Datenbanksysteme und WorldWideWeb

Birgitta König-Ries, Jutta A. Mülle, Bethina Schmitt (Hrsg.)

Seminar im Sommersemester 1997
am Institut für Programmstrukturen und Datenorganisation
Fakultät für Informatik
Universität Karlsruhe

Juli 1997

Vorwort

Die Informationsflut ist durch die Fortschritte in der weltweiten Vernetzung und durch technologisch immer leistungsfähigere und auch kostengünstigere Rechner ein Thema, von dem Firmen bis hin zu Privathaushalten immer stärker betroffen sind. Der Zugang zu vielen großen Datenbeständen wird im Internet über sogenannte WorldWideWeb-Server geregelt. Das Ziel des Seminars, im Rahmen dessen die hier vorliegenden Berichte entstanden, war es, einen Überblick über Methoden aus der Datenbanktechnologie zum Zugang zu den enormen Informationsmengen im WorldWideWeb als auch die Gestaltung von Schnittstellen zu Datenbasen und von Datenbanksystemen selbst, die im Internet verfügbar sein sollen, zusammenzustellen.

Als Einführung wird ein historischer Überblick über das noch recht junge Internet, die eigentliche Grundlage der globalen Informationsvernetzung, gegeben. Im darauf folgenden Teil wird mit dem Caching im WorldWideWeb eine grundlegende Verarbeitungstechnik vorgestellt.

Techniken des Information Retrieval, die ursprünglich in Literaturverwaltungssystemen intensiv erforscht und entwickelt wurden, bieten naheliegenderweise ebenfalls Grundlagen für den Umgang mit den häufig semi-strukturierten Informationen im Internet. Nach einer Einfürung in entsprechende klassische Techniken wird deren Einsatz im WWW im Rahmen von Suchmaschinen vorgestellt. Intelligente Suche ist eine Grundvoraussetzung, um in Zukunft überhaupt noch eine Chance zu haben, sinnvoll mit der exponentiell wachsenden Menge an weltweit verfügbaren Informationen umgehen zu können. Die Nutzung von Datenbanktechniken, die auf die Verwaltung gut strukturierter Daten ausgerichtet sind, bieten ebenfalls Einsatzmöglichkeiten zur Erstellung von Information Retrieval Systemen und damit zum Umgang mit semi-strukturierten Daten, wie sie im WorldWideWeb die Regel sind.

Die nächsten Beiträge behandeln Anfragesprachen und entsprechende Datenmodelle für das WorldWideWeb anhand der Systeme Lore, einem Forschungsprototypen, sowie den Systemen WebSQL und W3QL.

Die Möglichkeiten, um relationale Datenbanksysteme in das WorldWideWeb zu integrieren und damit über WWW-Schnittstellen Zugang auf verteilte Datenbasen zu realisieren, ist Thema der folgenden Beiträge. Die momentan wichtigste Entwicklung bei der Programmierung von WWW-Anwendungen ist in der Programmiersprache Java zu sehen, deren Erweiterung als Anwendungsprogrammierinterface (API) für Datenbanksysteme ebenfalls vorgestellt wird.

Zum Abschlußwird ein konkretes Anwendungsbeispiel für verteilte Informationsverwaltung im WorldWideWeb anhand des Bereichs der Umweltinformationssysteme präsentiert.

König-Ries
Jutta Mülle
Bethina Schmitt

Inhaltsverzeichnis

Kapitel 1

Einführung in das Internet

Thorsten Macherey

Kurzfassung *Ein Abriß der Geschichte des Internets von 1968 bis heute, sowie kurze Ausführungen zu seinen informationstechnischen und sozialen Auswirkungen.*

1.1 Einleitung

Fragt man heute Computerbesitzer, was sie unter dem Internet verstehen, werden sie wohl Stichworte wie weltweites Computernetz, Multimedia, Browser und Links anführen. Das sog. World Wide Web (WWW) dominiert heute das Internet. Mit Hilfe verknüpfter Dokumente ist man in der Lage, beliebige Informationen aus dem Internet zu lesen. Doch entstand diese Technik, auf die weiter unten noch eingegangen wird, erst 1991; das Internet hingegen besteht seit 1968. Was also ist das Internet?
Am 24. Oktober 1995 veröffentlichte das Federal Network Council (FNC), eine amerikanische Bundesbehörde, folgende Definition:

Internet bezieht sich auf ein globales Informationssystem, welches

- logisch verbunden ist, durch Nutzung weltweit eindeutiger Adressen gemäß dem Internet Protocol (IP) bzw. dessen Ergänzungen/Nachfolgern gemäß.
- in der Lage ist, Kommunikation gemäß dem Transmission Control Protocol/Internet Protocol (TCP/IP) bzw. dessen Ergänzungen/Nachfolgern zu unterstützen.
- höhere Dienste, die auf der oben genannten Kommunikation bzw. Infrastruktur aufbauen, entweder öffentlich oder privat zur Verfügung stellt, nutzt oder zugänglich macht.

Leider hat diese Definition den Nachteil, den zu definierenden Begriff selbst zur Beschreibung zu nutzen. Das Internet läßt sich nicht so leicht fassen und kann am Besten durch die Geschichte des größten, die Welt umspannenden und sogar in den Weltraum reichenden Computernetzwerkes dargestellt werden.

1.2 Die Visionen - die Anfänge

1945, die ersten Atombomben werden gezündet und katapultieren die USA an die Spitze der Hochtechnologienationen, sie fühlen sich als Vorreiter der Wissenschaften. Die ersten Computer erlauben es, hochkomplexe Berechnungen in nie gekannter Geschwindigkeit zu lösen und damit ehrgeizige Projekte zu realisieren.
Daher führte 1957 der Abschuß des ersten künstlichen Sateliten - des Sputnik durch die UdSSR - zu einer traumatischen Ernüchterung der USA und ihrer westlichen Alliierten. Als Reaktion wurde die (später Defense) Advanced Research Projects Agency (DARPA) von den Vereinigten Staaten ins Leben gerufen. Ihr einziges Ziel sollte es sein, den USA durch Konzentration nationaler Forschungseinrichtungen und deren gezielten Förderung, die Führung in Wissenschaft und Technik, auf militärisch nutzbare Projekte bezogen, zurückzuerobern. Hier wurde und wird mit hohem finanziellen Aufwand interdisziplinäre Forschung unterstützt und die Ergebnisse verschiedenster Projekte zu neuen Erkenntnissen/Technologien zusammengefaßt.

Mitte der 50er und zu Anfang der 60er Jahre begann sich eine neue Wissenschaft zu bilden, die Informatik bzw. Computertechnologie. 1961 veröffentlichte J. Licklider vom MIT visionäre Artikel über ein "galaktisches Netzwerk", die sich mit den sozialen Aspekten des weltweiten Einsatzes von Computernetzwerken und deren Verknüpfung beschäftigten. Folgerichtig holte sich die DARPA Licklider als Vordenker und ersten Leiter ihres neuen Computerforschungsprogramms, das ab Oktober 1962 seine Aufgaben suchte und aufnahm. Man darf nicht vergessen, daß bisher noch keine einzige Fernverbindung zwischen zwei Computern bestanden hatte! Und es sollte selbst mit den vereinten Kräften bei DARPA noch drei Jahre dauern, bis es mit Hilfe einer normalen Telefonverbindung gelang, Rechner miteinander kommunizieren zu lassen.

Die erste "Fern"-Vernetzung zweier Rechner 1965 zeigte sehr schnell, daß ein zu dieser Zeit von drei Gruppen entwickeltes Konzept zur Grundlage jedweder Kommunikation entfernt betriebener Rechner gemacht werden mußte: Paketvermittlung. Paul Baran von RAND, eine der amerikanischen Luftwaffe unterstehenden Forschungseinrichtung, definierte diese Art der Kommunikation: "Paketvermittlung bedeutet das Herunterbrechen der Daten in kleine Datagramme , welche derart markiert werden, daß sich Herkunft und Ziel der Information erkennen lassen, und das Weiterleiten dieser Pakete von einem Rechner zum anderen, bis die Information ihr endgültiges Ziel erreicht. Dies ist entscheidend bei der Entwicklung eines Netzwerkes. Sollten Pakete verloren gehen, können sie nochmals versendet werden, auch über andere Rechner."
Diese Arbeit von Baran, die eine Informationsbewahrung im Netz auch nach massiven Verlusten einzelner Knotenrechnern garantieren sollte, führte zu dem sich bis heute hartknäckig haltenden Gerücht, die USA wollten mit dem Internet ein Computernetzwerk aufbauen, das selbst nach einem Atomschlag die korrekte Weitergabe von Befehlen und

Informationen sicherstellen sollte. Daß insbesondere spätere Arbeiten hohes Gewicht auf Robustheit und Toleranz gegenüber fehlerhafter Infrastruktur legten, erklärt sich aus den zu jener Zeit vorhandenen mangelhaften Verbindungen, nämlich Telefonkabel bzw. bis einschließlich 1980 lediglich Datenleitungen mit einer maximalen Übertragungsleistung von 50 Kbps.

1.3 Vom ARPANET zum Internet

Nachdem man nun Erfahrung in der telefonischen Verbindung zweier Rechner gesammelt hatte, konnte man 1967 endlich einen Plan für das sog. ARPANET veröffentlichen. Er sah den Aufbau des ersten Computernetzes vor, legte die Paketvermittlung als Kommunikationsprotokoll fest und gab Implementierungsdetails der Vernetzung vor.
In den folgenden Jahren konnte der erste Paketvermittler (IMP) fertig- und somit der erste Host in Dienst gestellt werden. Das Internet nahm also im September 1969 mit nur einem Knoten seinen Betrieb auf, was keine spannende Kommunikation erlaubte. Erst einen Monat später gelang es, einen zweiten Knoten fertigzustellen und erstmals ein wirkliches WAN (Wide Area Network) zu installieren. Ende 1969 waren bereits vier Hostrechner permanent ans ARPANET angeschloßen. Man konnte nun auf fremden Rechnern Programme ausführen lassen, also deren Kapazität anzapfen, und die Ergebnisse auf dem eigenen Rechner weiterverwenden. Allerdings mußte jede dieser Anwendungen "zu Fuß" implementiert werden. Dennoch sollte sich dieses Netz innerhalb weniger Jahre zum heutigen Internet auswachsen.
1972 wurde E-Mail, die elektronische Post, im ARPANET implementiert und eingesetzt. Damit hatte das erste WAN für knapp ein Jahrzehnt seine Hauptaufgabe gefunden: den Austausch von Information zwischen beliebigen Personen an beliebigen Orten zu vereinfachen und überhaupt erst praktikabel zu machen.
Als nächster netzweiter Dienst ist FTP (File Transfer Protocol) zu nennen. Er erlaubt das Laden und Versenden von Dateien zwischen Rechnern. Und Telnet schließlich erlaubt das Ausführen von Programmen auf entfernten Rechnern bei gleichzeitiger Nutzung und Anzeige der Ergebnisse auf dem eigenen Rechner.

Eines der grundsätzlichen Probleme des ARPANET war aber die komplizierte Anbindung neuer Hostrechner. Es mußte jedesmal eine komplett neue Implementierung der Kommunikationsprotokolle sowie aller Dienste vorgenommen werden; eine sehr mühselige Arbeit angesichts der Assemblerprogrammierung. Außerdem waren die gefahrenen Protokolle immer noch von der eingesetzten Infrastruktur abhängig.
Um dies zu umgehen, wurde das Konzept des Internet entwickelt. Es basiert auf der Idee der offenen Kommunikation. Hier spielen Bauart und interne Details der anzuschließenden Rechner und Netze keine Rolle und werden als Black Box hingenommen. Genausowenig sollte die genutzte Netzinfrastruktur von Bedeutung sein.
All dies führte zur Definition des Transport Control Protocol, das sich an vier Grundregeln zu orientieren hatte:

- Jedes Netzwerk steht für sich alleine und zur Anbindung dieses Netzes ans Internet dürfen keine internen Änderungen verlangt werden.

- Die Kommunikation erfolgt nach dem Prinzip des größten Erfolges; d.h. sollte ein Paket das Ziel nicht erreichen, so wird es nochmals von der Quelle gesendet.

- Black Boxes (später Gateways und Router) sollen die Verbindung zwischen einzelnen Netzen übernehmen.

- Es existiert keine globale Kontrolle des Internets auf Betriebsebene!

Zur gleichen Zeit begann die Entwicklung einer neuen Technologie: Ethernet. Sie erlaubte die lokale Verbindung von Rechnern mit Hilfe von Koaxialkabeln. Damit war zum ersten Mal der Aufbau schneller lokaler Netzwerke möglich. TCP/IP ermöglichte die Einbindung dieser LANs ins Internet, so daß ab 1976 die Zahl der Hosts schnell anwuchs.
TCP/IP ermöglichte darüberhinaus die Entwicklung neuer, offener und noch vom ARPANET getrennter Netze. Offen meint hier, daß jeder Rechner, der das neue Protokoll fuhr, in der Lage war, sich in diese Netze einzubinden, natürlich nur insofern er Zugangsberechtigung hatte. Als wichtige Beispiele sind hier USENET (user group network) und BITNET, ein erster Mail- und File-Server, zu nennen.
Ein weiterer Meilenstein in der Größenentwicklung des Internets war konsequenterweise 1983 die Umstellung des kompletten ARPANET auf TCP/IP. Damit war die Verbindung aller bisher existierenden Netze zu einem globalen Netzwerk möglich. Das Internet war endlich das, was sein Name impliziert: ein weltweites offenes Netz, das keine Unterscheidung mehr zwischen den einzelnen, in ihm aufgegangenen Netzen macht. Jeder Teilnehmer konnte nun direkt mit jedem anderen kommunizieren. Diese Vielzahl direkter Kommunikationsmöglichkeiten machte die Entwicklung und den Aufbau leistungsfähiger Backbones notwendig, um die daraus resultierende Datenflut überhaupt bewältigen zu können. 1985 verfügte man bereits über Backbones mit einer Bandbreite von 1.5 Mbps, eine enorme Steigerung zu den ehemaligen 56 Kbps. Jetzt zeigte sich die Stärke von TCP/IP. Da es so implementiert wurde, daß keine Annahmen über die zugrunde liegende Netzinfrastruktur gemacht wurden, konnten die neuen Leitungen einfach parallel zu den alten betrieben werden und diese nach und nach ersetzt werden, so daß auch heute noch eine heterogene Infrastruktur das Internet prägt. Je nach Stand der Technik und Finanzmitteln des Betreibers können Backbones Stück für Stück ausgebaut werden.

1.4 WWW - Entwicklung der Informationsgesellschaft

Die Entwicklung leistungsfähigerer Fernverbindungen stieß wiederum die Entwicklung immer komplexerer Netzanwendungen an. Die typische Entwicklungsschraube der Informatik hatte endlich auch das Internet erfaßt. 1990 existierten bereits 617 000 Hosts im Netz und zum Ende dieses Jahres veröffentlichte das europäische CERN in der Schweiz ein neues Dokumentformat, das sog. HTML (Hypertext Markup Language). Es ermöglichte die direkte Verknüpfung von Informationen über Dokumentgrenzen hinweg. Darauf basierend wurde

das Konzept für das World Wide Web aufgestellt; es erlaubt die Verknüpfung beliebiger Daten sowohl über Dokument- als auch Rechnergrenzen hinweg. Mitte 1991 wurden erste zeilenorientierte Editoren vorgestellt, die das neue Format verarbeiten konnten; die Browser waren entwickelt, mit deren Hilfe man nun durch eine Dokumentenstruktur schmökern (neudeutsch: browsen) konnte.
Die neue Technik stieß jedoch noch nicht auf große Aufmerksamkeit. Selbst die Entwicklung komfortabler und grafikfähiger Browser, wie z.B. Mosaic, sollte noch nicht den Durchbruch bringen.
1993 riefen die europäische Kommission, das Fraunhofer Institut und das CERN das erste Web-basierte Projekt ins Leben, WISE. Dieses Projekt ermöglichte die Verbreitung von Information in benachteiligte Gebiete Europas mit Hilfe des WWW. Damit nahmen die Medien erstmals Notiz von dieser neuen Technik. Artikel erschienen in der ”Times”, im ”Guardian” und im ”Economist”, die das Internet und die neue Technologie des WWW analysierten. In diesem Aufwind organisierte das CERN die erste internationale Konferenz über das WWW und erreichte damit im Mai 1994 erstmals ein breites Fachpublikum. Das Interesse übertraf bereits jetzt alle Erwartungen, von 800 Angereisten konnten nur 400 als Zuhörer eingelassen werden. Das Wort vom ”Woodstock des Web” machte die Runde. Die Programmierer des ersten wirklichen Browsers, Mosaic, gingen nach Amerika und gründeten Netscape. Mit Hilfe ihrer Software lassen sich neue Web-Server einfach implementieren; die Zahl der WWW-fähigen Server stieg damit auf über 1500 an.

Ab jetzt überschlugen sich die Ereignisse:

Juni 1994 Martin Bangemann, Europarat, berichtet vor der europäischen Kommission über Planungen, einen europäischen Daten-Superhighway einzurichten, der die Verbreitung und Anwendung der WWW-Technologie unterstützen soll.

Juli Bangemann veröffentlicht ein Abkommen zwischen dem MIT und CERN, eine WWW Organisation (W3C) aufzubauen, die eine internationale Standardisierung und Verwaltung überwachen soll. Dies hat ein ungeheures Presseecho zur Folge; das renommierte Wall Street Journal veröffentlicht detaillierte Analysen und prognostiziert einen millardenschweren Investitionsmarkt.

August Aufgrund des hohen Interesses wird das ”International WWW Conference Committee” (IW3C2) gegründet. Seine einzige Aufgabe besteht darin, neue Konferenzen zum Thema WWW vorzubereiten.

September Die Europäische Kommission und das CERN verkünden das WebCore Projekt. Es soll die Kerntechnologien für das WWW entwickeln und europaweit installieren.

Oktober Das zweite Web-Woodstock: die internationale Konferenz zu ”Mosaic und das

Web" in Chicago ist genauso überlaufen wie die erste in der Schweiz; von 2000 Anmeldungen können nur 1300 zugelassen werden.

Dezember Das W3C tritt zum erstenmal am MIT zusammen und setzt sich kurz darauf in Brüssel mit Vertretern der europäischen Wirtschaft und der europäischen Kommission zusammen. Das CERN gibt aus Geldmangel die weitere Entwicklung am Web auf. Seine Aufgaben, insbesondere die Arbeiten am WebCore Projekt, werden dem französischem INRIA (Institut National pour la Recherche en Informatique et Automatique) überantwortet.

Februar 1995 Das Web wird zum Hauptthema des G7-Treffens in Brüssel. Hier spricht man von der Informationsgesellschaft, welche die industrielle Gesellschaft in absehbarer Zeit ablösen werde.

April Die dritte Internationale WWW-Konferenz wird von der Fraunhofer Gesellschaft in Darmstadt ausgerichtet. Ihr Thema lautet "Werkzeuge und Anwendungen"; neben Mosaic und Netscape betreten die Software-Riesen die Webbühne: Microsoft, IBM etc. entwickeln Browser bzw. bauen den lizensierten Netscape-Bowser in ihre Systeme ein.

Nicht nur die Industrie, sondern auch der Handel erkennt das immense Web-Potential. So bietet Pizza Hut die Möglichkeit, auf ihrer Web-Seite Pizza zu bestellen. Die "First Virtual", die erste WWW-Bank, öffnet ihre "Pforten". Der Begriff 'Cyber-' nimmt Gestalt an. "First Virtual" ist eine Cyberbank, d.h. ihr entspricht keine physische Repräsentation mehr, sie existiert nur virtuell!
Neben dieser WWW-Hektik entwickelte sich allerdings auch das ursprüngliche Internet erheblich weiter. Mit Einführung der Asynchronous Transmission Mode Technologie (ATM) 1994 standen nun Fernverbindungen mit Kapazitäten von 145 Mbps zur Verfügung. Die Zahl der Hosts erhöht sich auf über 3 800 000.
Durch diesen Boom wird das Internet kommerzialisiert. Die bisher öffentlichen Träger der Netzinfrastruktur können die notwendigen Mittel, zur Versorgung so vieler Rechner, nicht mehr aufbringen. Es bilden sich sog. Provider, die gegen Entgelt industriellen und privaten Interessenten Zugang zum Internet gewähren.

Ende 1995 sind weit über 6 Mio. Hosts im Internet verbunden, ein Jahr später bereits über 15 Mio. Der größte Teil des Datenverkehrs wird ab Anfang 1996 über Backbones privater Investoren abgewickelt, die jetzt Mitte 1997 bereits über 30 Mio. Rechner im Internet versorgen und ein Ende dieser exponentiellen Zunahme der Hosts ist nicht abzusehen.

