeinigt. Hierbei handelt es sich keinesfalls um einen offiziellen Standard, welcher von irgendjemandem überwacht wird. Es ist also nicht obligatorisch, diesen zu realisieren, doch im Allgemeinen halten sich die meisten Autoren daran – schließlich ist die Umsetzung recht simpel. [7] Dem Standard liegt folgende Methode zugrunde:
Auf jedem Server wird ein lokaler URL /robots.txt erzeugt. Dies ist auch für bereits bestehende Server einfach zu bewerkstelligen und durch die Standardisierung für jeden Robot einfach auffindbar. In diesem URL wird den Agenten dann mitgeteilt, welche Zugriffe nicht erwünscht sind. Ein solches File hat folgendes Aussehen: Es enthält einen oder mehrere Blöcke der Form <field>:<optionalspace><value><optionalspace>.
Bei der Verwendung mehrerer Blöcke werden diese durch eine oder mehrere Leerzeilen voneinander getrennt. Kommentare in diesem File werden genauso wie in UNIX bourne shell mit „♯" gekennzeichnet. Jeder Block beginnt mit einem *User-agent field* , auf welches dann eine oder mehrere *Disallow fields* folgen.

User-Agent: Hier wird der Name des auszuschließenden Agenten angegeben. Falls mehr als ein Robot ausgeschlossen werden soll, so wird dies durch Verwendung mehrerer Blöcke mit den entsprechenden User-agent Feldern zum Ausdruck gebracht. Eine Besonderheit stellt hier das Joker-Zeichen „*" dar. Wie nicht anders zu erwarten gibt es zu verstehen, daß die folgenden Angaben für alle Robots gelten. In darauffolgenden Blöcken können jedoch für bestimmte Agenten Ausnahmen vereinbart werden.

Disallow: Hier werden die Directories festgelegt, welche für oben angegebene Robots gesperrt sind. Auch hier existiert ein Joker-Zeichen „/", welches alle Directories sperrt. Falls keine Directories gesperrt sind, so wird in diesem Feld nichts eingetragen. [5]

Am leichtesten läßt sich das Aussehen der /robots.txt Files an ein paar einfachen Beispielen verdeutlichen:

- ♯ Für Cypermapper ist /bin/ und /cyberworld/map/ gesperrt:
  User-agent: Cypermapper
  Disallow: /bin/
  Disallow: /cyberworld/map/
- ♯ Für alle Robots ist /bin/ und /cyberworld/map/ gesperrt, mit Ausnahme von Cypermapper
  ♯ Zuerst sperren...
  User-agent: *

Disallow: /bin/
Disallow: /cyberworld/map/

♯ ...dann Sondergenehmigung verteilen
User-agent: Cypermapper
Disallow:

- ♯ Für alle Robots sind alle URL's tabu
  User-agent: *
  Disallow: /

- ♯ Für niemanden ist nichts gesperrt
  User-agent: *
  Disallow:

Nachdem der Administrator des Servers seine Arbeit gewissenhaft verrichtet hat, bleibt es jetzt Aufgabe des Agenten-Programmierers, daß /robots.txt File zu laden, auszuwerten und sich daran zu orientieren – allein der Netiquette wegen. [8] Wie dies verwirklicht werden kann, sehen wir später.

13.4 Grundgerüst für Internet-Agenten

In diesem Kapitel möchte ich ein Grundgerüst eines Agenten fürs Internet vorstellen. Dabei werde ich C auf einem UNIX-System benutzen. Da aufgrund des rasant wachsenden Angebots an WWW-Seiten die Behandlung von HTML-Dokumenten immer größere Bedeutung findet, will ich mich hier auf die Übertragung und Bearbeitung solcher Dateien beschränken; die Art und Weise ist auch für andere Dokumente ähnlich. Dieses Gerüst kann also als Vorlage für die verschiedensten Aufgaben dienen, da es die wichtigsten Fähigkeiten eines Robots vorstellt:

1. Aufbau einer TCP/IP-Verbindung. Jeder Agent, welcher sich im Internet bewegt, muß die Möglichkeit der Kommunikation mit Servern besitzen.

2. Verwirklichung des Ausschlußprotokolls.

3. Laden von Dokumenten.

4. „Ausschlachten“ von Dokumenten. Dieser Punkt ist natürlich für den einzelnen Agenten sehr wichtig, da er den eigentlichen Sinn des Programmes darstellt. Ebenso wie die Anwendungsgebiete liegen jedoch auch die Verwirklichungen dieses Punktes weit auseinander.