1.5 Informationstechnische Auswirkungen

Das Internet hat unseren Umgang mit Information, Datenerhebung und -vermittlung wesentlich beeinflußt. Mit Hilfe von Telnet ist es möglich, die Kapazitäten und Ressourcen entfernter Rechner zu nutzen. Mit solchen Rechnerverbünden ist es prinzipiell möglich,

eigene Leistung zu potenzieren. So gelang es erst vor kurzem, knapp 14 000 Rechner zusammenzuschalten, um das im Internet benutzte DES Codiersystem in nur vier Monaten zu knacken; hierzu mußten maximal 72 Billarden Schlüssel abgeglichen werden. Frei nach Goethes Zauberlehrling, durch seine Größe und der TCP/IP-Forderung keinerlei globale Kontrolle auf Betriebsebene durchzuführen, sind das Internet und die darauf ablaufenden Programme jeglicher Beschränkung enthoben. Es ist aber ein Paradebeispiel für die ungeheure Potenz, die in der Ressourcenteilung liegt, und die u.U. mit Methoden der massiven parallelen Systeme noch verbessert und effizienter genutzt werden könnte.
Dank der offenen Architektur des Internets, die es erlaubt, beliebige Rechnersysteme und Netze miteinander zu verknüpfen, ist es für den einzelnen nun genauso möglich, auf Systeme zuzugreifen, die vorher nicht finanzierbar waren. Rechenleistung läßt sich damit nach Bedarf und Mitteln einkaufen bzw. Großsysteme müssen nicht mehr durch einzelne Unternehmen oder Projekte finanziert werden.

Mit der elektronischen Post, E-Mail, öffnete sich ein komplett neuer Weg, miteinander zu kommunizieren. Theoretisch ist es möglich, innerhalb von Sekunden Megabyte von Daten einmal rund um die Welt zu schicken. Erweiterungen dieser Idee sind z.B. Chat-Anwendungen, die eine "Unterredung" in Echtzeit erlauben. Hierbei können beliebig viele Anwender auf einem sog. Kanal miteinander in Kontakt treten, unabhängig von ihrer räumlichen Entfernung. Damit wird auch deutlich, warum in unserer Zeit vom globalen Dorf geredet wird. Entfernungen spielen in der Rechner-Kommunikation keine Rolle mehr. Darüber hinaus entwickelt sich z.Zt. die Internet-Telefonie. Man kann nun zum Ortstarif einmal rund um die Welt telefonieren. Es existieren Anwendungen, die es erlauben komplette Videokonferenzen über das Internet abzuwickeln, Bildschirminhalte zu übermitteln usw.

Durch all diese Kommunikationswege ist es prinzipiell möglich geworden, Produktionsstätten ferngesteuert zu überwachen, Forschung und Entwicklung zu verteilen und zu dezentralisieren. So betreibt z.B. Mercedes Benz ein Designbüro in Kalifornien, das den Entwürfen der in Deutschland ansässigen Fahrzeugkonstrukteure den letzten ästhetischen Schliff gibt. Große Software Engineering Projekte können nun im Internet verteilt bearbeitet werden. Es ist möglich, Arbeitsgruppen nach Bedarf weltweit zusammenzustellen und somit Dienstleistungskapazitäten global zu vermarkten. Existiert in einem Teil der Welt eine Problemlösung, kann sie ohne weiteres ausgetauscht und übernommen werden. Wie bei der persönlichen Kommunikation durch das Netz spielt es auch hier keine Rolle, ob das Büro meines Partners nebenan oder auf der anderen Seite des Globus liegt.

Die wohl grundlegendste Änderung im Umgang mit Information bewirkte das WWW. Mit seiner Hilfe bzw. mit HTML läßt sich Information strukturiert verknüpfen. Hierbei kann über Dokument- und Rechnergrenzen hinweg referenziert werden. Daten werden multimedial dargestellt, d.h. Texte mit Ton oder kompletten Video-Animationen unterlegt. All dies entspricht der Lernstruktur unseres eigenen Gehirns besser als jede andere Form der Informationsdarstellung zuvor. Nicht nur, daß verschiedenste Datenarten (Text, Ton und Video) in einem Dokument Eingang finden, es können auch verschiedene Quellen dieses Dokument bilden. Dadurch kommt es zu einer Globalisierung der Information. Alles was irgendwo im Netz publiziert wird, ist prinzipiell sofort an jedem anderen Ort einsehbar.

Es kann sich dabei um automatisch erfaßte und damit relativ verläßliche Daten oder um manuell erstellte handeln, ihre Wirkung ist jedenfalls global und unmittelbar.
Dank der sich bislang gegenseitig vorantreibenden Entwicklungen sowohl der Netzwerktechnologien als auch der Anwendungen blieb der schon oft prognostizierte Datenstau durch Informationsüberfluß im Internet aus. Natürlich kommt es zu Geschwindigkeitseinbußen bei der enormen Zahl von Teilnehmern weltweit. Der völlige Zusammenbruch, wie er besonders im Vorfeld der Internet-Telefonie erwartet wurde, blieb jedoch aus. Das Internet und das ihm zugrunde liegende Protokoll TCP/IP ist offensichtlich flexibel genug. Ganz anders im Bereich des deutschen BTX. Hier kann das System durchaus Komplettausfälle erleiden, z.B. durch übermäßiges E-Mail Aufkommen.

Ein ganz anderes Problem ergibt sich mit der Fülle der Information für den Anwender. Er muß Mittel und Wege finden, die von ihm benötigte Information aus dem globalen Datenpool herauszufiltern. Hat man einen Einstiegspunkt gefunden, kann es passieren, daß man sich in zu feinen Verzweigungen des Internets versteigt. Hierfür bietet das WWW bzw. spezielle Anbieter sog. Suchmaschinen oder -agenten an, die zu vorgegebenen Stichworten entsprechende Dokumentadressen liefern. Leider haben diese den Nachteil, nur nach Stichworten und nicht nach Sinngehalt der Dokumente zu suchen. Nicht zuletzt deswegen hat sich das Bild vom Net-Surfer gebildet, der über die Links einzelner Dokumente quasi auf einer Informationswelle reitet.
Da niemand genau kontrollieren kann, welche Art an Information im Netz publiziert wird, bzw. sich aus einer Vielzahl offener und legaler Quellen geschützte Zusammenhänge ableiten lassen, kann die Informationsflut auch zu einem Unterlaufen des Datenschutzes führen. Dies und die ungeklärte Sicherheitsproblematik (Verschlüsselung, s.o.) stellen ernsthafte Bedrohungen für das Internet in seiner jetzigen, offenen Form dar.

1.6 Soziale Auswirkungen

Wie bereits oben angesprochen ergibt sich mit Hilfe des Internets eine Globalisierung der Information, Forschung und Entwicklung, Software Engineering etc. Darüber hinaus erfolgt aber auch eine globale Nivellierung der Ethik und moralischen Maßstäbe. Wenn man in einem globalen Dorf wohnt, hat man auch ähnliche Ansichten und Einstellungen. Weltbilder und Paradigmen der verschiedenen Kulturen werden sich kurz über lang annähern und möglicherweise zu einem neuen Bild verschmelzen.
Es besteht keine effektive Möglichkeit der Zensur im Internet. Hat jemand die notwendige Zugangstechnologie, stehen ihm grundsätzlich alle vorhandenen Informationsquellen offen. Damit ist jede Information jederzeit an jedem Ort verfügbar. Dabei kann es sich um offene Diskussionsforen als auch um verwerfliche Daten, wie pornographisches Material oder Anleitungen zum Herstellen von Bomben handeln. Besonders die beiden zuletzt angführten Beispiele werden häufig in den konventionellen Medien zitiert. Dabei wird jedoch vergessen, daß solche Informationen erstens mit erheblichem Aufwand gesucht werden müssen und zweitens diese Daten auch auf anderen, herkömmlichen Wegen zu erhalten sind. Da das Internet über Landesgrenzen hinweg reicht, ist es ohnehin schwer einheitliche juristische

(oder auch moralische, s.o.) Maßstäbe anzulegen, geschweige denn eine effektive Strafverfolgung aufzubauen. Gerade durch die Vielzahl und Diversifikation der vorhandenen Daten erhöht sich die Verantwortung des einzelnen, welche Art der Information sie sich oder anderen zugänglich macht, welcher Quelle sie Glauben schenkt und wie sie die Informationen für sich nutzt.
Man darf bei der gesamten Diskussion über die Informationsflut im Internet nicht vergessen, daß alle hier publizierten Daten auf der einen oder anderen Weise ohnehin existieren und daß lediglich der Zugang zu diesen Informationen durch das WWW erleichtert wurde.

Der Cybermensch ist ein zentrales Schlagwort im sozialen Zusammenhang des Internet. So befürchtet man, daß der direkte soziale Kontakt unter Teilnehmern am Internet verkümmern könne. Als letzte Konsequenz sehen Kritiker des Netzes hier die totale Aufgabe zwischenmenschlicher Beziehungen. Diese Befürchtungen mögen für einzelne Net-Surfer durchaus relevant sein. Aber wie bereits oben angeführt, bleibt hier nur wieder die Eigenverantwortlichkeit des einzelnen. Niemand darf oder kann einer anderen Person, vorschreiben wie sie ihr Leben zu führen hat.
Ganz anders hingegen die Problematik "gläserner Mensch". Durch unkontrollierte Publizierung personenbezogener Daten, durch Verknüpfung offener Quellen, durch Ausspioneren der Netzgewohnheiten einzelner Teilnehmer und im Extremfall durch Dekodieren privater Nachrichten entsteht die Gefahr, daß sich der Einzelne nicht mehr einer unerwünschten Durchleuchtung oder Überwachung entziehen kann. Hier kann man nur von außen her einschreiten, der einzelne hat kaum die Möglichkeit, einen solchen Angriff auf seine Privatspähre zu erkennen, geschweige denn zu verhindern.

1.7 Ausblick

Die oben angesprochenen Problematik des Schutzes des einzelnen wird eine Hauptaufgabe der nächsten Zeit sein. Bereits jetzt liegen Konzepte vor, die ein Verfolgen der Netzgewohnheiten der Teilnehmer unmöglich machen sollen. So soll es jedem überlassen sein, sich bei Nutzung des Internets zu anonymisieren. Jeder Teilnehmer hat bereits jetzt die Möglichkeit, seine Post zu kodieren, so daß andere seine Nachrichten nicht oder nur erschwert ausspionieren können.
Auch hier ist wieder der einzelne gefragt, einen gewissen Aufwand zu betreiben, um sich und seine Daten zu schützen. Daß alle erforderlichen Schutzmaßnahmen vom Internet selber getroffen werden, ist weder praktisch durchzusetzen noch wünschenswert. Jede automatische Schutzvorschrift bietet Hackern Gelegenheit, sie zu durchbrechen, somit können bei ausreichender krimineller Energie keinerlei Schutzgarantien abgegeben werden. Ziel kann es daher nur sein, das Internet so sicher zu machen, wie es konventionelle Kommunikationsmittel sind. Auch Telefone können abgehört und Post abgefangen werden.

Herausforderungen liegen auch in der Größe des Netzes. Bei 30 Mio. Hosts fehlt einfach jede Möglichkeit, die auf ihnen bereitgehaltene Information zu katalogisieren. Die aktuell existierenden Suchmaschinen haben den Nachteil, vorhandene WWW-Seiten nur nach Stichworten organisieren zu können. Einen Sinngehalt der Dokumente läßt sich daraus nur

bedingt herleiten. Hier existiert die Idee, die Verfasser der jeweiligen Seiten in die Pflicht zu nehmen. Sie sollen einem Standard gemäß ihre Texte indizieren, wie dies für Bücher bereits seit Jahren der Fall ist. Die Größe des Internets setzt aber auch hier Grenzen. So gibt es lokal unterschiedliche Ansichten, wie so ein Index auszusehen hat und was er an Information zur Verfügung stellen soll. Darüber hinaus streiten sich naturgemäß auch die großen Provider und Software-Häuser über die notwendigen Standardisierungen.
Da das Internet ein offenes und damit auch dynamisches Netz ist, besteht auch keinerlei Garantien darüber, daß einmal gefundene Daten zu späteren Zeitpunkten immer noch im Netz vorhanden sind. WWW-Seiten können geändert, gelöscht oder an andere Stellen verschoben werden. Einmal erstellte Archive, die Daten aus dem Netz verwenden, haben also ebenfalls nur eine begrenzte Lebensdauer bzw. Verläßlichkeit. Daher benötigt die gezielte Suche nach Information im WWW wohl auch noch in nächster Zukunft das menschliche Urteilsvermögen. Inwieweit dieses doch durch Computertechnik ersetzt werden kann, sollen die nachfolgenden Arbeiten dieses Seminares klären.

Dank der immensen Kapazitätssteigerungen der Backbones, können gleichermaßen auch die Volumina der transportierten Daten wachsen. Bereits heute wird an Erweiterungen des HTML-Standards gearbeitet, die den Aufbau von Web-Seiten mit 3D-Grafiken erlauben sollen. Dadurch kann Information noch plastischer, 'greifbarer', dargestellt werden. Gleichzeitig wird die oben bereits angesprochene Internet-Telefonie ausgebaut. Es existieren bereits Bildtelefonsysteme, welche die Durchführung kompletter Vidoekonferenzen erlauben. Wieweit diese Entwicklung noch gehen wird, läßt sich nicht abschätzen. Das Internet stellt jedenfalls keiner Entwicklung Steine in den Weg. Wann der Rechner zum Fernseher und zur Tageszeitung wird, scheint jedenfalls nur noch eine Frage von wenigen Jahren zu sein, glaubt man den Ausführungen der großen Software-Ersteller. Viele deutsche Fernsehsender jedenfalls bieten zu den meisten Sendungen weiterführende Informationen im Netz an. Einige stellen stellen sogar Teile ihrer Sendung direkt ins WWW, wie z.B. die Tagesschau. Seit kurzem wird sogar der Einzug von Rundfunkgebühren durch die GEZ erwogen, seit ein bayrischer Radiosender sein komplettes Programm im Netz verbreitet.

Enormes und derzeit noch weitestgehend ungenutztes Potential liegt in der Möglichkeit, das Netz als Arbeitsplattform zu nutzen. Die Projektarbeit in individuell zusammengestellten virtuellen Arbeitsgruppen findet derzeit nur bei kleinen Softwareprojekten Anwendung. Natürlich scheuen sich Industrie und Handel, große, wichtige und u.U. schützenswerte Projekte an ihnen grundsätzlich unbekannte Programmierer zu vergeben. Es wird auch immer Projekte geben, die nicht außer Haus durchführbar sind bzw. die nur in lokal abgegrenzten Gruppen durchgezogen werden können. Die Tendenz aber geht eindeutig in die Richtung, virtuelle und zeitlich begrenzte Teams zusammenzustellen. Denn nur dadurch kann man sich dem globalen Preisdruck stellen, der u.a. auch erst durch den Boom des Internets ausgelöst wurde. Insbesondere Dienstleistungen waren bis vor wenigen Jahren noch national konzentriert. Doch heute spielt es keine Rolle mehr, ob ein Programmierer in Deutschland, Rußland oder Indien sitzt; wo eine im Internet vertretene Bank ihren Sitz hat, spielt auch nur eine untergeordnete Rolle.
Durch das Internet entstand ebenfalls die Möglichkeit, qualifizierte Heimarbeitsplätze zu

schaffen. Sofern das Einsatzgebiet keine persönliche Anwesenheit erfordert, bieten sich hier große Einsparpotentiale. Ähnlich dem Gleitzeitkonzept, kann der einzelne entscheiden, wann er seine Leistung erbringt; Anfahrtszeiten und Bürobedarf entfallen. Insbesondere in Schweden wird dieser Weg seit einigen Jahren konsequent auf- und ausgebaut.

Das sog. Electronic Commerce, der elektroniche Handel, bietet die Möglichkeit global im Netz Handel jeder Art zu betreiben. Ob man sich Medikamente, Bücher oder Software, also beliebige Produkte, bestellt, oder aber Bankgeschäfte tätigt, das Internet bietet hierfür eine weltumspannende Infrastruktur an. Folgt man der Idee deutscher Banken, elektronisches Geld auf Chipkarten zu laden, kann man prinzipiell dieses Geld auch im Internet direkt ausgeben. Mit der Entwicklung der notwendigen Sichheitsstrukturen wird es jedenfalls innerhalb weniger Jahre zu erwarten sein. Bereits vor einiger Zeit wurde in einem spielerischen Experiment die Akzeptanz des "Internet-Geldes" überprüft.
Mittlerweile geben ein Großteil der in Deutschland werbenden Firmen bereits Internet-Adressen während ihrer Werbung an. In erster Linie natürlich, um Informationen über die Resonanz ihrer Werbung zu gewinnen und um ihr Produkt weiter zu bewerben. Aber einige, hier sind besonders die Versandhäuser zu nennen, bieten gleichzeitig die Möglichkeit, Bestellungen aufzugeben.
Eine weitere Neuerung im Rahmen des Electronic Commerce, stellt das Konzept des virtuellen Marktplatzes dar: verschieden Branchen sammeln sich unter jeweils einer Adresse. Was auf den ersten Blick unsinnig und widersprüchlich erscheint, nämlich die direkte Nähe zur Konkurrenz zu suchen, erweist bei genauerem Hinsehen als durchaus vorteilhaft. Ähnlich den Diamantenstraßen in Amsterdam und New York, konzentriert sich somit ein Großteil der potentiellen Kundschaft auf wenige Anbieter.

All dies führt zu dem Fazit, daß das Internet, wie jede Technologie, Gefahrenpotential enthält, dieses aber durch ein ungeheures Chancenpotential relativiert wird. Lamentieren über unglöste Sicherheitsfragen löst diese nicht. Sicher ist jedenfalls, daß das Internet, und hier insbesondere das WWW, eine Technologie ist, die aus userem zukünftigen Leben sowenig wegzudenken ist, wie Telefon oder Fernsehen. Es gilt damit, die besondere Dynamik des Netzes positiv zu nutzen und an seiner Entwicklung aktiv teilzunehmen, als Nutzer oder Anbieter.

Kapitel 2

Caching im WWW

Hans-Jörg Fischer

Kurzfassung *Im folgenden Abschnitt werden Methoden und Ansätze für das Caching im WWW vorgestellt. Hierbei werden insbesondere Strukturen der Caches beleuchtet. Außerdem werden auf die spezifischen Problemfelder des Cachings im WWW eingegangen und ein kurzer Ausblick auf die anstehende weitere Entwicklung gegeben.*

Die Techniken des HTTP/HTML werden im zunehmenden Maße für Problemstellungen benutzt, für die sie ursprünglich nicht entworfen wurden; als Beispiele seien Datenbankzugriffe, Hilfesysteme und Fernwartung genannt. Aus diesem Grunde kommt es im zunehmenden Maße zu Netzwerkverkehr, sowohl auf dem Internet, als auch innerhalb des Intranets, da die genannten Techniken ein hohes Maß an redundantem Datenaufkommen verursachen.

Eine Möglichkeit unter Ausnutzung dieser Eigenart dem Problem zu begegnen, stellt das *Caching* dar ([CDN+95, CHH+94, Har]).

2.1 Begriffsdefinition Caching

Im Zusammenhang mit dem Word-Wide-Web versteht man unter *Caching* die Zwischenspeicherung von Anfragen einer Menge von Rechnern auf einem Ihnen zugeordneten Cache-Rechner (vgl. Abb. 2.1). Hierzu müssen diese gebündelt über den Cache-Rechner abgewickelt werden. Dieser versucht die Anfragen zuerst aus einem Pool der bereits angeforderten Datenblöcke zu befriedigen; schlägt dies fehl, so greift er auf andere Rechner, u. U. auch den eigentlichen Zielrechner der Anfrage, zurück. Im Rahmen dieser Abhandlung werden nur die beiden für das WWW derzeit wichtigsten Protokolle, *HTTP* (*Hypertext Transfer Protocol*) und *FTP* (*File Transfer Protocol*) behandelt.