13.4.1 Aufbau einer TCP/IP-Verbindung

Der Aufbau einer solchen Verbindung stellt die Grundlage der Kommunikation zwischen Agent und Server dar. Eine sehr einfache Möglichkeit dazu bietet die Verwendung von Sockets. Mit ihrer Hilfe kann durch simples Verschicken von Nachrichten kommuniziert werden. In C stehen zu diesem Zweck eine Vielzahl von Funktionen zur Verfügung. [2] Diese befinden sich in der Header-Datei <sys/sockets.h>. Darin enthalten ist auch eine Funktion `int socket (domain, type, protocol)`.
Als Rückgabewert erhalten wir von dieser Funktion einen Socketdescriptor. Da wir uns im Internet befinden, verwenden wir dieses auch als *domain* , duch AF_INET ausgedrückt. Hierbei besteht jede Adresse aus zwei Teilen: Der Hostadresse und einer Portnummer, auch als „transport suffix“ bekannt. Dabei ist zu beachten, daß der Hostname meist als String vorliegt und erst in die entsprechende Internetadresse umgewandelt werden muß. Als *type* wird gern SOCK_STREAM gewählt, da hier die Kommunikation über einen üblichen Stream sehr einfach gestaltet wird. Zum Verbinden des Sockets existiert die Funktion

```
int connect (socket, struct sockaddr_in,
                              sizeof (struct sockaddr_in )).
```

Sie stellt die Verbindung über den Socket her und ist für den aktiven Part unbedingt notwendig.

```
int create_socket (char* host, char* port)
{ int sock;
  struct hostent* hostpointer;
  struct sockaddr_in  address;
  /* Diese Struktur beschreibt eine Adresse in der Internet-
     domain, und wird in <netinet/in.h> definiert./*
```

/* Zuerst müssen wir Hostname, welcher ja bis jetzt als String vorliegt, in verständliche IP-Adresse umwandeln. */

```
  hostpointer=gethostbyname(host);
  if (hostpointer==NULL)
  { printf ("\nFehler in gethostbyname\n");
    return -1; }
```

/* Lösche Adresse */

```
  bzero ((void*) &address, sizeof (struct sockaddr_in));
```

/* übermittle gewünschte Adresse und Port */

```
  bcopy (hostpointer->h_addr, (char*) &address.sin_addr,
                                 hostpointer->h_length);
  address.sin_family = hostpointer->h_addrtype;
  if (port == "")
  { /* Kein Port angegeben, verwende Port 80 - http-Port */
    address.sin_port = htons(80); }
  else
  {/*Umwandlung des String port in int durch Funktion atoi*/
    address.sin_port = htons (atoi (port)); }
```

/* Jetzt ist Domain in address.sin_family, host in address.sin_addr als IP-Adresse und port in address.sin_port eingetragen und dem Socket steht nichts mehr im Wege... */

```
  sock = socket (address.sin_family, SOCK_STREAM, 0);
  if (sock < 0)
  { printf ("\nFehler in socket\n");
    return -1; }
```

/* Bevor wir nun Daten übertragen können, müssen wir unseren Socket noch verbinden */

```
  if (connect (sock, &address, sizeof (address)) < 0)
  { printf ("\nFehler in connect\n");
    return -1; }
```

/* So, jetzt ist ein Socket erstellt und auch verbunden */

```
  return sock;
}
```

13.4.2 Lesen eines Files vom Server

Über diesen Socket erfolgt nun die Kommunikation vorzugsweise mit Hilfe der Funktionen
`int send (socket, message, flag)` zum Senden von Nachrichten und
`int recv (socket, buffer, buf_size, flag)` welche die empfangenen Nachrichten in Buffer ablegt. Um den Netzverkehr so gering wie möglich zu halten, lesen wir das File und speichern es vor der Auswertung lokal ab.

```
int read_file (char* host, char* port, char* filename)
{ int sock;
```

```
   char buffer[256];
   int buf_size = 256;
   FILE* lokales_file;
```

/* Zuerst lokales File für schreibenden Zugriff öffnen */

```
   lokales_file = fopen ("ausgabe.txt", "w");
   if (lokales_file == NULL)
   { printf ("\nFehler in fopen\n");
     return -1; }
```

/* Jetzt Socket erzeugen und Verbindung aufbauen */

```
   sock = create_socket (host, port);
   if (sock < 0)
   { printf ("\nFehler in create_socket\n");
     return -1; }
```