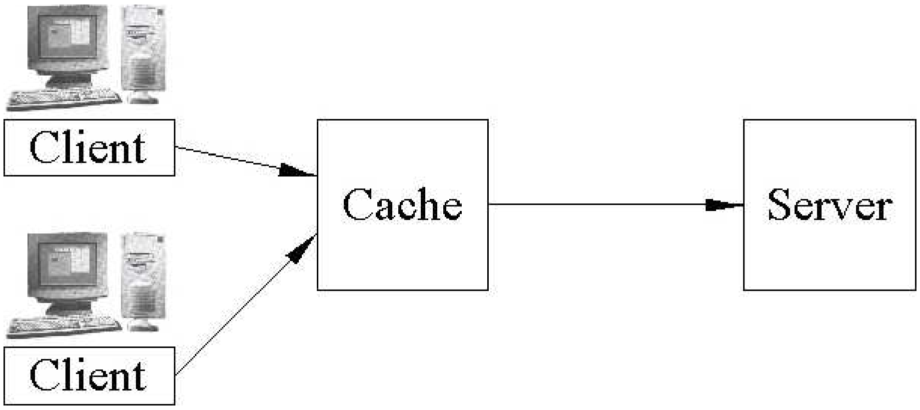


Abbildung 2.1: Funktionsweise eines Caches

	Bandbreite		
	minimal	häufig	maximal
LAN	10 Mbps	10 Mbps	100 Mbps
WAN	64 kbps	2 Mbps	50 Mbps

Tabelle 2.1: Übliche Bandbreiten

2.2 Ziele des Cachings

Ziel des Cachings ist im Wesentlichen die Verringerung der Netzlast bzw. das Verlagern eines Teiles des Transportaufkommens in das lokale Netzwerk.

Dies geschieht unter Berücksichtigung der Tatsache, daß in letztgenannten Netz im Allgemeinen wesentlich höhere Bandbreiten zur Verfügung stehen, als dies gemeinhin bei Weitverkehrsverbindungen der Fall ist. Die in der Praxis anzutreffenden Bandbreiten sind in Tab. 2.1 exemplarisch aufgeführt. Hierbei ist insbesondere zu berücksichtigen, daß die Bandbreite, die jedem einzelnen Benutzer zur Verfügung steht, bei gegebenem Medium und wachsendem Nutzerkreis zurückgeht.

Durch die Entlastung der Weitverkehrsnetze kann außerdem, falls diese auf Basis der übertragenen Datenmenge abgerechnet werden, eine Senkung der Netzkosten erreicht werden.

Darüber hinaus wird sich durch die oben skizzierte Veränderung der Netzbelastung das Antwortverhalten für die einzelnen Benutzer verbessern.

Beim Entwurf der FT- und HTT-Protokolle wurde nicht auf ein mögliches Caching der zu transferierenden Daten geachtet. Aus diesem Grunde stellte sich beispielsweise beim Dateizugriff über FTP die Situation ein, daß zu einem Server eine Menge von *Spiegel-Servern* (*Mirror*) entstanden sind, die die gleichen Daten halten. Beim Zugriff auf die Daten lag es damit in der Verantwortung des Benutzers, welchen Server er wählte. Da viele Benutzer aus unterschiedlichsten Gründen nicht auf die Mirror zugreifen, ist ihr Nutzen in der Praxis vergleichsweise gering. Aus diesem Zusammenhang kann man unmittelbar die

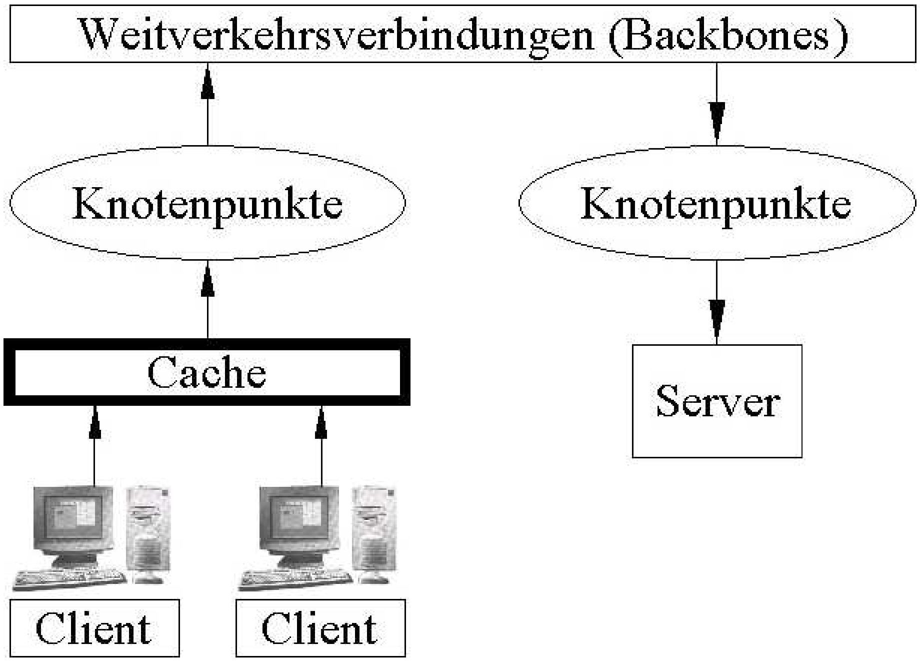


Abbildung 2.2: Cache nahe an Clients

Forderung nach einem *transparenten* Caching der Zugriffe ableiten.

2.3 Strukturkonzepte für den Cache

Die Qualität eines Zwischenspeichers wird im Allgemeinen über die Trefferrate qualifiziert; d. h. welcher Teil der Anforderungen durch den Cache befriedigt werden können, ohne auf die eigentlichen Quelle zurückgreifen zu müssen. Da sich Datenquellen im WWW bezüglich ihrer Geschwindigkeit und Kosten beim Zugriff unterscheiden, erscheint es sinnvoll, diese Eigenschaften ebenfalls als Qualität beim Zugriff zu berücksichtigen. Somit wird ein Cache bei gleicher Zugriffshäufigkeit Daten von weiter entfernten Quellen bevorzugt zwischenspeichern.

Entsprechend dem spezifischen Einsatzzweck wird man die Struktur des Caches wählen, um die bestmögliche Nutzung zu erreichen. Hierbei stehen verschiedene Ansätze zur Auswahl.

2.3.1 Plazierung nahe am Client

Bei der Plazierung nahe am Client wird der Cache-Rechner mittelbar an den Arbeitsplatzrechnern angebracht. Beispielsweise könnte ein Cache einem Poolraum mehrerer Rechner zugeordnet sein und ihren Datenverkehr abwickeln. Wie diese Konstellation aussieht, skizziert Abb. 2.2

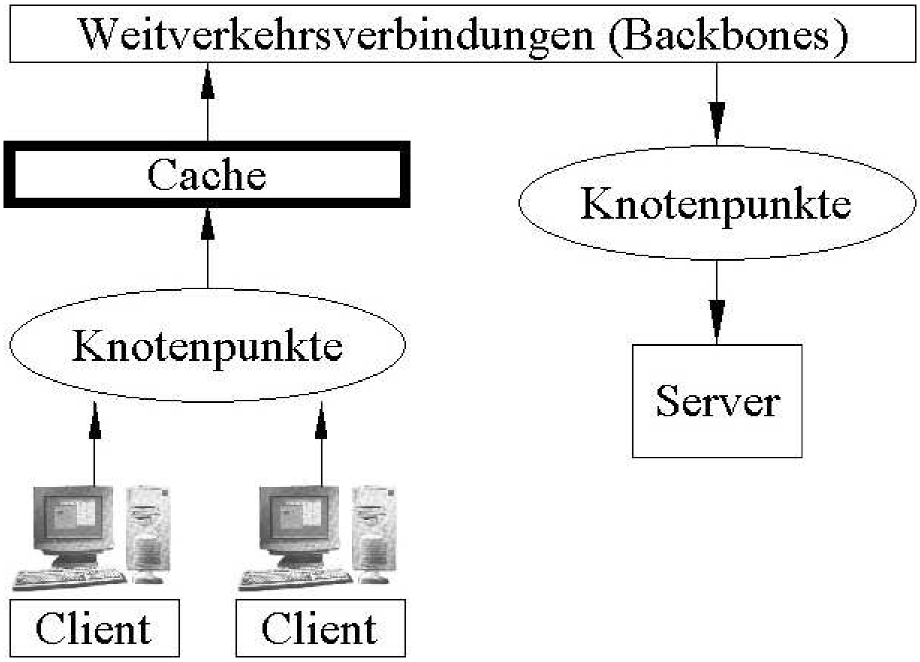


Abbildung 2.3: Cache an Backbones

Somit ist er von ihnen aus über das LAN zu erreichen, womit der Datenverkehr möglichst lokal im gleichen Netzsegment bleibt. Dadurch, daß der Cache-Rechner nur einer überschaubaren Anzahl von Clients zur Verfügung steht, hat man es desweiteren mit einer vergleichsweise geringen und abschätzbaren Arbeitslast auf dem diesem Computer zu tun.

Nachteilig wirkt sich aus, daß die Wirkung des Caches nur einer relativ geringen Anzahl von Rechnern zugute kommt.

2.3.2 Plazierung nahe an Backbones

Ein anderer Ansatz ist die Plazierung des Caches nahe an Backbones, d.h. Weitverkehrsverbindungen, um diese gezielt zu entlasten. Wie man sich dies vorzustellen hat, wird in Abb. 2.3 skizziert.

In diesem Fall werden Anfragen, die normalerweise über den Backbone geleitet werden würden, durch den Cache "gefiltert"; es kann eine Teilmenge der Anfragen durch diesen befriedigt werden.

Allerdings kann der so gestaltete Cache nicht die Knotenpunkte zwischen sich und den Clients entlasten. Diese werden sich erfahrungsgemäß im weiteren Verlauf als Engstellen erweisen.

Desweiteren steht dem Vorteil, eine große Menge von Clients zu bedienen, das Problem einer äußerst hohen Arbeitslast auf dem Cache-Rechner gegenüber. Somit muß auch hier damit gerechnet werden, daß dieser als Engstelle in der Übertragung auftritt.

2.3.3 Hierarchische Caches

Um die Vorteile der beiden bisher vorgestellten Ansätze zu vereinigen, bieten sich *Hierarchische Caches* an. Hierbei hat man es mit einer verallgemeinerten Baumstruktur zu tun, bei der jedem Cache-Rechner außer den Wurzelrechnern mindestens ein *Eltern*-Rechner zugeordnet ist. Außerdem können auch zwischen Rechnern der gleichen Ebene Verbindungen bestehen. Man spricht hierbei von einer *Geschwister*-Beziehung.

Wenn ein Rechner der Cache-Phalanx eine Anfrage einer seiner Clients erhält, versucht er zuerst, diese selbst zu befriedigen. Scheitert dies, wird sie an eventuell vorhandene Geschwister-Rechner weitergereicht. Ist dies ebenfalls nicht von Erfolg gekrönt, wird geprüft, ob der in der Dokument-URL genannte Server-Rechner hinreichend nah am eigenen Cache-Rechner liegt. In diesem Falle wird die Anfrage direkt gestellt; andernfalls wird sie an den Eltern-Rechner übergeben.

Dieses Vorgehen spielt sich auf jeder Stufe der Hierarchie ab, bis entweder die Anfrage aufgelöst wurde oder die oberste Stufe der Hierarchie erreicht ist. Entsprechend wird dann die Anfrage an den geforderten Host direkt weitergereicht.

Aus dem Zusammenführen der beiden ersten Strukturansätze hat man den Vorteil, daß die auf den Cache-Rechnern entstehende Last gut gestaffelt und verteilt ist. Die Cache-Hierarchie kann Anfragen wesentlich effizienter und somit schneller als die gleiche Menge isolierter Cache-Rechner beantworten. Außerdem zeigt sie sich unempfindlicher gegenüber Ausfällen einzelner Rechner; ihre Aufgaben können von anderen Rechnern der gleichen Ebene übernommen werden.

Dem gegenüber steht ein wesentlich erhöhter Administrationsaufwand, da nicht nur einzelne Rechner, sondern auch ihre Verknüpfungen gepflegt werden müssen.

Für den Betrieb der Hierarchie muß desweiteren ein von HTTP abweichendes Protokoll verwendet werden, da es für diese Belange nicht geeignet ist. Der in der Praxis oft anzutreffende *Harvest-Cache* (*squid*) verwendet beispielsweise das *HOP* (*Harvest Object Model*).

2.4 Probleme des Cachings im WWW

Der Nutzen, den man aus dem Caching von Daten im WWW zieht, erkauft man sich mit einer Anzahl von Problemen, deren Lösung zwingende Voraussetzung für einen gewinnbringenden Einsatz ist.

2.4.1 Auswahlstrategien

Da die Resourcen eines Caches begrenzt sind, muß im Betrieb entschieden werden, nach welchen Kriterien Daten in den Cache aufgenommen und aus diesem gestrichen werden, um eine maximale Trefferquote bei Anfragen an den Cache zu erhalten.

So ist es z.B. zweckmäßig, kleinen Dateien im Cache eine höhere Priorität als großen einzuräumen, um zu verhindern, daß zugunsten einer einzelnen Übertragung einer großen Datei mehrere kleinere Dateien entfernt werden.

Auch der Typ der Daten kann als Kriterium in diese Betrachtungen aufgenommen werden: Bilder sind im WWW im Allgemeinen weniger Änderungen unterworfen, als dies bei Texten der Fall ist. Dementsprechend erscheint ihr cachen lohnenswerter.

Einsichtig ist weiter, daß Abfrageergebnisse nicht in den Cache gelangen dürfen, da dieser nicht auf Veränderungen der Datenbasis reagieren kann und häufige Veränderungen der Daten zu erwarten sind.

2.4.2 Abgleich mit veränderten Daten

Wichtig beim Caching von Datensätzen ist die Fortscheibung ihrer Änderungen; d. h. die Kohärenz der über den Cache abgerufenen Daten.

Unglücklicherweise ist in der jetztigen Fassung des HTT Protokolles keine Unterstützung eines cache-Kohärenz-Protokolls vorgesehen. Aus diesem Grunde kann man sich Cache-seitig nicht auf eine Benachrichtigung von Änderungen durch den Server verlassen. Die intuitive Lösungsstrategie wird damit ausgeschlossen.

Statt dessen bleiben momentan nur zwei Ansätze, die sich ergänzen:

- *Gültigkeitszeitraum* Bei diesem Ansatz werden den in den Cache eintretenden Daten Gültigkeitszeiträume zugeordnet. Nach Ablauf dieser Frist sind die Daten entweder zu verwerfen oder aber neu mit dem Server abzugleichen. Hierzu existiert im HTT-Protokoll ein "Get if Modified" Kommando.

  Welcher der beiden Wege beschritten wird, hängt von der Häufigkeit, mit der die Daten abgerufen wurden, ab.

  Das wesentliche Problem hierbei ist das Bestimmen der Gültigkeitszeiträume. Zwar können in HTML Dokumenten Verfallsdaten angegeben werden, leider machen von dieser Möglichkeit nur ein Bruchteil der Autoren Gebrauch. Auf Seiten des Caches muß man deshalb oft eigene Gültigkeitszeiträume erzeugen. Man legt hierbei statistische Erfahrungswerte als Funktion des Dokumentalters zugrunde.

  Demnach haben beispielsweise Bildinformationen eine höhere Lebensdauer im Cache als Textdokumente. Außerdem wird, je älter ein Dokument auf dem Server bereits ist, ihm eine entsprechend verlängerte Lebensdauer im Cache zugewiesen. Damit kann man gut häufige Änderungen neuer Seiten nachbilden und andererseits etablierte Informationsangebote des WWW lange im Cache halten.

- *Manuell* Obiger Ansatz kann nicht immer kohärente Daten garantieren. Aus diesem Grunde bietet man dem Nutzer die Möglichkeit, manuell ein erneutes Laden der Daten vom Server in den Cache anzustoßen. Dies geschieht im Allgemeinen durch Aufruf der Funktion "*Reload*" innerhalb des Browsers.

  Unglücklicherweise sind durch den Benutzer ungültige bzw. veraltete Daten nicht immer als solche zu erkennen.

2.4.3 Sicherheit und Verfügbarkeit

Ein wesentlicher Punkt bei der Diskussion über Caches ist ihr Einfluß auf die Sicherheit innerhalb des Netzwerkes und die Verfügbarkeit der über sie abgewickelten Dienstleistungen.

Sicherheit

Aus der Sicht des Clients muß ein Cache-Rechner auf jeden Fall vertrauenswürdig sein, da über ihn die angeforderten Daten laufen. Ein Eindringen in den Cache-Rechner würde schlimmstenfalls Einblick in die übertragenen Daten bis hin zu Passwörtern liefern.

Um dieses Sicherheitsrisiko etwas zu senken, werden bei heutigen Cache-Systemen sicherheitssensitive Daten (in diesem Fall solche, die authentisiert übertragen wurden) nicht gecachet. Somit sind diese Daten nur während der Übertragung und nicht darüber hinaus auf dem Cache-Rechner präsent.

Auch aus der Sicht des Servers muß ein Cache-Rechner als vertrauenswürdig eingestuft sein, da für den Dienstebieter nicht anderweitig nachvollziehbar ist, wer sich hinter dem Cache befindet. Er kann nur die Verbindung zum Cache selbst verfolgen.

Während diese Eigenart innerhalb eines Intranets mit sensitiven Daten eher problematisch ist, kann sie im Internet erwünscht sein, da sie verhindert, daß Server von einzelnen Rechnern im Netz Zugriffsprotokolle anlegen. Der Cache arbeitet als "Anonymisierer" und schützt somit quasi die persönlichen Daten des Nutzers. Allerdings kann das durch Verwendung von "Cookies" oder auch "Java-Applets" unterwandert werden.

Als weiterer Schritt in dieser Richtung bleibt noch, das lokale Unternetz über eine Firewall vom Internet abzutrennen und jeden Zugriff nach außen über den Cache abzuwickeln. Der Cache übernimmt in dieser Konstellation die Aufgabe eines Proxies. Falls die intern benutzten IP-Adressen nicht im Internet bekannt gemacht sind, kann man davon eine wesentliche Verbesserung der Sicherheit im lokalen Netzwerk erwarten.

Verfügbarkeit

Grundsätzlich senkt der Cache als zusätzlicher "Point of Failure" die mittlere Verfügbarkeit der über ihn abgewickelten Dienste. Da aber insbesondere bei Internet-Verbindungen eine wesentliche Geschwindigkeitssteigerung aus der Benutzung eines Caches folgt und bei solchen Anwendungen die Fehlzeiten anderer Rechner diejenigen des eigenen Caches übersteigen, ist nach Meinung des Autors dieses Handicap tolerierbar. Es wird aber eingeräumt, daß im Zusammenhang mit anderen Anwendungen, insbesondere im Intranet, diese Überlegungen den Einsatz eines Caches verbieten können.

2.5 Kosten

Im folgenden sollen die entstehenden Kosten einer Cache-Installation, das "*Cost of Ownership*", den erzielbaren Einsparungen gegenüber gestellt werden.

Die “Cost of Ownership” teilen sich in zwei Bereiche: die Installationskosten und die laufenden Kosten. Erstere sind ein einmaliger fester Betrag, der für die anzuschaffende Hardware, Software-Lizenzen sowie Installation aufzubringen ist.

Wesentlicher für die Nutzenüberlegung sind die laufenden Kosten, die aus dem Betrieb der Anlage entstehen. Sie bestehen aus Wartungskosten, Betriebskosten (Strom) und möglicherweise Erweiterungen der Soft- und Hardware.

Demgegenüber ist zu überlegen, welche Kostenpunkte durch die Einrichtung und den Betrieb der Anlage einzusparen bzw. zu senken sind. Hier sind vor allem Netzkosten zu nennen, also diejenigen Kosten, die bei der Nutzung von Leitungen entstehen. Sie entstehen im Allgemeinen nur bei WAN-Verbindungen; LAN-Verbindungen sind nach Installation kostenfrei zu nutzen.

Indirekt kann aber die effiziente Nutzung eines Caches die Installation neuer Leitungen beeinflussen bzw. diese überflüssig machen.

2.6 Ausblick

In den vorhergehenden Abschnitten wurde eine Bestandsaufnahme des Caching im WWW aufgenommen. An dieser Stelle nun soll versucht werden, die weitere diesbezügliche Entwicklung abzuschätzen.

Wie bereits angesprochen wird das HTT Protokoll in zunehmenden Maße für Aufgaben in Anspruch genommen, für die es ursprünglich nicht konzipiert war. Dementsprechend sind die Probleme, die beim Caching innerhalb des WWW auftreten, oftmals auf fehlende Hilfestellungen innerhalb des Protokolles zurückzuführen.

Während HTML einem dauernden Wandel unterworfen ist, zeigt das HTTP weit weniger Dynamik. Erst in letzter Zeit wurden Forderungen nach einem neuen Protokoll, HTTP-NG (*Hypertext Transfer Protocol-Next Generation*) laut, die mit vielen Beschränkungen und Problemen des bisherigen Protokolles aufräumen sollen. Die Wesentlichen sind hierbei:

- *Größere Informationseinheiten* Bisher muß für die Übertragung jeder einzelnen Datei (bzw. Informationseinheit) eine eigene Übertragung initiiert werden. Das heißt im konkreten Fall, daß für das Einlesen einer einzigen Seite mehrere Anfragen an den Server (eine für die Seite, jeweils eine pro Bild, ...) gesandt werden müssen. Dies senkt die Übertragungsrate wesentlich.

  Aus diesem Grunde soll im neuen Protokoll die Möglichkeit gegeben werden, solche Einheiten, die eh immer im Verband abgerufen werden, auch als solche zu übertragen.

- *Cache-Notifing* Die bereits angesprochene fehlende Unterstützung für die Benachrichtigung von Caches über Änderungen an Daten soll eingeführt werden.

Erst mit Einführung dieses neuen Protokolles wird eine rundum zufriedenstellende Lösung des Caching-Problems im WWW möglich sein, da dann erst eine dauernde Kohärenz der abgerufenen Daten zu gewährleisten ist.

Kapitel 3

Klassische Techniken des Information Retrieval

Ivanov Sotir

3.1 Einleitung

Nach einem Dokument mit einem bestimmten Inhalt zu suchen (Buch, Artikel, ... usw.), war schon immer schwierig, besonders wenn man gar nicht genau weiß, ob zum gefragten Thema überhaupt Literatur existiert. Im überschaubaren Rahmen kann eine gute Buchhändlerin, erfahrene Bibliotheksbedienstete oder eine Expertin auf dem jeweiligen Sachgebiet weiterhelfen. Schwieriger wird es, wenn die Materialmenge, in der gesucht werden muß, für einen Menschen zu groß oder zu unübersichtlich ist.
Datenbanken enthalten homogen strukturierte Datensätze, d.h. mit einer bestimmten Art und Anzahl von Attributen. Dabei ist es leicht, die Datensätze zu identifizieren, die einer bestimmten Anfrage entsprechen, weil die Anfrageformulierung einige Attributwerte beinhaltet, die die Datensätze charakterisieren. Jeder gefundene Datensatz hat denselben Attributwert, der schon in der Anfrage spezifiziert wurde, während alle übrigen Datensätze mindestens einen Attributwert besitzen, der mit dem Attributwert in der Anfrage nicht übereinstimmt.
In der textbasierten Informationsverarbeitung sieht es ähnlich aus wie bei Datenbankanfragen. Der Unterschied besteht darin, daß jetzt zusätzlich zu den formalen Attributen, die den Text charakterisieren, auch der Inhalt betrachtet werden muß.
Der Inhalt wird durch sogenannte Stich- oder Schlagworte beschrieben. Jedes Schlagwort beschreibt den Inhalt nicht vollständig, sondern nur bis zu einem bestimmten Umfang. Jedem Text wird eine Anzahl von Stichworten zur Charakterisierung des Inhalts sozusagen als Kurzfassung (Abstract) beigefügt. Bei diesen Umständen muß der Ausdruck in der Anfrage nicht vollständig mit dem Abstract des Dokuments übereinstimmen, sondern es reicht, wenn die Anfragebezeichner nur in einem bestimmten Maß mit den Dokumentbe-

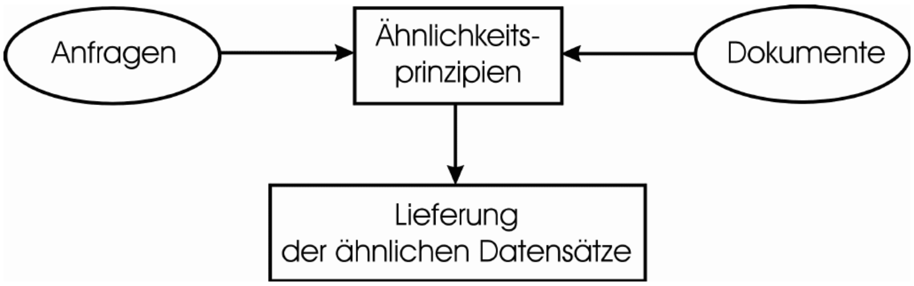


Abbildung 3.1: einfache Anfrage

zeichnern übereinstimmen.
Wir wollen für unsere Überlegungen eine Sammlung von elektronisch vorliegenden Dokumenten (=Datensätzen) zugrundelegen, die zum einen durch die charakteristischen bibliographischen Angaben von Autor, Titel, ... und einer Menge von Worten beschrieben werden und zum anderen einen Verweis auf die Lokation des Volltextes (d.h. Regalstandort oder Speicherplatz auf der Festplatte) enthalten.([Sal89])

3.2 Invertierte Listen und boolesche Anfragen

Die Methode der invertierten Listen eignet sich sowohl zum Indexieren der Kurzfassungen bzw. der Schlagworte von Dokumenten als auch zum Indexieren der Volltexte. Bisher wurden die Dokumente in Bibliotheken nach Schlagworten gesucht. Das Ergebnis war die Angabe des Standorts im Regal. Heute suchen wir in digitalen Bibliotheken oder im WWW nach vollständig indexierten Dokumenten, die uns dann in einem geeigneten Format (z.B. zum online–Lesen oder Ausdrucken) präsentiert werden. Prinzipiell basiert das Finden von gespeicherten Dokumenten als Antwort einer Informationsabfrage auf Ähnlichkeitsprinzipien zwischen Anfragen und gespeicherter Information. Das Ergebnis sind diejenigen Dokumente, die in einem bestimmten Maß ähnlich zu der Anfrage sind. Dieser Vorgang ist in Abbildung 3.1 skizziert.

Zusätzliche Zwischenschritte transformieren die Informationsanfrage in formale Anfrageanweisungen und die Datensätze in prägnante Kurzfassungen, bevor der eigentliche Vergleich ausgeführt wird (Abbildung 3.2).

Wie schon gesagt wird der Inhalt eines Dokuments durch die Schlagworte beschrieben. In Anfragen können logische (boolesche) Verknüpfungen von Suchbegriffen mit “AND”, “OR” oder “NOT” vorgenommen werden. Sind zwei Begriffe mit AND verknüpft, so müssen beide im Dokument vorkommen, sind sie mit OR verknüpft, reicht es, wenn einer von beiden auftritt, damit das Dokument ausgegeben wird. NOT kommt normalerweise in Verbindung

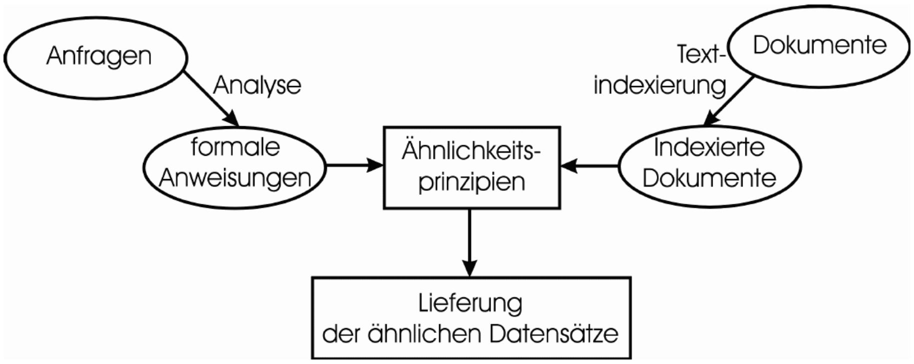


Abbildung 3.2: erweiterte Anfrage

mit AND zum Unterdrücken der Lieferung bestimmter Begriffe. So wird für jedes Wort ein Index d.h. eine invertierte Liste konstruiert, in dem die Dokumentadressen gespeichert werden. Die Menge der Indizes für alle Worte ist unter dem Namen **invertierte Datei** bekannt. Mit einer invertierten Datei kann man die Dokumente, die einer bestimmten Anfrageformulierung entsprechen, leicht bestimmen. So kann man von der invertierten Datei die Liste der Dokumente entnehmen, in denen das Wort von der Anfrage auftritt. Stellen wir uns z. B. vor, wir haben 4 Dokumente, die durch das Vorkommen von verschiedenen Begriffen/Worten charakterisiert sind (Abbildung 3.3).

	Begriff 1	**Begriff 2**	**Begriff 3**	**Begriff 4**
Dokument 1	1	1	0	1
Dokument 2	0	1	1	1
Dokument 3	1	0	1	1
Dokument 4	0	0	1	1

Abbildung 3.3: Beispiel

Eine 1 bedeutet dabei, daß das Dokument den jeweiligen Begriff beinhaltet. So ist Dokument 2 durch den Begriffvektor (0,1,1,1) dargestellt, also alle Begriffe außer Begriff 1 sind in diesem Dokument enthalten. Die invertierte Datei erhält man dadurch, daß die Zeilen und Spalten in der neuen Matrix vertauscht werden. Jede Zeile in der invertierten Matrix repräsentiert jetzt eine Liste der Dokumente, in denen der Begriff auftaucht. So bedeutet Zeile 2 in Abbildung 4, dargestellt durch (1,1,0,0), daß Begriff 2 die Dokumente 1 und 2, aber nicht die 3 und 4 beschreibt. Die booleschen Operationen auf der invertierten Datei basieren dabei auf Mischen und Duplikateleminierung von sortierten Listen.

Bei einer Anfrage kann man dann auf folgende Weise vorgehen:

- Für boolesche AND-Anfragen wie z.B. (T_i and T_j) wird eine Liste aus den Zeilen i und j der invertierten Datei erstellt, die nur diejenigen Dokumente enthält, die in

	Dokument 1	Dokument 2	Dokument 3	Dokument 4
Begriff 1	1	0	1	0
Begriff 2	1	1	0	0
Begriff 3	0	1	1	1
Begriff 4	1	1	1	1

Abbildung 3.4: Invertierte Liste

Anfrage	**((T1 or T2) and not T3)**	
invertierte Listen	**T1:**	**{R1,R3}**
	T2:	**{R1,R2}**
	T3:	**{R2,R3,R4}**
	Mische (T1,T2)	**{R1,R1,R2,R3}**
	Ergebnis für (T1 or T2)	**{R1,R2,R3}**
	Mische ((T1 or T2),T3)	**{R1,R2,R2,R3,R3,R4}**
	Ergebnis für ((T1 or T2) and T3)	**{R2,R3}**
	Mische ((T1 or T2),((T1 or T2) and T3))	**{R1,R2,R2,R3,R3}**
	Ergebnis für ((T1 or T2) and not T3)	**{R1}**

Abbildung 3.5: Operationen auf dem invertierten Index

beiden Listen vorkommen.

- Für OR-Anfragen wie z.B. (T_i or T_j) wird eine Liste aus den Zeilen i und j erstellt, die die Dokumente enthält, in denen T_i oder T_j oder beide Begriffe vorkommen.
- Für NOT-Anfragen wie z.B.(T_i and not T_j) wird zuerst eine Liste für (T_i and T_j) erstellt und die wird mit der Liste für die Zeile i gemischt. Endergebnis sind nur diejenigen Dokumente, die nur einmal in der gemischten Liste vorkommen.

Ein Beispiel der Operationen auf dem invertierten Index für die Anfrage ((T_1 or T_2) and not T_3) ist in Abbildung 3.5 gezeigt.

3.3 Erweiterungen

Verschiedene Erweiterungen der klassischen Technik der invertierten Liste wurden mit dem Ziel zur Vereinfachung und Verbesserung der Ergebnisse vorgenommen.

3.3.1 Distanzparameter

Die Beziehungen zwischen zwei oder mehreren Begriffen können durch Hinzufügen von Distanzparametern in der Anfrage festgelegt werden. Zwei mögliche Parameter hier sind *within sentence* und *adjacency*. Bei (A *within sentence* B) müssen die Worte A und B in demselben Satz im gesuchten Text, bei (A *adjanct* B) in der Nähe sein. Diese Distanzbegrenzung wird durch Hinzufügen von Begriff-Lokalisierungsinformation in den invertierten Listen erreicht. So kann man eine *within*-Anfrage durch Hinzufügen der Nummer der Sätze, in denen der Begriff auftaucht, beantworten. Beispielsweise könnten die Worte "information" und "retrieval" in folgender Form in der invertierten Liste erscheinen:

$$\text{information} :\{R_{345}, R_{348}, R_{350}, ...\}$$
$$\text{retrieval} : \{R_{123}, R_{128}, R_{345}, ...\}$$

Wenn die Satzlokalisierung hinzugefügt ist, sieht die Liste folgendermaßen aus:

$$\text{information} : \{R_{345}, 25; R_{345}, 37; R_{348}, 10; R_{350}, 8; ...\}$$
$$\text{retrieval} : \{R_{123}, 5; R_{128}, 25; R_{345}, 37; R_{345}, 40; ...\}$$

Also das Wort "information" erscheint zweimal im Dokument 345, im Satz Nummer 25 und 37, dann im Dokument 348, Satz 10 und im Dokument 350, Satz 8. Wenn man die Listenspezifikationen für "information" und "retrieval" betrachtet, dann lautet das Ergebnis der Anfrage (information within sentence retrieval) Dokument R_{345}, weil die beiden Begrifflisten den Eintrag "R_{345},37" enthalten, also beide Begriffe sind im Satz 37 von Dokument 345 vorhanden. Durch Erweiterung der invertierten Listen, die sich mit der Paragraphnummer, der Satznummer in dem Paragraph und der Wortnummer in dem Satz beschäftigt, kann eine Anfrage der Form (information *adjacent* retrieval) beantwortet werden. So erfüllen z. B. die beiden Listeneinträge die Bedingung, in der Nähe zu sein, weil der erste Begriff im Paragraph 2, Satz 3, Wort 5 von dem Text im Dokument 345, und der zweite Begriff im Paragraph 2, Satz 3, Wort 6 von demselben Text erscheint.

$$\text{information} :\{R_{345},2,3,5\}$$
$$\text{retrieval} : \{R_{345},2,3,6\}$$

Durch die Einführung von Distanzparametern wird die Ergebnismenge im Vergleich zu einer booleschen AND-Anfrage kleiner, da in diesem Fall weniger nicht relevante Dokumente gefunden werden.

3.3.2 Gewichtete Begriffe

Die Einführung von Gewichten für die Begriffe im Dokument (und in der Anfrage) kann dazu beitragen, die Begriffe, die wichtiger für die Anfragezwecke sind, von denen zu unterscheiden, die nicht so wichtig sind. Ein Dokument kann dann in der Form

$$R=(T_1, 0.2; T_2, 0.5; T_3, 0.6)$$

dargestellt werden. Das bedeutet: Begriff 3 hat ein Bedeutungsgewicht von 0.6, während Begriff 1 ein Gewicht von 0.2 hat, d.h. Begriff 3 ist wichtiger als Begriff 1. Wenn die Gewichte der invertierten Liste zugewiesen sind (z.B. aufgrund von Häufigkeitsüberlegungen), dann können die Anfrageergebnisse je nach ihrem Gewicht in absteigender Anordnung dargestellt werden. Die Dokumente, die der Anfrage am ähnlichsten sind, werden zuerst aufgeführt. Das ist ein wesentlicher Vorteil gegenüber der Ergebnismenge einer booleschen Anfrage, die oft sehr groß ist und die Dokumente ungeordnet zurückliefert, so daß sie der Benutzer prinzipiell ganz durchlaufen muß. Bei einer "gewichteten" Ergebnismenge kann der Benutzer selbst bestimmen, ab welchem Schwellwert Dokumente aufgezeigt werden sollen bzw. wie weit er die Ergebnismenge durchläuft. Dabei kann er sicher sein, daß er die relevantesten Dokumente gesehen hat. Durch diese Erweiterung wird die Ergebnismenge kleiner, weniger relevante Dokumente werden nicht mehr zurückgeliefert.

3.3.3 Spezifikation der Synonyme

Es ist möglich, den Begriffen in einer Anfrage verschiedene Synonyme, die z. B. ein Thesaurus bereitstellen könnte (vgl. Abschnitt 3.5.2), hinzufügen. Bei der Auswertung der Anfrage werden dann sowohl die Originalbegriffe als auch die Synonyme betrachtet. Bei den booleschen Anfragen wird das Synonym dem Original mit einem OR-Operator hinzugefügt. Hier ein Beispiel:

$$((T_1 \text{ AND } T_2) \text{ OR } T_3) \text{ wird in}$$
$$(((T_1 \text{ OR } S_1) \text{ AND } T_2) \text{ OR } (T_3 \text{ OR } S_3))$$

transformiert, wobei S_1 und S_3 Synonyme zu T_1 und T_3 sind. Die Ergebnismenge wird durch die Synonyme größer, da auf diese Weise mehr relevante Dokumente gefunden werden können.

3.3.4 Wortstammreduktion

Um Wörter nicht nur als Zeichenketten, sondern auch gemäß ihrer "Bedeutung" zu vergleichen, kann man in einem ersten Schritt nicht die spezielle Form verwenden, in der sie auftreten, sondern ihre Wortstämme. Im Englischen ist es verhältnismäßig einfach, mit einer Reihe von Regeln die Endungen zu entfernen und Wörter automatisch auf ihren Stamm zurückzuführen. Im Deutschen wird bei der Bildung vieler Wortformen auch der Stamm verändert, d.h. es ist weniger erfolgreich bzw. erheblich aufwendiger, da man auf die Informationen in Wörterbüchern zurückgreifen muß. Zum Beispiel die Form PSYCH* repräsentiert Psychologe, Psychologie, Psychose usw. Die Ergebnismenge wird durch die Wortstammreduktion größer.

3.3.5 Zusammenfassung

Durch die oben betrachteten Erweiterungen der invertierten Listen, die sich auch beliebig kombinieren lassen, werden die Ergebnismengen zum Teil kleiner (Distanzparameter und Gewichte) und auch größer (Synonyme und Wortstammreduktion), auf jeden Fall aber entstehen durch diese präziseren Anfragemöglichkeiten bessere Ergebnisse, d.h. es werden mehr relevante und weniger unrelevante Dokumente gefunden.

Nummer der Anfrage	Anfrage Hierarchie (engste bis breiteste)	gelieferte Ergebnisse	davon relevant
0	(A and B and C and D)	2	2
1	(A and B and C) or (A and B and D) or (A and C and D) or (B and C and D)	6	5
2	(A and B) or (A and C) or (A and D) or (B and C) or (B and D) or (C and D)	23	15
3	(A or B or C or D)	86	25

Abbildung 3.6: Beispiel einer Quorum-level Suche

3.4 Die Gütemaße precision und recall

Die Benutzung von booleschen Anfragen birgt viele Schwierigkeiten für die Benutzer in sich, weil die Ergebnisse stark von der Anfrageformulierung abhängen. Kleine Änderungen haben große Auswirkungen auf die Anfrageergebnisse. Der Mangel an Kontrolle über die Größe der Ergebnisse kann untrainierte Benutzer in Schwierigkeiten bringen. Es gibt verschiedene Vorschläge zur Vereinfachung der Anfrage und zur Kontrolle über die Ergebnisse. Ein Beispiel ist die Quorum-level Suche. In der Quorum-level Suche wird die originale boolesche Anfrage, bestehend aus n Begriffen, durch eine neue Anfrage ersetzt, ausgewählt aus einer Liste von n booleschen Anfragen. Ein Beispiel für eine Anfrage mit 4 Begriffen ist in Abbildung 3.6 dargestellt.

Diese Liste besteht aus einer Reihe von Formulierungen, von einer ganz engen Formulierung (A and B and C and D), bis hin zur breitesten Formulierung (A or B or C or D). Die dazwischen liegenden Anfragen bestehen aus Unteranfragen, die man aus der Originalanfrage bekommt, indem man 1, 2, ... oder mehrere AND-Klauseln entfernt, d.h. es müssen 3 von 4 Begriffen, 2 von 4, und schließlich nur 1 von 4 Begriffen im Dokument vorkommen. In einer Anfragehierarchie von dem in der Abbildung 3.6 gezeigten Typ nimmt die Anzahl der Anfrageergebnisse mit der Breite der Anfrage zu. So kann der Benutzer zwischen verschiedenen Anfragen auswählen, damit die Ergebnisse seinem Wunsch entsprechen. Wenn die ausgewählte Anfrage sehr eng ist, dann ist die Anzahl der Ergebnisse klein, aber die meisten Dokumente davon sind wichtig. Umgekehrt wenn die Anfrage sehr breit ist, dann ist die Anzahl der Ergebnisse groß, aber darunter sind auch viele nicht relevante Dokumente. Vor allem zwei Hauptparameter werden für die Effektivität der Anfrage eingesetzt: *recall* (R, die Nachweisquote) und *precision* (P, die Relevanzquote). Recall ist das Verhältnis der wichtigen Dokumente, die man durch die Anfrage bekommt, zu der Gesamtzahl der wichtigen Dokumente. Precision ist das Verhältnis der Anzahl der wichtigen zu der Gesamtzahl der Dokumente, die man durch die Anfrage bekommt. Für enge Anfragen ist P hoch, weil alle Ergebnisse wichtig sind, gleichzeitig ist R klein, weil man nur wenige von den wichtigen Dokumenten durch die Anfrage bekommt. Wenn die Anfrageausdrücke breiter werden, dann nimmt die Gesamtzahl der durch die Anfrage gelieferten und wichtigen Dokumente, also R zu. Gleichzeitig wächst auch die Anzahl der durch die Anfrage gelieferten, aber unwichtigen Dokumente, was P kleiner macht. Also enge Anfragen liefern eine hohe Precision und einen kleinen Recall, während breite Anfragen genau das umgekehrte tun. Die Situation ist in Abbildung 3.7 dargestellt.