/* GET-Request schicken inclusive Ausweis des Robots */

```
   sprint (buffer, "GET %s\n", filename);
   if (send (sock, buffer, strlen (buffer), 0) < 0)
   { printf ("\nFehler in send\n");
     return -1; }
   if (send (sock, "User-Agent: lsaRobot\n\n",
         strlen ("User-Agent: lsaRobot\n\n"), 0) < 0)
   { printf ("\nFehler beim Ausweisen\n"); }
   bzero (buffer, buf_size);
```

/* Lese Antwort des Servers, solange dieser Daten zurückgibt */

```
   while (recv (sock, buffer, buf_size-3, 0) > 0)
   { /* Schreibe Antwort in lokales File */
     fprintf (lokales_file, "%s", buffer);
     bzero (buffer, buf_size); }
   /* Schliesse lokales_file */
   fclose (lokales_file);
   close (sock);
   return 0;
}
```

13.4.3 Implementation des Standard for Robot Exklusion

Hierzu ist es notwendig, das /robots.txt File auf dem verbundenen Server zu laden und auszuwerten. Es ist zu beachten, daß es sich hierbei um kein HTML-Dokument handelt, sondern um ein reines Textfile. Dies hat zum Glück jedoch keine Folgen für den Aufbau der Kommunikation, da streng genommen auch HTML-Dokumente besondere Textdateien sind. Ansonsten müßten wir für die Realisierung des Robotausschlußprotokolls und die eigentliche Auswertung der URL's zwei verschiedene Kommunikationswege verwenden. So kann jedoch obige Funktion sowohl zum Laden eines HTML-Dokuments als auch zum Lesen des /robots.txt Files genutzt werden. Allerdings hat dies Auswirkung auf die Auswertung des Files, was in verschieden Quellen nicht berücksichtigt wurde. [7] Um den Rahmen nicht zu sprengen, lassen wir den gesamten Host außen vor, falls auf ihm wenigstens ein Pfad für alle oder speziell für uns gesperrt ist. Auch der Fall, daß speziell für unseren Robot Exklusivrechte vergeben werden, würde hier zu weit gehen.

```
int robot_exklusion (char* host, char* port)
{ int sock;
   FILE* lokales_file;
   char buffer[256];
   char* tmp;
```

/* Zuerst /robots.txt lesen */

```
   if (read_file (host, port, "/robots.txt") < 0)
   { printf ("\nFehler in read_file\n");
     return -1; }
```

/* Öffnen des lokalen Files, in welchem die Informationen zwischengespeichert sind */

```
   lokales_file = fopen ("ausgabe.txt", "r");
   if (lokales_file == NULL)
   { printf ("\nFehler in fopen\n");
     return -1; }
```

/* Einlesen und Auswerten des Files */

```
  do
  { strset (buffer, '\0');
    fgets (buffer, 256, lokal);
    /* Suche User-agent - Eintrag */
```

```
    if ((tmp=strstr (buffer, "User-agent:")) != NULL)
    { /* Ist User-agent lsaRobot oder * ? */
      if ((strstr (tmp, "*") != NULL) ||
                     (strstr (tmp, "lsaRobot") != NULL))
      { /* Suche Disallow - Eintrag */
        while ((tmp=strstr(buffer, "Disallow:")) == NULL)
        { fgets (buffer, 256, lokal); }
        /* Ist irgendein Pfad gesperrt  ? */
        if (strstr (tmp, "/") != NULL)
        { /* Bitte keinen Zugriff...*/
          printf ("gesperrt");
  return -1;
    } } }
  } while (buffer[0] != '\0');
  fclose (lokal);

  /* Zugriff erteilt */
  return 1;
}
```

Mit den vorgestellten Funktionen haben wir nun ein einfaches Gerüst für einen Internet-Agenten. Dieser verrichtet bis jetzt noch keine sinnvolle Aufgabe. Doch läßt er sich leicht zu einem „sinnvollen“ Agenten umwandeln. Soll er z.B. wie oben angedeutet HTML-Documente verarbeiten, so ist lediglich noch deren Auswertung hinzuzufügen. Diese Auswertung erfolgt theortisch analog zur Auswertung des /robots.txt Files. Zusätzlich zu den vorgestellten Funktionen benötigen wir noch ein Hauptprogramm. Hierbei ist besonderes Augenmerk auf die Headerdateien zu legen, da in ihnen die eigentliche Arbeit steckt.

/* Hauptprogramm */

```
#include <stdio,h>
#include <unistd.h>
#include <string.h>          /*  strstr (), strset ()  */
#include <sys/types.h>       /*  diverse Typen         */
#include <sys/socket.h>      /*  socket (), connect () */
#include <netinet/in.h>      /*  Internet-Strukturen   */
#includce <netdb.h>          /*  gethostbyname ()      */

int main (int argc, char** argv)
{ if (argc != 3)
   { printf "\nUsage: host port filename\n");
     return -1; }
```

```
   if (robot_exclusion (host, port) < 0)
   { printf "\nRobot ist nicht willkommen\n");
     return -1; }
```

/* Hier folgt jetzt die eigentliche Aufgabe, zum Beispiel Laden eines HTML-Documents */

```
   if (read_file (host, port, filename) < 0)
   { printf "\nFehler in read_file\n");
     return -1; }

   return 0;
}
```

13.5 Mobile Agenten fürs Internet

Bisher haben wir ausschließlich Agenten betrachtet, welche nicht über das Netz verschickt werden, sondern von einem Server quer durch das gesamte Internet agieren. Dabei entstehen weniger Probleme, da die Kommunikation mit anderen Servern oder das Laden von Informationen über Protokolle wie TCP/IP und HTTP standardisiert ist. Die Architektur des Servers, von welchem er momentan seine Informationen bezieht, kann dem Agenten bei seiner Arbeit relativ egal sein. Im Gegensatz dazu werden mobile Agenten auf andere Server verschickt und sollen dort, unabhängig von der Architektur, stabil arbeiten. Aus diesem Grund arbeiten mobile Agenten im Internet auf maschinenunabhängigen Plattformen. Eine der wichtigsten stellt die Telescript-Technologie dar. Hierbei wird, ähnlich zum SmallTalk- und JAVA-Konzept, auf der Maschine eine Plattform zur Verfügung gestellt, welche die entsprechenden Konstrukte interpretiert und dadurch von der Architektur der Maschine abkoppelt. Ein weiterer Vorteil dieser Technik ist, daß nach der Übertragung des mobilen Agenten keine permanente Kommunikationsverbindung mehr notwendig ist, da der Agent ab jetzt autonom seinen Job erledigt, ohne weiter extern gesteuert werden zu müssen. Dadurch wird trotz der geringen Bandbreite gerade mobiler Kommunikationsverbindungen eine billige und intelligente Kommunikation mit dem Netz möglich. Allerdings gibt es auch einen entscheidenden Nachteil: solche mobilen Agenten stellen offensichtlich erhöhte Anforderungen an die Sicherheit des Systems. [6] Deshalb werden in Telescript eine Reihe von Sicherheitskonzepten verwirklicht. [1] Desweiteren entstehen erhöhte Anforderungen an

1. Flexibiltät, also die Anpassung an die dynamische Umgebung und

2. Portabiltät, also der Fähigkeit, auf andere Server verschickt zu werden und dort auch tatsächlich lauffähig zu sein. [3]

Zur Verwirklichung dieser Punkte eignen sich objektorientierte Konstrukte besonders gut. So handelt es sich auch bei der Telescript-Technik um einen solchen Ansatz. Ob und wie objektorientierte Ansätze, speziell JAVA, in Zukunft neue Impulse vermitteln können, wird sich zeigen müssen.

Literaturverzeichnis

[1] *An Introduction to Safety and Security in Telescript*, 1995. http://www.genmagic.com/Telescript/TDE/security.html.

[2] L. Besaw. *Berkeley UNIX System Calls and Interprocess Communication*, 1991.

[3] R. Eppler. *Objektorientierte Programmierung im Internet.* Scriptum, Universität Kaiserslautern, 1996.

[4] M. Koster. *Guidelines for Robot Writers.* http://info.webcrawler.com/mak/projects/robots/guidelines.html.

[5] M. Koster. *A Standard for Robot Exclusion.* http://info.webcrawler.com/mak/projects/robots/norobots.html.

[6] P. Stenz M. Römer, B. Quendt. Autopiloten fürs Netz. *c't Magazin für Computertechnik*, März 1996.

[7] J. Weiß R. Hüskens. Roboter-Baukasten. *c't Magazin für Computertechnik*, März 1996.

[8] C. Reiser. *Die klassische Netiquette.* http://www.ping.at/guides/netmayer/.

[9] D. Riecken. Intelligent agents: Introduction to special issue. *Communications of the ACM*, April 1994.

[10] Y. Shoham. Agent-oriented programming. *Artificial Intelligence 60*, 1993.