In der Abbildung sind die Anfrageergebnisse schraffiert. Die Anfrage teilt die Gesamtmenge in vier Teile:

(a) wichtige und gelieferte Dokumente
(d) wichtige, aber nicht mitgelieferte Dokumente
(b) gelieferte, aber unwichtige Dokumente
(c) unwichtige, ungelieferte Dokumente

Recall und Precision sind folgendermaßen definiert:

$$\text{Recall: } R = a \;/\; (a+d)$$
$$\text{Precision: } P = a \;/\; (a+b)$$

An dieser Stelle wird auch die Wirkung der Erweiterungen (Abschnitt 3.3) nochmals klarer: Distanzmaße und Gewichte erhöhen P (b wird kleiner), Wortstammreduktion und Synonyme erhöhen R (die Anfrage wird breiter formuliert). Angestrebt werden Ergebnisse, die sowohl bei P als auch bei R gute Ergebnisse liefern.

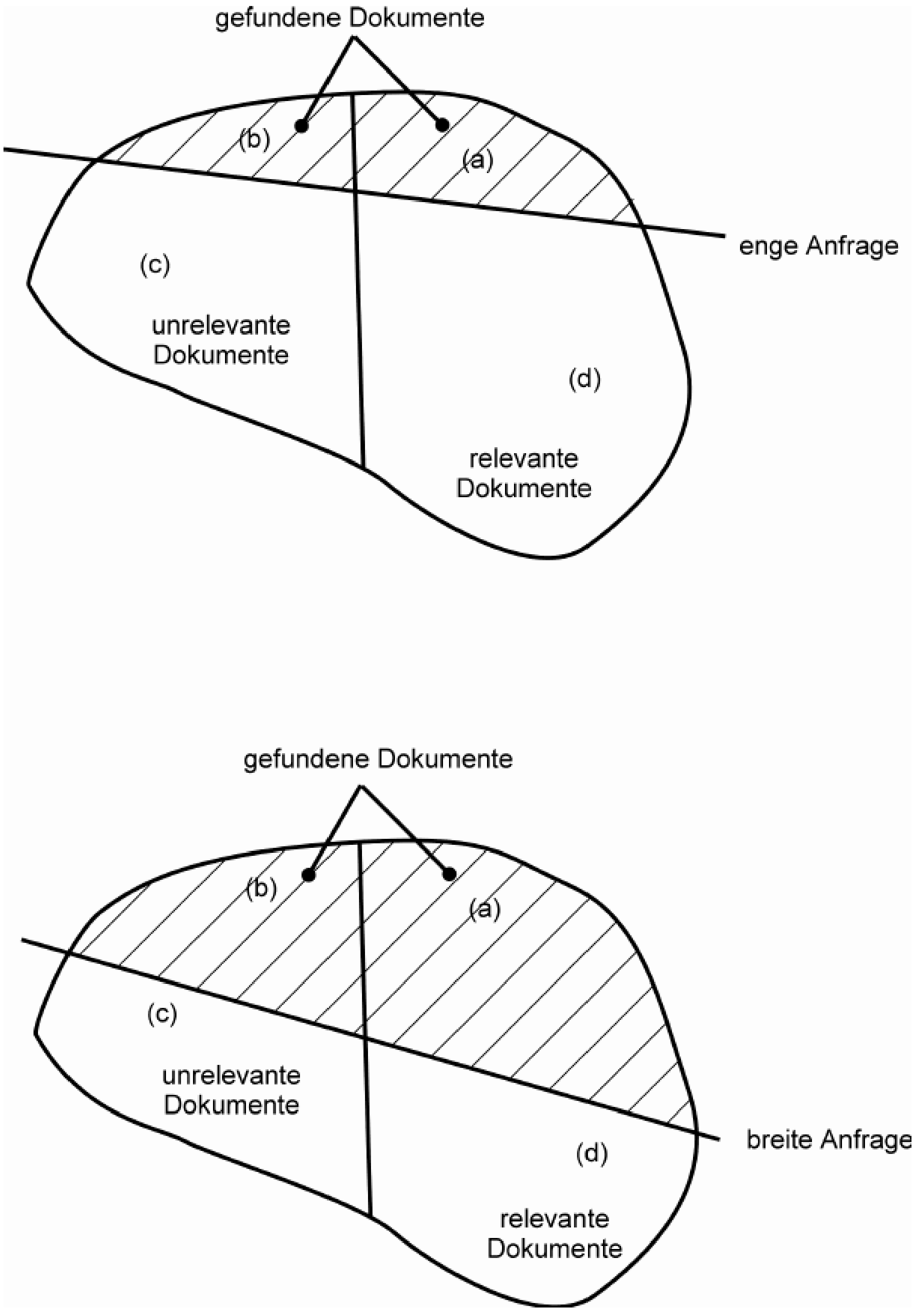


Abbildung 3.7: Vergleich zwischen breiter und enger Anfrage

3.5 Automatische Indexierung

3.5.1 Ziele bei der automatischen Indexierung

In der Vergangenheit wurden die Indexierungsoperationen manuell von Experten oder von trainierten Personen anhand eines vorgegebenen kontrollierten Vokabulars durchgeführt, wodurch eine hohe Qualität erreicht wurde. Bei der automatischen Indexierung ist diese Vorgehensweise schwer anzuwenden. Das Vokabular für die Indexbeschreibungen ist unkontrolliert. Also hat der Benutzer eine größere Freiheit bei der Anfrageformulierung im Vergleich zu Systemen mit kontrolliertem Vokabular. Die Effektivität einer Inhaltsanalyse oder Indexierung wird durch zwei Hauptparameter bestimmt: *indexing exhaustivity* und *term specificity*.
Durch Indexing exhaustivity wird ausgedrückt, in wieweit alle Aspekte der Bedeutung des Textes wirklich in das Indexierungsprodukt eingegangen sind. Wenn die Indexierung erschöpfend ist, dann werden durch die Schlagworte sogar kleinere Aspekte des Textes widergespiegelt. Anderenfalls würde eine Indexierung nur die Hauptaspekte des Inhaltes abdecken.
Die Prägnanz (term specificity) ist mit der Breite der Schlagworte verbunden. Wenn man weit verbreitete, häufig auftauchende Worte für die Indexierung benutzt, dann werden dem Benutzer mehrere nützliche Informationen geliefert, allerdings zusammen mit einer großen Menge unnützer Information. Normalerweise können breite Schlagworte die nützliche Information von der unnützen nicht unterscheiden. Die engen Schlagworte andererseits liefern wenige Ergebnisse, aber der größte Teil davon ist nützlich für den Benutzer.
Es ist gut, wenn man diese Ergebnisse (indexing exhaustivity und term specificity) zusammen mit den früher eingeführten Parametern für Anfrageeffektivität betrachtet. Recall und Precision variieren zwischen 0 und 100 Prozent, d.h. zwischen 0 und 1. Prinzipiell will der Benutzer beides: hohes R und hohe P, also ein großer Teil von den wichtigen Informationen wird geliefert und gleichzeitig werden nur wenige unwichtige Informationen mitgeliefert. In der Praxis aber muß man einen Kompromiß schliessen: also weder recall noch precision dürfen extrem klein sein. Während term specificity entweder hohen R (allgemeine Schlagworte) oder hohe P (spezielle Schlagworte) bewirkt, kann eine Verbesserung der indexing exhaustivity beide Parameter gleichzeitig erhöhen. Praktische Tests von IR - Systemen liefern verschiedene Kombinationen der beiden Parameter: von 80 Prozent Precision und 20 Prozent Recall bis zu 10 Prozent Precision und 80 Prozent Recall. Eine mittlere Leistung, wo Recall und Precision zwischen 50 und 60 Prozent sind, ist in der Praxis aber mehr erwünscht.

3.5.2 Vorgehensweise bei der automatischen Indexierung

Die Zuweisung von Schlagworten in den Dokumenten und Anfragen wird mit dem Wunsch ausgeführt, die für den Benutzer wichtigen von den überflüssigen Dokumenten zu unterscheiden. Bei der automatischen Indexierung geht man folgendermaßen vor:

1. Das Dokument wird als Menge seiner Worte betrachtet.

2. Beseitigung von allgemeinen Funktionsworten, wie z.B. "und", "von", "oder", "aber", die in Texten relativ häufig und gleichmäßig erscheinen.

3. Anhand einer Reihe von Regeln werden die Endungen entfernt und Wörter automatisch auf ihren Stamm zurückgeführt (vgl. 3.3.4).

4. Für jeden gebliebenen Wortstamm T_j in Dokument D_i wird das Gewicht w_{ij} berechnet. Die Häufigkeit tf_{ij} wird für alle Worte T_j in jedem Dokument D_i berechnet. Jedem Dokument D_i werden dann alle Worte T_j zugewiesen, für die gilt:

$$tf_{ij} > T$$

wo bei T die Grenzfrequenz ist. Wie schon erläutert dient eine solche Häufigkeitsbetrachtung nur einem der beiden Hauptziele eines IR-Systems, nämlich nur dem Recall. Ein Wort wie z. B. "Apfel", das häufig in einigen Dokumenten vorkommt, zeigt deutlich, daß die entsprechenden Dokumente sich mit dem Thema beschäftigen. Wenn wir dem Wort "Apfel" großes Gewicht zuweisen, dann wird das dazu beitragen, diese Dokumente schnell bei der Anfrage als Ergebnis zu liefern. Leider sind die häufig vorkommenden Worte nicht für den zweiten Anfrageparameter, die Precision von Nutzen. Voraussetzung für eine gute Präzision ist die Möglichkeit, die einzelnen Dokumente voneinander zu unterscheiden. Ein häufig vorkommendes Wort ist deshalb nur dann zur Indexierung geeignet, wenn seine Häufigkeit nicht dieselbe wie in allen Dokumenten ist. Gute Präzision erreichen wir genau dann, wenn wir die selten vorkommenden Worte als Indexierung nehmen. Solche Worte sind in der Lage, die wenigen Dokumente, in denen sie erscheinen, von den vielen, in denen sie nicht erscheinen, zu unterscheiden. Wenn N die Gesamtzahl der Dokumente ist, dann wird die Dokumentfrequenz df_j als die Anzahl der Dokumente definiert, in denen das Wort T_j erscheint. Daraus wird der invertierte Dokumentfrequenzfaktor (Dokumentdiskriminator) gebildet:

$$idf = log\frac{N}{df_j}$$

Die beiden Betrachtungen kann man jetzt in einem Modell kombinieren und die Schlußfolgerung ziehen, daß die besten Index-Worte für Recall und Precision gleichzeitig diejenigen sind, die häufig in Einzeldokumenten, aber selten in den restlichen Dokumenten vorkommen. So wird das Gewicht eines Schlagworts durch die Formel bestimmt:

$$w_{ij} = tf_{ij} log\frac{N}{df_j}$$

Die Praxis zeigt, daß die Kombination aus Wortfrequenz und Dokumentfrequenz gute Ergebnisse liefert.

5. Jedes Dokument wird dann in der Form

$$D_i = (T_1, w_{i1}; T_2, w_{i2}; ...; T_t, w_{it})$$

   dargestellt
   Ein einfaches System kann mit den bisher beschriebenen Schritten funktionieren. Man kann aber das IR-System auch weiterverfeinern, was zu höherer Qualität bei der Suche führt.

6. Für alle Wortstämme wird der Unterscheidungswert (Diskriminationswert) berechnet.

7. Bei seltenen Wörtern, die schlecht diskriminieren, werden Thesauri verwendet, also Zusammenstellungen von Begriffen, die nach ihrer Bedeutung als Oberbegriffe, Spezialisierungen oder verwandte Begriffe geordnet sind. Eine Anfrage kann breiter gemacht werden, indem die Dokumente statt nach einem Thesaurusbegriff nach dessen Oberbegriff und allen seinen Unterbegriffen durchsucht werden. Sie kann eingeschränkt werden, indem ein spezifischerer Begriff gewählt wird.

8. Für häufige Wörter, die schlecht diskriminieren, werden zusammengesetzte Ausdrücke mit mittelmäßiger Frequenz und großer Diskriminationskraft gebildet. Statt "computer" werden Phrasen wie "computer science" oder "computer program" benutzt.

Kapitel 4

Einsatz von IR-Techniken im WWW

Markus Staudt

4.1 Einführung

Dieses Kapitel beschäftigt sich mit zwei Projekten der Universität von Arizona zum Thema: "Wie finde ich Informationen in großen Datenmengen?" Das eine der beiden Projekte heißt Glimpse und beschäftigt sich damit, Informationen in einem kompletten Dateisystem so zu indizieren, daß ein Auffinden jederzeit möglich ist. Es stammt aus dem Jahre 1993. Das zweite, eine Erweiterung von Glimpse auf das WWW, hat den Namen WebGlimpse und stammt aus neuster Zeit. Sie sollen hier erst einmal vorgestellt werden und dann bezüglich Indizieren, Suchen und Anwendung, und ihrer Leistung verglichen werden. Am Schluß folgt dann noch ein kurzer Ausblick in die Zukunft.

4.2 Vorstellung der Werkzeuge Glimpse und WebGlimpse

4.2.1 Glimpse

Glimpse steht für GLobal IMPlicit SEarch und stellt ein Werkzeug zur Indizierung und zur Suche in Dateisystemen dar [MW94]. Die Neuheit daran ist, daß es einen sehr kleinen Index (meist 2% - 4% des Originals) benutzt und dennoch eine sehr flexible Volltextwiedergewinnung mittels Boolscher Abfragen zuläßt, wobei ein ungefähres Passen (z.B. Zulassung von Fehlschreibung) und sogar die Suche nach regulären Ausdrücken möglich ist.

Notwendig wurde dieses Administrationswerkzeug, weil durch die Explosion der Datenmengen, vor allem in Netzwerken, das Organisieren dieser Datenmengen immer schwerer wird. Denn wer weiß schon nach Jahren noch, wo er Texte abgelegt hatte oder bei weniger wichtigen Texten vielleicht gar von deren Existenz als Datei überhaupt. Um dieses Problem zu lösen, gab es bisher zwei Arten von Werkzeugen zur Suche der Daten. Das eine basiert auf sequentieller Suche im Dateisystem, was nur bei kleineren Datenmengen wirklich schnell arbeitet. Die zweite Art ist die Indexbasierte, welche im allgemeinen einen sehr großen Index benötigt, der vorher berechnet werden muß. Glimpse ist eine Kombination beider Verfahren, indem es einen kleinen Index benutzt, aber auch sequentiell sucht. Die am meisten vorkommende Datenstruktur in IR-Systemen ist der invertierte Index, bei dem jedes Vorkommen jedes Wortes in eine Tabelle mit eben diesem Wort als Wurzel eingetragen wird. Der größte Nachteil daran ist, daß diese Indexe typischerweise in der Größenordnung von 50% - 300% des Orginaltextes an Platz beanspruchen. Dies mag aufgrung des billigen Speichers für große Firmen und Organisationen nicht ganz so dramatisch sein, aber welcher gewöhnliche Benutzer wird seinen Speicher schon verdoppeln, nur um ihn zu indizieren. Infolge dessen sind heute die meisten Dateisysteme unindiziert, wobei sie aber eigentlich groß genug wären, um IR-Fähigkeiten zu benötigen. Kennwortdateien wurden als Alternative vorgeschlagen, da sie nur 10% - 30% des Platzes des Textes brauchen. Da sie aber auf Hash-Verfahren basieren, muß die Wahl der Parameter sehr sorgfältig getroffen werden, um die Fehlgriffswahrscheinlichkeit gering zu halten.

Eine weitere Schwäche der invertierten Indizes ist die Voraussetzung der korrekten Schreibweise. Um falsch geschriebene Worte zu finden, muß dies heute meist von Hand durchprobiert werden. Dies kann dazu führen, daß ein Stück Information, in dem ein Schreibfehler steckt, virtuell verloren ist. Dies wird um so gravierender, je mehr Informationen durch optische Lesegeräte eingescannt werden, da diese Geräte typischerweise eine Fehlerkennungsrate von 1% - 5% haben.

Glimpse benötigt hingegen einen sehr kleinen Index und unterstützt das annähernde Entsprechen. Es erlaubt dem Benutzer, die Zahl der erlaubten Fehler (Einfügungen, Auslassungen, Substitutionen oder irgendeine Kombination daraus) einzustellen, sogar die Möglichkeit das genaueste Entsprechen zu suchen ist realisiert. Durch den kleinen Index ist Glimpse natürlich langsamer als invertierte Indizes, jedoch mit Abfragezeiten im Bereich weniger Sekunden ist es für den Durchschnittsnutzer immer noch schnell genug.

Somit nimmt Glimpse im Vergleich zu invertierten Indizes beinahe die entgegengesetzte Stellung im Vergleich Zeit- zu Speicheraufwand ein. Dies ist in den meisten Systemen kein Problem, da Zeit nichts kostet, jedoch auch in durchsatzintensiven Anwendungen kann Glimpse als Zweitmechanismus eingesetzt werden, um Schreibfehler aufzufinden bzw. sie zu umgehen.

Eingesetzt wird eine Zwei-Ebenen-Suche. Die Idee ist ein Hybrid zwischen einem invertiertem Volltextindex und sequentieller Suche. Dies basiert auf der Beobachtung, daß sequentielle Suche bei Datenmengen bis zu einigen MB schnell genug ist. Deshalb unterteilt Glimpse Dateisysteme in Abschnitte dieser Größe. Jetzt indiziert Glimpse nur das Auftreten eines Wortes in diesem Sektor, auf den ein Zeiger gerichtet ist. Das exakte Lokalisieren wird nun innerhalb diese Bereiches mit sequentieller Suche bewerkstelligt und dann dem Benutzer präsentiert.

4.2.2 WebGlimpse

Browsen und Suchen sind die zwei Hauptparadigmen, um Informationen online zu finden. Das Suchparadigma hat eine lange Geschichte, Suchmöglichkeiten sind in jeder Rechnerumgebung zu finden. Das Browsen ist wesentlich jünger und weniger gebräuchlich, es gewinnt jedoch enorm (und unerwartet) an Popularität durch das World-Wide Web. Beide Arten haben ihre Grenzen. Suchen ist z.B. insofern schwierig, als es genaue Suchanfragen benötigt, um zu relevanten Daten vorzudringen. Die Suchmaschinen erscheinen den meisten Benutzern sowieso als eine einschüchternde Blackbox, deren Arbeitsweise und Inhalte mysteriös wirken. Browsen kann den Inhalt zum Leben erwecken, was den Nutzer in seinem Vorangehen bestätigt. Dennoch ist das Surfen sehr zeitaufwendig und man verliert sehr schnell das ursprüngliche Ziel aus den Augen.

Die beiden Paradigmen werden bisher meist getrennt eingesetzt. So haben sie auf einer WWW-Server Seite einen Link auf eine Suchseite, die es ihnen ermöglicht die Seite zu durchforsten. Aber eine dort ausgeführte Suche wird dem Weg, auf dem man zur Suchsite gekommen ist keine Rechnung tragen, und deshalb das Wissen, das man bereits beim Browsen gewonnen hat nicht einbinden.

Der Grundgedanke bei WebGlimpse [MSG97] ist der, den Benutzern hiermit ein System an die Hand zu geben, das beide Paradigmen, Browsen und Suchen, jederzeit zur Verfügung stellt. Sie sollen damit die Möglichkeit haben, frei umher zu surfen - das übliche Hypertext-Modell - und von jedem beliebigen Punkt aus zu suchen. Die Suche wird hierbei auf Material eingeschränkt, das irgendwie mit dem aktuellen Dokument in Beziehung steht. Eine globale Suche kann natürlich auch durchgeführt werden.

Angenommen man sucht nach Informationen über Netzwerkrecherche bei einer bestimmten Institution. Zuerst wird man wahrscheinlich auf die Homepage zugreifen. Wenn man nun nach “Netzwerk” in der ganzen Institution sucht, wird man zu viele Treffer haben, die das Thema nicht betreffen. Wenn man nach “Netzwerkrecherche” sucht, wird man zu wenige Treffer haben, da das Wort “Recherche” in Recherchedokumenten oft nicht auftaucht (was häufig der Fall ist, weil diese Schlüsselwörter oft aus dem Zusammenhang klar sind).

Wenn man nun aber nur die Seite der Rechercheabteilung, oder die Seiten, die mit Recherche in Zusammenhang stehen, durchsucht, würde die Suche nach ”Netzwerk“ wahrscheinlich relevante Ergebnisse aufzeigen. Den Kontext für die Suche wird man von der Seite bekommen, auf der die Suche gestartet wurde. Solch eine Suche ist auf jeden Fall besser, als sich durch dutzende von Seiten durchzuwühlen. Denn eigentlich sollte doch der Computer die meiste Arbeit leisten und nicht der Benutzer.

Einige einfache Möglichkeiten, die Suche beim Browsen einzuschränken, sind realisiert. So sind z.B. eingeschränkte Such- und Surfmöglichkeiten in GlimpseHTTP eingebaut. Da sich WebGlimpse einige davon ausleiht, sollen sie hier kurz beschrieben werden. GlimpseHTTP indiziert ein Unix-Dateisystem, das eine Suche unterstützt, die auf jeden Teil des Verzeichnisbaumes eingeschränkt werden kann. Bei der ComputerScience-Zitierung (dem meistgenutzten Demo von GlimpseHTTP) kann man in einer der 16 verschiedenen Kategorien browsen und eine Suche auf diesen Teil einschränken. Hierzu wird trotzdem nur ein allgemeiner Index benötigt, denn GlimpseHTTP erkennt selbständig, von wo aus die Suche gestartet wurde, und schränkt diese dann automatisch ein. Die Suchseiten werden von GlimpseHTTP aufgebaut, und sofern die Informationen hierachisch gegliedert sind, arbeitet GlimpseHTTP auch ziemlich gut. Die Such- und Surfmöglichkeiten sind für Seiten mit beliebigen Links oder einer nicht baumartigen Struktur ziemlich ungeeignet. Insofern wurde WebGlimpse die natürliche Erweiterung von GlimpseHTTP, das die Merkmale nicht nur übernimmt, sondern sie deutlich erweitert.

Die Idee der Suche nur in Teilen der Hierachie wird von Yahoo genutzt, wobei nur eine Kategorie zu einer Zeit durchsucht wird. Dies wird in sehr ähnlicher Weise wie bei GlimpseHTTP durchgeführt und leidet unter den gleichen Problemen wie dieses. Das Problem besteht darin, daß ein Objekt manchmal schwer zu katalogisieren ist, weil es z.B. zu zwei Kategorien gehören würde, aber nur in eine eingereiht werden kann.

Andere Websuchmaschinen haben einen anderen Ansatz gewählt. So bieten z.B. Infoseek und Lycos den Menüpunkt: "Finden von verwandten Seiten". Die Verbindung wird dadurch festgelegt, daß die komplette Datenbasis nach Schlüsselwörtern durchsucht wird, die ähnlich sind wie die des aktuellen Dokumentes. Dies ist ein typisches IR-Instrument, das ziemlich effektiv sein kann, manchmal aber auch zu ungewöhnlichen Ergebnissen führt. Die Idee bei WebGlimpse ist es, sich nicht nur die Nachbarschaft zunutze zu machen, sondern auch die Suche nach Wörtern auf diesen zu unterstützen.

Ein weiteres Beispiel für focusierte Suche auf eingeschränkten Bereichen abhängig von der aktuellen Datenbasis ist das WWW-Entrez-System des Nationalen Zentrums für biotechnische Informationen (USA), welches die Nachbarschaft vorher berechnet und diese als festverankerte Links präsentiert. Was daran stört ist, daß die Vorberechnung der Nachbarschaft bei Millionen von Registern Wochen an CPU-Zeit benötigte und Entrez das Suchen in der Nachbarschaft nicht unterstützt.

4.3 Indizieren

4.3.1 Glimpse

Wir gehen davon aus, daß der Informationsraum eine unstrukturierte Sammlung von Textdateien ist. Ein Text wiederum ist eine Abfolge von Worten, die durch die üblichen Trennzeichen wie Leerstellen, Linienende, Komma etc. getrennt sind. Glimpse startet nun mit dem Indizieren, indem es die Gesamtheit aller Daten in kleinere Teile aufteilt, die man Blöcke nennt. Man versucht, daß diese Blöcke annähernd gleich groß sind, was jedoch keine wesentliche Rolle spielt; wichtig ist dabei nur, daß die Anzahl der Blöcke 2^8=256 nicht überschreitet, weil so zur Adressierung eines Blocks ein Byte ausreicht. Diese Art der Aufteilung ist nicht zwingend, erscheint aber sinnvoll.

Jetzt wird die Sammlung Wort für Wort gescannt, im Prinzip wie beim invertierten Index, nur mit der Abweichung, daß nicht jedes Auftreten des Wortes exakt lokalisiert wird, sondern nur das Auftreten des Wortes in einem Block. Es ist also jedes Wort auf dem Index, jedoch nicht unbedingt jedes Auftreten desselben. Im Index stehen also zu jedem

Wort nur die Blöcke, in denen es vorkommt; also wird auch bei mehrmaligem Auftreten in einem Block nur einmal die Blocknummer abgelegt.

Somit kann der Speicherplatz relativ gering gehalten werden. Normale Indizes müssen für jedes Auftreten eines Wortes mindestens ein Wort an Speicherplatz reservieren, manchmal sogar mehr. Deshalb ist hierbei der Speicherplatzbedarf mit dem des Originaltextes vergleichbar. Bei Glimpse wird je Vorkommen innerhalb eines Blocks nur 1 Byte benötigt.

4.3.2 WebGlimpse-Gesamtarchitektur

In aller Kürze arbeitet WebGlimpse wie folgt. Zur Indexierungszeit analysiert es ein vorgegebenes WWW-Archiv (z.B. eine ganze WWW-Server Seite, eine Sammlung spezieller Dokumente oder den privaten Cache an WWW-Seiten), bestimmt die Nachbarschaft (abhängig von der Spezifikation des Nutzers), fügt Suchboxen zu ausgesuchten Seiten hinzu, sammelt alle entfernen Dokumente falls sie relevant sind und cacht diese lokal. Wenn diese Indizierung einmal durchgeführt wurde, können alle Nutzer, die auf dieser Seite gelandet sind, von irgendeiner der hinzugefügten Suchboxen aus suchen und sich dabei auf die Nachbarschaft dieser Seite beschränken (oder natürlich das ganze Archiv scannen). Auf diese Art wandelt WebGlimpse das Archiv in einen einfacher zu navigierenden Hypertext um. Wie der Name schon nahelegt, nutzt WebGlimpse die Glimpse-Suchoperationen, die leicht angepaßt wurden, um ein paar Menüpunkte die für WebGlimpse nützlich sind, hinzuzufügen.

Nur ein Index wird für das Archiv benötigt, das Focusieren der Nachbarschaft und das Einsammeln der entfernten Dokumente wird effizient während der Indizierung durchgeführt, so daß das Suchen immer schnell ist.

WebGlimpse besteht aus vielen Programmen, die fünf Hauptabschnitte ausführen. Die ersten vier werden von der Serveradministration durchgeführt, und nur der fünfte ist dem Benutzer vorbehalten. Die genauere Ausprägung der fünf Hauptschritte ist unten ausführlicher erklärt, hier sei nur ihre Reihenfolge aufgezählt:

1. **Analyse eines gegebenen Archivs**
2. **Sammeln der entfernten Dokumente**
3. **Bestimmen der Nachbarschaften**
4. **Hinzufügen von Suchboxen zu ausgewählten Seiten**
5. **Suchen** siehe Abschnitt 4.4.2

1. Analyse eines gegebenen Archivs

Das Archiv wird von WebGlimpse durch ein Skript mit dem Namen *confarc* aufgebaut. Hierbei wird zuerst der richtige Pfad zum Archiv aufgestellt, die URL festgehalten und der Titel des Archivs und andere administrative Details abgespeichert. Weitere wichtige Parameter in *confarc* sind die Festlegung der URL, von der aus das Durchschreiten des Archivs begonnen werden soll, ob nicht lokale URLs eingesammelt werden sollen oder nicht sowie die Definition der Nachbarschaft. Das Durchschreiten des Archivs wird so gemacht, daß aus jedem HTML-Dokument die Links zu anderen Dokumenten herausgearbeitet werden.

Alle Links werden auf ihre Lokalisierung untersucht (mittels IP-Adressen) und mit der Tiefensuche abgearbeitet (soll eventuell auf Breitensuche geändert werden). WebGlimpse stell flexible Möglichkeiten zur Verfügung, um mittels regulärer Ausdrücke festzulegen, welche URLs bzw. Dateien ins Archiv aufgenommen werden sollen oder welche davon ausgeschlossen bleiben sollen.

2. Sammeln der entfernten Dokumente

WebGlimpse stellt Möglichkeiten zur Verfügung, um entfernte URLs einzusammeln und diese in den Index mit einzubeziehen, und somit auch der Suche zur Verfügung zu stellen. Eine Nachbarschaft kann deshalb auch entfernte Dokumente enthalten und Benutzer können durch eine Nachbarschaftssuche zu entfernten Seiten hüpfen. Dies entspringt der Überzeugung, daß dies wichtig ist und bei lokal begrenzten Suchanwendungen fehlt. Die Hinzunahme entfernter Dokumente bei der Suche hilft dem Benutzer, Verbindungen aufzudecken, die ihm ansonsten wahrscheinlich nie aufgefallen wären. Durch diese Option wird WebGlimpse in etwa zu einer globalen Hypertextsuchmaschine.

Die entfernten URLs, die WebGlimpse einsammelt, werden in seperaten Dateien abgelegt, die mit der URL verbunden werden. Somit kann das Indizieren und das Suchen transparent gemacht werden Der Orginalinhalt der URL wird dabei nicht aufgegeben. Falls jedoch ein Platzproblem besteht, können diese Spiegelbilddateien jederzeit entfernt werden und die Suche bleibt dennoch möglich, da sie bereits in den Index eingearbeitet sind, und immernoch ein Verweis auf die Original-URL vorhanden ist, was ja die typische Ausgabe der meisten Suchmaschinen im Web ist. Festgehalten werden soll hier noch, daß WebGlimpse nicht als Roboter ausgelegt ist, der rekursiv entfernte URLs aufsucht und einsammelt. Dies kann, falls man diese Option benötigt, durch andere externe Programme erledigt werden.

3. Bestimmen der Nachbarschaft

Beim Durchschreiten des Archivs berechnet WebGlimpse auch die Nachbarschaften. Zur Zeit werden zwei Arten der Nachbarschaft unterstützt. Die erste besteht aus allen Seiten, die einen bestimmten Abstand (z.B. max. 2 Links entfernt) von der aktuellen Seite aus nicht überschreiten. Diese Art der Nachbarschaft ist eigentlich implizit, da Links meist auf Seiten weisen, die ähnlichen Inhalt haben.

Die zweite Art der Nachbarschaft ist noch einfacher, denn sie umfaßt nur alle Unterverzeichnisse und alles, was darunter ist. Die Implementierung der Nachbarschaftssuche ist zwar einfach, da die Liste aller Dateinamen, die eine Nachbarschaft zu einer Seite ausmachen, in einer Datei gespeichert sind (auch die von entfernten URLs).

Wenn eine Suche ausgeführt wird, durchsucht Glimpse zuerst den Hauptindex, findet die Liste aller Dokumente mit relevanten Treffern und schneidet diese mit der Nachbarschaftsliste der zugehörigen Seite. Die Nachbarschaftsliste wird während der Suche aufgenommen, und der Index ist unabhängig von der Nachbarschaft. So kann z.B. ein unwichtig gewordenes Dokument einfach aus der Liste der Nachbarschaften gelöscht werden.

In der aktuellen Version von WebGlimpse werden die Nachbarschaftsdateien zu den Seiten komprimiert gespeichert, um Platz zu sparen. Die Nachbarschaftsdateien werden von WebGlimpse zur Suchzeit dekomprimiert, so daß die Suchzeiten dennoch schnell bleiben.

Es wird auch an anderen Arten der Nachbarschaft gearbeitet, die dem Benutzer mehr Möglichkeiten zur Definition eigener Typen geben sollen. Eine Idee ist es, z.B. mehrere Nachbarschaften für eine Seite zuzulassen, wobei dennoch nur eine Nachbarschaftsdatei abgelegt wird und die Nachbarschaftstypen als Mengen darauf definiert sind. Somit kann zur Abfragezeit darüber entschieden werden, welcher Nachbarschaftstyp verwendet werden soll.

4. Hinzufügen von Suchboxen zu ausgewählten Seiten

Um die Konzepte von Suchen und Browsen zu unterstützen, fügt WebGlimpse ausgesuchten Seiten automatisch eine kleine Suchbox hinzu. Der HTML-Code zum Integrieren der Suchbox ist in einer einfachen Datei gespeichert, die vom Administrator schnell seinen Bedürfnissen angepaßt werden kann. Die Suchboxen werden während des Indiziervorgangs hinzugefügt, indem ihr HTML-Code an den der ausgewählten Seite anghängt wird. Die Suchbox wird dann unten an die bestehende Seite anghängt und kann jederzeit mit einem WebGlimpse-Werkzeug wieder entfernt werden. Normalerweise werden solche Suchboxen an alle Seiten angehängt, die eine nichtleere Nachbarschaft haben. Hier können aber genauso Seiten ein- bzw. ausgeschlossen werden, wie sie von der Nachbarschaft ein- bzw. ausgeschlossen werden können. Durch das Hinzufügen einer solchen Suchbox zu allen Seiten wird die Anzahl der Kontextwechsel verringert und hilft dabei, daß der Nutzer auf seinem Weg bleibt. Denn nur so können die Benutzer den Inhalt der Seite im Auge behalten, während sie eine neue Suchanfrage zusammenstellen.

Jede dieser Suchboxen enthält einen dynamischen Link zu weiteren Suchoperationen. Er ist in der Art dynamisch, daß diese Zusatzoptionen zur Laufzeit aufgebaut werden und die Nachbarschaft mit einbeziehen (welche übrigens auch zur Laufzeit der Abfrage erst generiert werden). Diese erweitereten Suchoptionen enthalten Glimpseoperationen, wie "Fallsensitivität" etc. genauso wie die Möglichkeit, direkt in die Zeile zu springen, in der der Treffer liegt. Eine weitere Möglichkeit der Suche ist die, nur nach Dateien zu suchen, die in den letzten X Tagen geändert wurden. Ein Nachteil der Suchboxen ist, daß hierzu jede Seite geändert werden muß. Die Änderungen sind jedoch so minimal, daß diese in Kauf genommen werden können.

Entwursentscheidungen

Hier sollen kurz die Hauptentwurfsentscheidungen vorgestellt werden. Wie bereis erwähnt, sind die vier Ziele: schnelles Suchen, effizientes Indizieren, flexible Möglichkeiten der Nachbarschaftsdefinition und schonender Umgang mit den Internetresourcen.

Die erste Entwursentscheidung ist die für feste Nachbarschaften, die zur Indexierungszeit festgelegt werden müssen. Es gibt heute jede Menge Diskussionen über Suchroboter, die das Web automatisch nach speziellen Informationen absuchen. Das Web ist aber zur

Zeit nicht schnell genug und die Bandbreite zu beschränkt, um eine große Zahl solcher Roboter zu verkraften. Durch den WebGlimpse-Entwurf legt nicht der Benutzer sondern der Administrator fest, was die Nachbarschaft ist, und somit muß die Nachbarschaftsberechnung eben nur einmal gemacht werden.

Die Möglichkeit, die Nachbarschaft, die man will, selbst festzulegen, ist sehr wichtig. Deshalb wurde Wert darauf gelegt, die Nachbarschaftsdefinition und den Gebrauch des restlichen Programms strikt zu trennen. So werden die Nachbarschaftsdateien erst am Ende des Suchprozesses einbezogen, und der Index ist völlig unabhängig von Nachbarschaften.

Schnelles Suchen kollidiert indirekt immer mit effizientem Indizieren. Je mehr Zeit- und Speicheraufwand man beim Indizieren verwendet, je schneller wird die Suche sein. WebGlimpse ist für kleine bis mittlere Archive entwickelt worden und legt mehr Wert auf effizientes Indizieren, was bei dieser Archivgröße immer noch schnelles Suchen zuläßt. Indizieren kann dennoch ziemlich lange dauern, vor allem, wenn viele entfernte Dokumente eingesammelt werden müssen oder eine zu große oder komplexe Nachbarschaft bestimmt werden muß.

Die Benutzerschnittstelle ist wichtig für jede Suchanwendung. Diese ist hier mit "hervorheben der Schlüsselwörter", dem Einbeziehen der Linien mit Treffern und der Anzeige des Datums der letzten Änderung ziemlich gut gelöst.

Die Entscheidung die Suchbox allzeit verfügbar zu machen, im Vergleich zum typischeren Link zu einer Suchseite, ist eine gute Idee. Die Suchbox benötigt nicht viel Platz und erlaubt es dem Benutzer, Suchanfragen zu formulieren, solange er die aktuelle Seite noch vor sich hat.

4.4 Suchen und Anwendung

4.4.1 Glimpse

Die Suchroutine besteht wie schon erwähnt aus zwei Phasen. Zuerst wird der Index nach den Blöcken, in denen das Wort auftritt, abgesucht. Danach wird jeder Block extra abgearbeitet. Obwohl man die erste Phase mit hashen oder B-Bäumen beschleunigen könnte, entschied man sich bei Glimpse für Agrep (ein Suchprogramm der gleichen Forscher aus früheren Tagen), weil es mehr Flexibilität zuläßt. Beim Hashen oder bei B-Bäumen könnte man nur nach Schlüsselwörtern, die vorher ausgewählt wurden, suchen. Mit Agrep kann man die ganze Kraft der sequentiellen Suche ausnutzen, da der Index klein ist.

Agrep ist im Gebrauch fast wie jedes andere sequentielle Suchprogramm, außer daß es viel offener und allgemeiner ist. Es läßt die Suche nach Mustern mit einer bestimmten Anzahl an erlaubten Fehlern zu, es kann benutzerdefinierte Register, wie z.B Mails oder Paragraphen, ausgeben und nicht nur einzelne Zeilen. Es unterstützt boolsche Abfragen, Wildcards, reguläre Ausdrücke und viele andere Optionen. Wenn nun das Muster gegeben ist, wird zuerst mit Agrep im Index die Liste aller Wörter gesucht, die dem Muster entsprechen und dannach werden wiederum mit Agrep die zugehörigen Blöcke nach dem genauen Auftreten abgesucht. Die gleiche Prozedur läuft auch bei den komplizierten Mustern mit

Wildcards etc. ab. Diese komplizierten Muster werden von anderen Suchroutinen nicht unterstützt.

Boolsche Abfragen werden ähnlich abgearbeitet wie bei den normalen invertierten Indizes. Nehmen wir z.B. an, die Suchabfrage lautet "Muster1" AND "Muster2", so wird zuerst die Liste aller Blöcke für jedes der Muster gesucht, dann werden diese miteinander geschnitten. Danach werden alle Blöcke der Schnittmenge abgesucht. Beachten sollte man hierbei jedoch, daß selbst wenn Blöcke gefunden werden, in denen "Muster1" und "Muster2" vorkommen, die Suche nicht unbedingt erfolgreich sein muß, da die beiden Muster eventuell in verschiedenen Teilen des Blocks liegen können. Außerdem kann man Agrep natürlich auch in normalen invertierten Indizes benutzen, um die Vorteile ausnutzen zu können.

Als Fazit folgt hier eine Liste der Stärken und Schwächen von Glimpse im Vergleich zu invertierten Indizes:

Stärken:

1. Sehr kleiner Index

2. Annäherndes Passen wird unterstützt

3. Schnelle Konstruktion des Index

4. Kein Bedarf, Dokumentgrenzen im voraus festzulegen. Man kann dies noch während der Abfrage erledigen.

5. Einfach an eigene Bedürfnisse anzupassen

6. Parallelrechnerfähigkeit (versch. Blöcke können von versch. Prozessoren abgesucht werden)

7. Einfache Änderung des Index, da er so klein ist. Deshalb können auch dynamische Texte verarbeitet werden.

8. Kein Bedarf, Wortstämme zu extrahieren, denn Teilwortabfragen werden automatisch unterstützt.

9. Abfragen mit Wildcards, Klassen von Zeichen sowie regulären Ausdrücken werden unterstützt.

Schwächen:

1. Langsamer als invertierte Indizes. Nicht geeignet für Systeme bei denen Schnelligkeit die wichtigste Rolle spielt.

2. Zu langsam bei sehr umfangreichen Datenmengen (mehr als 500 MB).

3. Boolsche Abfragen mit zu allgemeinen Worten sind zu langsam.

Glimpse wurde auf vielen Systemen zwischen 30 MB und 250 MB eingesetzt. Hier sollen nun einige Merkmale von Glimpse vorgestellt werden und die Erfahrungen hiermit.

Glimpse ist eine Sammlung von Programmen in Umfang von etwa 8000 Zeilen C-Code, die darauf ausgerichtet sind, Dateisyteme oder Teile davon zu indizieren. Der Befehl

`glimpse_index [-n] dir_name [dir_names]`

indiziert das Verzeichnis dir_name, optional auch mehrere, außerdem optional [-n] auch Zahlen. Indiziert wird normalerweise nachts, wobei ca. 7-8 Sekunden je MB benötigt werden. Bevor eine Datei indiziert wird, überprüft Glimpse, ob die Datei eine Textdatei ist. Falls die Datei zu viele nicht alphanumerische Zeichen enthält (z.B. komprimierte Dateien oder Bilder), wird sie nicht indiziert, und der Dateiname wird einer .log-Datei hinzugefügt. Festzustellen, ob eine Datei eine Textdatei ist oder nicht, ist nicht einfach, so z.B. in Sprachen, in denen Zeichen wie Umlaute verwendet werden. So wurden Programme in Glimpse eingefügt, die diese Entscheidung treffen, z.B. Essence. Auf der anderen Seite gibt es natürlich ASCII-Dateien, die nicht indiziert werden sollen, so z.B. DNA-Sequenzen, denn durch ihre große Anzahl nahezu zufälliger Bruchstücke, wächst der Index ins unermeßliche. Das Indizieren von Zahlen ist gebräuchlich für Daten und andere signifikante Zahlen, jedoch große Dateien mit numerischen Daten werden den Index nur unnötig wachsen lassen. Außerdem bietet Glimpse dem Benutzer die Möglichkeit, Dateien vom Indizieren auszuschließen, z.B. persönliche Dateien, indem man sie in eine .prohibit-Datei aufnimmt.

Die Aufteilung in Blöcke erfolgt auf dem direkten Weg, indem der Computer den Umfang der zu indizierenden Daten bestimmt und daraus die Größe eines Blocks herleitet. Danach werden Dateien so zusammengestellt, bis der Block die gewünschte Größe erreicht hat, dann wird der nächste Block begonnen. Hier sollen zwei Entwicklungen für die Zukunft vorangetrieben werden: Die Aufteilung in Blöcke sollte der ursprünglichen Organisation der Daten mehr entsprechen und dem Benutzer soll die Möglichkeit eingeräumt werden, das Aufteilen in Blöcke zu kontrollieren.

Die Benutzerschnittstelle von Glimpse ist der von Agrep sehr ähnlich, bis auf den Unterschied, daß kein Dateiname mehr benötigt wird. Die normale Verwendung ist

`glimpse information`

was alle Zeilen aus den indizierten Texten ausgibt, die "information" enthalten.

`glimpse -1 HardToSpell`

wird alle Auftreten von "HardToSpell" finden, mit bis zu einem Buchstabierfehler.

`glimpse -w 't[a-z]j@#uk'`

wird alle dreistelligen Emailadressen finden, die mit t beginnen und j enden und bei denen uk im Host-Namen steht. (Bei Agrep steht # für eine beliebige Anzahl an Wildcards und -w stellt die Bedingung, daß das Muster einem kompletten Wort entsprechen muß).

Glimpse unterstützt Boolsche Abfragen, so wird

`glimpse -1 'Lazoka;Zahorian'`

alle Auftreten beider Namen auffinden, (wobei es einen Schreibfehler zuläst, was hier aber nicht übertrieben erscheint). Der Strichpunkt dient als AND.

`glimpse 'Winter Usenix'`

wird zuerst denn Index getrennt nach beiden Worten absuchen und alle Blöcke finden in denen beide Worte vorkommen. In der zweiten Phase sucht Agrep die Blöcke nun nach der gesammten Phrase ab.

Eine weitere Möglichkeit bei Glimpse ist die Suche auf Dateien einzuschränken, die einem vorgegebenen regulären Ausdruck entsprechen.

`glimpse -F 'Heuhaufen\.c$' Nadel`

wird alle Nadeln in allen Heuhaufen.c Dateien finden.

4.4.2 WebGlimpse

WebGlimpse benutzt zur Suche Glimpse, d.h. alle oben beschriebenen Funktionen stehen auch in WebGlimpse zur Verfügung. Deshalb soll hier besonders auf die Art der Ausgaben eingegangen werden. Die Ausgabe auf eine Suchanfrage ist eine Menge von Registern, jeweils eines für eine Datei, in der der Treffer ist. WebGlimpse formatiert die Ergebnisse in vier Arten:

Zusammenhang für jeden Treffer

Der Titel jeder URL, in der der Treffer ist und ein Link auf sie, sowie alle Zeilen, in denen das gesuchte Wort auftaucht, werden ausgegeben.

Unterstützen der Zeilennummern

Da Glimpse die genaue Zeile findet und WebGlimpse dies auch ausnutzt, kann direkt auf die Zeile gesprungen werden, in der der Treffer liegt. Dies geschieht durch das Einfügen der richtigen Brücken zur Laufzeit, was nicht ganz so einfach ist, da manche Links neu berechnet werden müssen.

Herausheben der Schlüsselwörter

Alle passenden Schlüsselworte werden herausgehoben, indem sie sowohl in den Ausgaberegistern als auch in den Zeilen fett ausgegeben werden.

Anzeige des letzten Änderungsdatums

Glimpse verarbeitet seit neustem das Datum zu jeder Datei, und zeigt dieses auch an.

Die Hauptanwendung für WebGlimpse ist natürlich in Hypertextdateisammlungen zu suchen. Hier sollen noch andere artverwandte Anwendungen kurz angedacht werden.

* **Aufbau persönlicher oder thematischer Sammlungen:** WebGlimpse erlaubt es dem Benutzer, relevante Dokumente zu sammeln. Er erhält sogar eine Link-Struktur. So können mit WebGlimpse relativ einfach Hypertext-Bücher geschrieben werden, welche alle Links einsammeln und indizieren werden.

* **Gedächtnis oder Cachedateien:** Die meisten WebBrowser cachen die Seiten, die sie aufsuchen. Diese Sammlung meist doch relevanter Daten kann sehr hilfreich sein. Wenn man diese nun -außer sie nur zu cachen- auch noch indiziert und organisiert, so kann auf einer persönlichen Datensammlung eine Volltextsuche unterstützt.

* **Visualisierung und Anpassung:** Wenn man WebGlimpse mit Grafikpaketen verbindet, wird die Struktur des Hypertextes einfacher überschaubar. So können die Ergebniszeilen von Suchvorgängen z.B. mit Icons versehen werden, die einfacher zu nutzen sind.

4.5 Leistung

4.5.1 Glimpse

Alle Leistungsdaten hier sind Erfahrungswerte. Die Frage die hier beantwortet werden soll ist nicht die des Vergleichs mit anderen Systemen, sondern die, festzustellen, daß Glimpse schnell genug ist innerhalb seines Anwendungsbereiches. Suchzeiten hängen sehr stark von der Art der indizierten Daten und der Suchabfrage ab, um die Architektur und das Betriebssystem mal außen vorzulassen. Falls das gesuchte Muster nur in einem Block auftritt, wird die Suchzeit immer relativ kurz sein unabhängig davon, wie groß die Datenmenge ist oder wie kompliziert das Muster selbst sein mag. Die schnellstmögliche Anfrage ist diejenige, welche erfolglos bleibt, da hier nur der Index durchsucht werden muß.

Je mehr Treffer eine Abfrage hat, destso langsamer wird sie werden. Also muß bei einer Suche in großen Datenmengen das Abfrageschema sorgfältig gewählt werden. Egal wie schnell die Suche bewerkstelligt wird, wird es bei einer großen Anzahl von Treffern zulange dauern, diese zu sichten.

Dennoch kann eine allgemeine Abfrage auch Vorteile haben, weil man schnell merkt, daß es viele Treffer gibt, so daß man die Suche stoppen kann, um die Abfrage genauer zu formulieren. Zu diesem Zweck gibt es die -N Option, die dem Benutzer anzeigt, wieviele Blocks gescannt werden müssen.

Hier nun eine Liste von Laufzeiten (auf einer DEC 5000/240 Workstation) für ein persönliches Dateisystem, das 69 MB Text in 4296 Dateien enthält. Es braucht 4,9 Minuten CPU-Zeit (9 Minuten echter Zeit), das Dateisystem zu indizieren, und der Index hat 1,9 MB Platzbedarf, was 2,7% des Orginaltextes entspricht. Eine typische Suche dauert 2-10 Sekunden, z.B. die Suche nach “biometrics” dauert 1,6 Sekunden, eine nach “incredible” dauert 2,8 Sekunden (mit 27 Treffern in 22 Blöcken). Eine Suche nach “Chescoslavia” mit 2 erlaubten Schreibfehlern dauert 3,6 Sekunden. Eine Suche nach “Usenix” dauert 13,9 Sekunden mit 93 Treffern in 70 Blöcken, jedoch die Suche nach Usenix und Winter nur 4,5 Sekunden mit 19 Treffern. Also kann eine Boolsche Abfrage durch Verringerung der zu durchforstenden Blöcke zur Schnelligkeit beitragen.

Für eine wesentlich größere Datenmenge, wurde eine Kopie des ganzen Bibliothekskataloges der Universität von Arizona indiziert. Hierbei wurden 2,5 Millionen Bände in 440 Dateien mit zusammen 258 MB indiziert. In 18 Minuten war der Katalog auf 8 MB indiziert, was 3,1% der Originalgrö" se entspricht. Auch hier bewegen sich die Suchzeiten in der Größenordnung weniger Sekunden.

Diese Erfahrung kann ich bestätigen, da man über
http://glimpse.cs.arizone.edu bzw. http://www.old.com/glimpsehelp.html
diese Programme ausprobieren kann.

4.5.2 WebGlimpse

Die Erfahrungen mit WebGlimpse sind noch ziemlich gering. Hier nun aber dennoch einige Zahlen, die man erhalten hat, als man WebGlimpse in einem komplexen System eingesetzt hat.

Das Archiv, das hier genutzt wird, ist die Site des Gesetzesinformationsdienstes von Arizona, welches Informationen über den Etat, Ausschüsse, Statuten, die Verfassung, Börsennachrichten, Tagesordnungen und die kompletten Texte von Gesetzesentwürfen umfaßt. Dieses Archiv belegt 152MB in über 20.000 Dateien. Die Indizierungsphase dauert 3 Stunden 5 Minuten bei einer Nachbarschaftstiefe bis zu 3 Links. (Davon 1 Std. zur Berechnung der Nachbarschaften, 1 Std. um alle HTML-Dokumente zu analysieren und ihnen die Suchboxen hinzuzufügen, 22 Min. das ganze zu indizieren und 41 Minuten alle Nachbarschaftsdateien zu komprimieren.)

Das Archiv belegt nach Indizierung 205,8 MB, also 34% mehr; diese verteilen sich annähernd gleichmäßig auf den Index, die (komprimierten) Nachbarschaften und die hinzugefügten Suchboxen.

Die Suchzeiten belaufen sich im Rahmen weniger Sekunden (2-12), wovon die Hälfte wiederum echte Suchzeit wist und die andere Hälfte zur Erstellung der HTML-Dokumente der Ausgabe benötigt wird.

Wer es probieren will, hier die Adresse:

http://glimpse.cs.arizone.edu/webglimpse

4.6 Ausblick in die Zukunft

4.6.1 Glimpse

Suche in komprimierten Texten

Falls der Text in komprimierter Form vorliegt, muß er dekomprimiert werden, bevor er sequentiell durchsucht werden kann. Dies wird die Durchführung erheblich verlangsamen. Deshalb werden Werkzeuge entwickelt, die es ermöglichen, auch komprimierte Texte zu durchsuchen. Hierbei wird das Suchmuster in eine komprimierte Form gewandelt, um sich das Dekomprimieren des Textes zu sparen. Diese Art des Durchsuchens beschleunigt die Suche, genauso wie sie Platz spart. Dieses Werkzeug ist inzwischen realisiert und scheint alle daran gestellten Ansprüche zu erfüllen.

Wachsende Indizes

Der 2-stufige Index ist einfacher zu ändern als der invertierte Index, weil er wesentlich kleiner ist. Um eine neue Datei dem aktuellen Index hinzuzufügen, wird sie zu einem existierenden Block hinzugefügt oder es wird ein neuer Block angefangen, falls sie groß bzw. wichtig genug ist. Danach wird der Index gescannt und jedes Wort, das in dem Block bisher nicht vorkam, wird diesem hinzugefügt. Da der Index klein ist, dauert es nicht allzu lange, ihn von der Platte zu lesen und überarbeitet wieder abzulegen. Das Löschen einer Datei dauert länger, da hier für jedes Wort geprüft werden muß, ob es nicht noch an anderer Stelle im Block vorkommt und deshalb nicht gelöscht werden sollte.

Anpassungen

Es gibt einige Wege, wie der Benutzer die Indizier- und Suchzeit verkürzen kann, indem er Anpassungen vornimmt. So sollte der Nutzer die Möglichkeit haben, zu entscheiden, ob bestimmte Muster indiziert werden sollen oder nicht, so z.B. Zahlen. Eine Möglichkeit ist, eine Schnittstelle zur Verfügung zu stellen, an der der Nutzer mit eigenen Filterprogrammen festlegen kann, was indiziert werden soll, je nach Inhalt, Dateiart, Dateinamen usw. Außerdem soll dem Nutzer die Möglichkeit gegeben werden, eine Menge von mehrwortigen Phrasen zu definieren, die dann gemeinsam indiziert werden. Eine weitere Option besteht darin, die Datenmenge aufzuteilen und diese Kategorien dann getrennt zu indizieren. All diese Anpassungen sind der Schlüssel zu schnelleren und damit besseren Suchmöglichkeiten.

4.6.2 Folgerungen

Das Suchen im Netz wächst schnell aus dem Kindheitsstadium heraus. Die aktuellen Möglichkeiten der Werkzeuge sind erstaunlich und ziemlich hilfreich, dennoch sind sie nicht das letzte Wort zum Thema Suchen und gar weit entfernt von wirklich "guten“ Lösungen. Das Auffinden von nützlichen Informationen im Web ist immernoch ein ziemlich frustrierender Vorgang. Es sollten deshalb noch mehr Möglichkeiten ausprobiert werden, die als Prototyp zur Verfügung gestellt werden könnten. Glimpse und WebGlimpse stellen solche Versuche dar, sie sind einfach, simpel aufgebaut und flexibel, so daß das Erweitern möglich ist. Glimpse und WebGlimpse sind Teile des FUSE (Find and USE)-Projekts an der Universität von Arizona.

Kapitel 5

Datenbanksysteme als Information Retrieval Systeme

Heiko Bauer

5.1 Einleitung

Zwei Trends begründen die Aktualität dieses Themas: Zum einen versucht man immer mehr beim Systementwurf bestehende Software zu berücksichtigen, sie als Basis zu verwenden und kundenspezifische Anpassungen via offener Schnittstellen individuell zu realisieren. Bekanntestes Beispiel, wenngleich auf einem anderem Sektor, ist die SAP AG mit ihrer betriebswirtschaftlichen Standardsoftware R/3. auen. Einzig und allein die s sie nicht neu bauen. Einzig und allein die Zum anderen wächst durch das Internet der Bedarf und die Anforderungen an Information Retrieval Systeme. Man mußs versuchen dem wachsenden Informationspool Herr zu werden, Verfahren zur Verfügung stellen, die über effiziente und zugleich sichere Suchmechanismen verfügen.
Unter diesem Aspekt hat man darüber nachgedacht, Standard-DBMS als Basis für Information Retrieval Systeme einzusetzen und den Teil der individuellen Entwicklung auf die Benutzerschnittstelle zu reduzieren.

5.2 Informationssysteme

5.2.1 Verschiedene Arten von Informationssystemen

Informationssystemen ist gemein, daß sie in irgendeiner Weise Informationen speichern und Benutzern zeitversetzt wieder zugänglich machen. Sie unterscheiden sich im wesentlichen in den Inhalten, der Art der Speicherung und des Auslesens. Es folgt eine grobe Beschreibung der heute gängigsten Arten von Informationssystemen.

Information Retrieval Systeme (IRS)

Bei der in IRS gehaltene Informationen handelt es sich um Dokumente. Eingelesen werden Texte in natürlicher Sprache, ausgegeben Referenzen auf im System abgelegte Dokumente, die bestimmten Suchkriterien genüge leisten. Die Ergebnismenge enthält keinesfalls ausschlielich das oder die gesuchten Dokumente, sondern vielmehr eine Auswahl von Texten die möglicherweise interessant sein könnten. Benutzer solcher Systeme verfolgen - vergleichbar den Besuchern einer Bibliothek - ganz unterschiedliche Interessen.

Datenbankmanagementsysteme (DBMS)

Auch DBMS befassen sich mit dem Ablegen, Auslesen und Verändern von Informationen in einem Datenspeicher. Allerdings liegen hier die Daten in einer explizit vorgegebenen Form vor, nicht als Text in natürlicher Sprache. Im Beispiel eines relationalen Datenbankmanagementsystems (RDBMS) werden Informationen in Tabellen mit festgeschriebenen Feldern eingefügt. Bei einer Abfrage spezifiziert der Benutzer Werte, die in bestimmten Feldern enthalten sein müssen. Die Sätze in der Ergebnismenge erfüllen exakt die vorgegebenen Auswahlkriterien. Die Hauptaufgaben eines DBMS sind neben Select-/Insert-/Update-/Delete-Funktionen das Bereitstellen einer hohen Systemsicherheit, die Kontrolle nebenläufiger Transaktionen und die Wiederherstellbarkeit der Daten bei auftretenden Fehlern.

Management-Informationssysteme (MIS)

Ein MIS ist ein DBMS zugeschnitten auf die Bedürfnisse der Geschäftsführung. Aufbauend auf einer Datenbasis werden spezielle Funktionalitäten zur Verfügung gestellt, die es dem Management ermöglicht Entwicklungen in einem Unternehmen zu verfolgen, dem Produktivsystem übergeordnet Einblicke zu erhalten, ganz allgemein die Informationen zu beschaffen, die benötigt werden, um ein Unternehmen zu führen.

Decision-Support-Systeme (DSS)

DSS haben die Aufgabe, Informationen in einem Unternehmen aus unterschiedlichen Datenquellen zu sammeln und als Basis für Entscheidungsfindung aufzubereiten. Solche Systeme enthalten oft Komponenten der Planung, die Prozesse in die Zukunft projizieren und die Ergebnisse für heutige Entscheidungen heranziehen. Es gibt hier keine klare Trennung zwischen den zwei vorher beschriebenen Systemen; denkbar ist an dieser Stelle auch eine Kombination verschiedener Informationssysteme.

Question-Answering Systeme (QAS)

Die Idee eines QAS ist, einem Rechner eine Frage in natürlicher Sprache zu stellen und eine Antwort zu erhalten. Die gestellte Frage wird analysiert und mit dem gespeicherten Wissen verglichen. Solche Systeme befinden sich in der Entwicklung. Die Hauptprobleme bestehen darin, aus einer Ansammlung von Begriffen in natürlicher Sprache eine Bedeutung zu extrahieren.

5.2.2 Vergleich IRS vs. DBMS

Im folgenden sollen IRS und DBMS genauer betrachtet werden: An welchen Stellen liegen Berührungspunkte, wo die Unterschiede.

Gehandelte Informationen

IRS Bei einem IRS besteht eine klare Vorstellung darüber, welche Informationen behandelt werden sollen: Dokumente und Informationen über Dokumente. Ziel ist es, eine Sammlung von Texten systematisch abzulegen und auf eine vom Anwender spezifizierte Abfrage Referenzen auf diese Texte auszugeben. Dabei wird unabhängig von den Inhalten ein Mechanismus gesucht, der dieses leistet.

DBMS Die in einem DBMS gehaltenen Informationen richten sich nach dem implementierten Datenmodell. Dies können im Einzelfall Kunden, Lieferanten, Aufträge, Arbeitsgänge genauso sein wie Meßseihen, Analyseergebnisse, usw., d.h. ein DBMS ist zunächst in seiner Ausrichtung völlig frei. Nachdem die Entwurfsentscheidungen realisiert worden sind, ist die Struktur festgeschrieben, das Datenbasisschema, in der Daten abgelegt werden. Im Gegensatz zum IRS besteht hier ein klarer Zusammenhang zwischen der Struktur der Information in der Datenbasis und den Inhalten, genauer, die Struktur ergibt sich aus den gewünschten Inhalten.

Konzeption der Systeme

IRS IRS sind in den meisten Fällen von Anfang an individuell entwickelte Systeme. Sie arbeiten in aller Regel nach dem Prinzip des Inverted File Index (IFI). Der IFI funktioniert nach folgendem Schema: Zunächst werden Referenzen auf Dokumente als ein Array von indizierten Records angelegt. Dies entspricht einer Matrix mit den Dokumenten als Zeilen und Begriffen als Spalten. Erscheint ein Begriff in einem Dokument, so wird dieses Feld in der Matrix markiert. Ist diese Indizierung abgeschlossen, so wird die Matrix invertiert, d.h. von nun an werden die Begriffe in den Zeilen und die Dokumente in den Spalten aufgezeigt. Bei der Anfrage, in welchen Texten ein Begriff b nun vorkommt, wird die Zeile b nach Markierungen durchsucht, und alle markierten Einträge werden zurückgegeben. Während in IRS die Funktionen zum Ablegen und Suchen erst noch entwickelt werden müssen, sind diese und einige andere mehr in einem DBMS schon vorhanden.

DBMS Ein DBMS ist zunächst eine leere Hülle, die über Funktionalitäten verfügt, eine Datenbasis anzulegen und zu administrieren. Produktiv einsetzbar wird das DBMS allerdings erst durch Anlage einer individuellen Datenbasis. Grundlage für die Implementierung einer Datenbank ist ein Datenbankentwurf, der einen Ausschnitt aus der realen Welt abbildet. Dieser Entwurf wird in ein konkretes, z.B. relationales Datenmodell transformiert. Im Datenmodell wird eine Struktur festgelegt, die dann für alle Operationen verbindlich gegeben ist.

Funktionalität

IRS Die Forderungen an ein IRS sind klar und schnell umrissen. Es mu Dokumente ablegen und wiederfinden können, es mußs Mechanismen zur Verfügung stellen, die adhoc Abfragen bearbeiten und die Ergebnisse bewerten können.

DBMS Ein DBMS bietet über das bloße Einlesen und Manipulieren von Daten hinausgehende Fähigkeiten:

- Wiederherstellbarkeit der Daten nach einem Systemausfall
- Kontrolle von konkurrierenden Zugriffen durch Sperren von Daten
- Authorisierung der Benutzerzugriffe
- Constraints zur Erzwingung der Datenintegrität
- Transaktionsverwaltung
- Berichtsgeneratoren
- ...

Motivation zur Integration

Die obigen Betrachtungen zeigen, daß es sehr wohl Berührungspunkte zwischen IRS und DBMS gibt. Muß ein IRS immer völlig neu entwickelt werden, oder kann man es vielleicht auch auf einem Standardprodukt, einem DBMS aufbauen, und vielleicht sogar noch von dessen umfangreicher Funktionalität und Sicherheit profitieren? Kann man bei der Realisierung eines IRS auf bestehende ausgereifte Produkte, die sich am Markt bewährt haben, also umfangreich Tests im Echtbetrieb bestanden haben, zurückgreifen? Abbildung 5.1 zeigt ganz grob die Funktionsweise eines Information Retrieval Systems. Wenn man von der Eingabe, den Dokumenten in natürlicher Sprache abstrahiert, behandelt ein IRS Stichworte, Titel, Autoren, Referenzen auf Dokumente, Zugriffshäufigkeiten, fügt Objekte in ein vorhandenes Datenvolumen ein und fragt sie ab. Können das die Datensätze in einem DBMS sein?

Die beiden folgenden Kapitel belegen mit Beispielen den möglichen Einsatz von DBMS in IRS. Kapitel 3 zeigt theoretische Ansätze des IR in relationalen Datenbasen [Bla88]. Überlegungen zu folgenden Fragen werden angestellt: Welche Relationen werden benötigt,

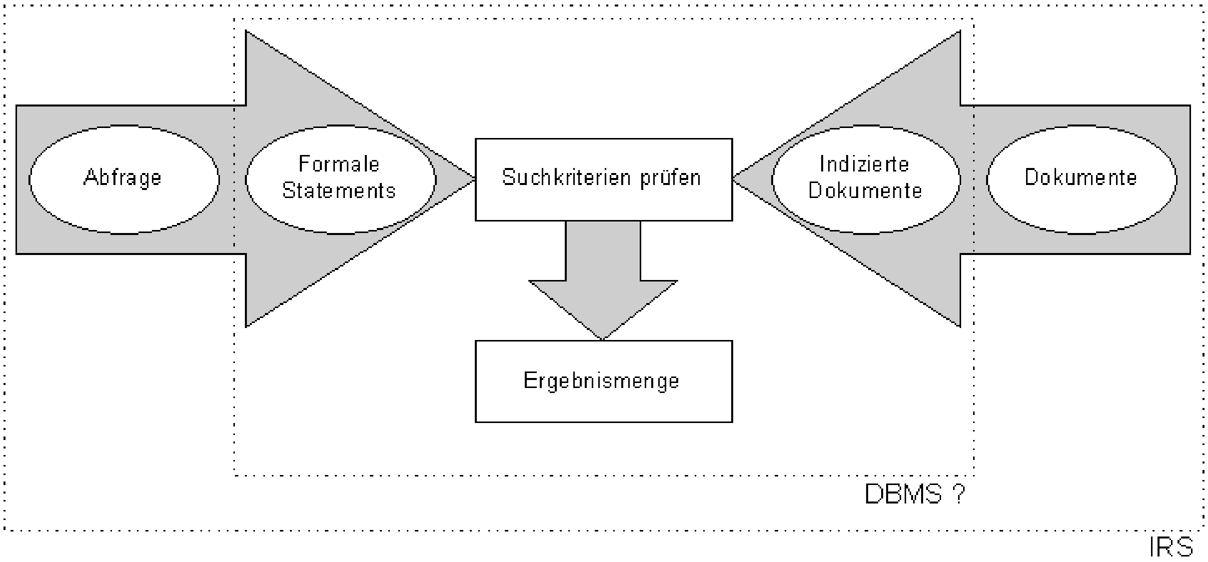


Abbildung 5.1: Funktionsweise eines Information Retrieval Systems

um IR-Funktionalität anbieten zu können? Welche Mechanismen erleichtern eine Einschränkung der Ergebnismenge? Kapitel 4 gibt konkret ein Anwendungsbeispiel eines IRS basierend auf einer objektorientierten Datenbank [BCC+94]. Dabei wurde im Department of Computer Science, University of Massachusetts, in ein bestehendes IRS ein OODBMS als Kern implantiert.

5.3 IRS auf Basis RDBMS - konzeptionelle Überlegungen

Eine wesentliche Anforderung an ein IRS ist die Fähigkeit, adhoc auf Benutzeranfragen Ergebnisse zu liefern. Das Ziel hierbei ist in den meisten Fällen gleich: die Identifikation von Dokumenten, die den Wünschen der Anwender entsprechen. Die Anfragen an das System dagegen erweisen sich als äußerst heterogen, z.B. kann nach einem bestimmten Autor, einer Zeitspanne, einem Titel, einem Thema, einem Verlag, usw. gesucht werden wollen. Vom System sind daher flexible Mechanismen zur Verfügung zu stellen, die es erlauben, diese Anfragen auf einfache Art und Weise zu formulieren. Neben dem herkömmlichen Text Retrieval ist es wünschenswert, Metainformationen auswerten zu können, z.B. wieviele Artikel gab es in den letzten zwei Wochen zum Thema t, mit welchen Themen beschäftigt sich Autor a vorzugsweise, usw.

5.3.1 Basisrelationen

Die folgenden Relationen dienen im Beispiel als Basis, dem IRS eine Grundfunktionalität zu ermöglichen. Von den Attributen in Klammern bezeichnet das kursiv markierte den

Primärschlüssel.

- Referenz (*dokument_id*, titel, dokument_art, veröffentlichungs_datum, zeitschrift_name, ausgabe_nr, seite, ausgabe_datum)
- Kurzbeschreibung (*dokument_id*, kurzbeschreibung)
- Autor (*dokument_id*, name)
- Institution (*name*, institution, art, adresse, telefon_nr)
- Zeitschrift (*zeitschrift_name*, veroeffentlicher)

Typische Abfragen für ein RDBMS mit diesem Inhalt könnten sein: Wie heißen diejenigen Titel, die von Autor a geschrieben wurden? Wie lauten die Kurzbeschreibungen der Artikel, die von Autor a stammen? Wo arbeitet Autor a? Welches sind die Artikel, die in der letzten Ausgabe der Zeitschrift z erschienen sind?

5.3.2 Relation Stichwort

Die oben aufgeführten Anfragen beziehen sich mehr auf beschreibende Attribute für die Dokumente, als auf die Dokumente mit ihren Inhalten selbst. Um den letztgenannten Aspekt, die Inhalte, im Datenmodell abbilden zu können, muß eine Relation

- Stichwort (*dokument_id*, *stichwort*, gewicht)

hinzugefügt werden. Diese Relation spezifiziert Stichworte zu Dokumenten und mißt ihnen über eine Gewichtung eine relative Bedeutung bei.

Bsp.: Gewichtung von Stichworten
(312, Computer, 0,5)
(312, Programmierung, 0,3)
(312, Datenbankmodellierung, 0,9)

Diese drei Tupel besagen, daß im Dokument 312 die Themen Computer, Programmierung und vor allem aber Datenbankmodellierung behandelt werden. In dieser Relation gibt es keine Obergrenze für die Anzahl der abgelegten Stichworte pro Dokument, erfahrungsgemäß bewegt sich die Zahl der Einträge zwischen sechs und zehn. Eines der größten Probleme in IRS ist eine zu umfangreiche Ergebnismenge. Über die Gewichte kann man eine Priorisierung der Ausgabe erzwingen, bzw. Resultate unterhalb einer gesetzten Schwelle per Selektion ignorieren:

Bsp.: Abfrage mit Berücksichtigung von Gewichten

```
SELECT *
FROM   Referenz
```

```
WHERE dokument_id IN
      (SELECT dokument_id
      FROM    Stichwort
      WHERE   stichwort='Stichwort A'
      AND     weight >= 0.7
      ORDER BY weight DESCENDING)
```

5.3.3 Ableitbare Informationen

Ein relationales IRS beinhaltet mehr als nur Dokumente. Es werden auch Informationen über die Dokumente vorgehalten, die per SQL-Funktionen ausgewertet werden können. Zu welchen Themen werden Artikel vorzugsweise in welchen Zeitschriften veröffentlicht, welcher Autor schreibt bei welchen Blättern; auch Trends, wie Veränderungen und Entwicklungen der Ausrichtung der Inhalte eines Magazins im Laufe der Zeit o.ä., können abgefragt werden. So können beispielsweise zu einem bestimmten Stichwort diejenigen Zeitschriften ermittelt werden, die über Artikel zu diesem Thema verfügen. Die Ergebnismenge wird nach der Anzahl der Artikel pro Zeitschrift sortiert:

Bsp.: Abfragen von Metainformationen

```
SELECT DISTINCT journal_name, COUNT(dokument_id)
FROM   Referenz
WHERE  dokument_id in
       (SELECT DISTINCT dokument_id
       FROM    stichwort
       WHERE   stichwort='Stichwort A')
GROUP BY journal_name
ORDER BY COUNT(dokument_id) DESCENDING
```

5.3.4 Assoziative Suche mit der Relation Thesaurus

Assoziative Suchmechanismen erlauben, semantische Zusammenhänge innerhalb von Dokumenten aufzudecken. Eine einfache Variante davon ist, zu überprüfen, welche Stichworte zusammen in einem Text erscheinen, vielmehr zu hinterfragen, wie groß ist die Wahrscheinlichkeit unter der Voraussetzung des Vorkommens des Worts X, für ein Vorkommen des Worts Y. Um diese Funktionalität behandeln zu können, wird die Relation Thesaurus (stichwort, co_term, prozent) eingeführt. Man betrachtet alle Texte, die das Stichwort A enthalten, und ermittelt aus den weiteren Stichworten B die Häufigkeit ihrer nachgeordneten Erscheinung. Daraus ergibt sich der Prozentsatz. Ein Ziel dieser Relation ist es, die Basis einer Suchabfrage nach Stichworten zu verbreitern, indem man zunächst zu einem Stichwort die zumeist im gleichen Dokument erscheinenden Wörter ermittelt, um sie bei weiteren Abfragen mitzuverwenden. Das Beispiel sucht nach den Stichworten, die mit einer Wahrscheinlichkeit von mehr als 4 % in einem Text vorkommen, der schon das 'Stichwort

A' enthält:

Bsp.: Assoziative Suche

```
SELECT co_term, prozent
FROM   Thesaurus
WHERE  stichwort='Stichwort A'
AND    prozent > 0.04
ORDER BY prozent DESCENDING
```

Solche Abfragen unterstützen außerdem den Indizierungsprozeß derart, daß bei einem einzufügenden Dokument nur das Hauptstichwort identifiziert werden muß. Danach kann über die Thesaurus-Relation eine Liste von nachgeordneten Termen ermittelt werden, aus der weitere Stichworte für das Dokument abgeleitet werden können.

5.3.5 Boolsche Suche

Stichwort oder Themensuche verwenden oft boolsche Operatoren, um eine Anfrage genauer zu spezifizieren. Per SQL realisiert man diese Anforderung durch das Verknüpfen (Join) der Stichwort-Tabellen mit sich selbst.

Bsp.: Konjunktive Abfrage

```
SELECT titel
FROM   Referenz
WHERE  dokument_id IN
       (SELECT dokument_id
        FROM   Stichwort S1, Stichwort S2
        WHERE  S1.stichwort = 'a'
        AND    S2.stichwort = 'b'
        AND    S1.dokument_id = S2.dokument_id)
```

Bsp.: Disjunktive Abfrage

```
SELECT titel
FROM   Referenz
WHERE  dokument_id IN
       (SELECT dokument_id
        FROM   Stichwort
        WHERE  stichwort in ('a', 'b'))
```

Man kann eine allgemeine Form für boolsche Abfragen in SQL entwickeln. Dazu transformiert man den Selektionsterm in konjunktive Normalform und wendet folgendes Konstrukt an:

```
SELECT FIRST.dokument_i
FROM   Stichwort FIRST,
       Stichwort SECOND,
       ...
       Stichwort Nth
WHERE  FIRST.dokument_id=SECOND.dokument_id
AND    SECOND.dokument_id=THIRD.dokument_id
...
AND    FIRST.stichwort in ('a', 'b', ...)
AND    SECOND.stichwort in ('a', 'b', ...)
AND
...
```

Theoretisch ist auch eine Anfrage auf Basis disjunktiver Normalform denkbar; sie ist jedoch wesentlich komplexer zu formulieren. Im relationalen Schema erreicht man einen großen Teil der Flexibilität und Transparenz durch die Normalisierung. Es entstehen viele kleine anstatt weniger großen Relationen. Möchte man diese Tabellen für bestimmte Abfragen wieder zusammenbringen, bezahlt man dies mit teuerer Laufzeit. Diese Leistungseinbußen treten ganz deutlich auf, wenn - wie im obigen Beispiel - große Relationen mehrfach mit sich selbst durch Joins verbunden werden müssen. Man kann sich vorstellen, daß mit einem solchen Modell Abfragen mit zehn angegebenen Stichworten untragbar sind.

5.3.6 Sichten

Relationale Systeme bieten u.a. als Anfrageerleichterung das Sichtenkonzept. Man definiert eine virtuelle Relation, die einen Ausschnitt des Datenbankschemas präsentiert. So beziehen sich Abfragen dann anstatt auf mehrere reale, auf eine virtuelle Relation, d.h. die Abfrageschnittstelle wird für diesen speziellen Fall leichter überschaubar. Man wird sich dieses Konzeptes natürlich nur dann bedienen, wenn die virtuelle Relation, also eine bestimmte Abfrage, regelmäßig gebraucht wird. Man kann in einer View ein Fenster auf eine Menge bestimmter Dokumente mit definierten Inhalten geben.

Bsp.: Erzeugen einer View

```
CREATE VIEW dokument_ab AS
            (SELECT *
             FROM   Referenz
             WHERE  dokument_id IN
                    (SELECT dokument_id
                     FROM   Stichwort
```

WHERE stichwort IN ('Stichwort A', 'Stichwort B')))

Bei der Abfrage bezieht man sich dann auf die virtuelle Relation, die nur Dokumente dieser Art enthält und selektiert vielleicht noch auf ein aktuelles Zeitfenster.

Bsp.: Einfachere Abfragekonstrukte nach Definition einer View

```
SELECT *
FROM    dokument_ab
WHERE datum >=' 01.01.1997'
```

Ein Bonus des Sichtenkonzeptes ist, daß kein physikalischer Speicherplatz in der Datenbank verbraucht wird. Lediglich die Definition wird im Data Dictonary abgelegt.

5.4 IRS auf Basis OODBMS - ein Anwendungsbeispiel

Das im folgenden vorgestellte Projekt geht zurück auf die University of Massachusets. Das bestehende IRS Inquery benutzt zur Realisierung seines Inverted-File-Index-Konzepts B-Bäume. Diese werden hier ersetzt durch das objektorientierte Datenbankmanagementsystem Mneme.

5.4.1 Vorüberlegungen

Ziel dieses Unterfangens war zu prüfen, ob es möglich und sinnvoll ist, ein IRS auch auf einem OODBMS fußen zu lassen. Man hat versucht, die Aufgaben des Inverted File Index von Inquery auf Mneme abzubilden. Grundsätzlich gibt es in einem IRS drei Operationen: Einfügen, Suchen und Verändern. Suchen ist davon die am meisten Verwendete, wenn man also versucht das System zu optimieren, dann sollte man hier ansetzen. Die Größe einer invertierten Liste ist abhängig von der Anzahl der Vorkommnisse des Terms in der Dokumentensammlung. Bildet man eine Rangfolge der Begriffe mit der Häufigkeit ihres Auftretens absteigend geordnet, so wird man feststellen, daß es eine Konstante gibt, die etwa dem Produkt von 'Platz in der Hitliste' und 'Anzahl der Einträge' entspricht. Man wird außerdem erkennen, daß ungefähr die Hälfte der Terme nur ein- oder zweimal auftreten. Mit einer Verteilungsfunktion, die den Prozentsatz derjenigen Terme ermittelt, deren zugehörige invertierte Liste kleiner oder gleich einer oberen Grenze in Byte ist, werden Kennzahlen festestellt, die für die Dimensionierung von Mneme von Bedeutung sind (s.u.). Ungefähr 50% der Listen befinden sich unterhalb einer Grenze von 12 Byte, die größten, etwa 3% über 4 Kbyte. Hier wird schon deutlich, daß das Vorgehen bei dieser Integration in keinster Weise einem konventionellen Datenbankentwurfsprozeß entspricht.

5.4.2 Inquery

Der Flaschenhals eines IRS ist das Auslesen und Bewerten von Dokumenten. Inquery arbeitet intern mit einer Hash-Tabelle und dem IFI. Die Hash-Tabelle bildet die vorkommenden Worte auf eindeutige term_ids ab. Außerdem werden hier die Anzahl der Zugriffe pro Wort mitgeschrieben. Der IFI verwendet die term_ids als Schlüssel in einem B-Baum. Pro Record existiert eine term_id. Der Record hält in seinem Header die Anzahl der Zugriffe, danach die Liste der zugehörigen Dokumente und die Stelle, an der der Term wiedergefunden werden kann. Der Record wird als komprimierter Vektor bestehend aus Integer-Werten gehalten. Bei einer Abfrage liest Inquery den ganzen Record pro Term und vermengt das Resultat mit den der zuletzt gelesenen Terme.

5.4.3 Mneme

Mneme kennt keine vordefinierte Datentypen, sondern nur Gruppen von benachbarten Bytes, denen ein eindeutiger Schlüssel zugewiesen ist. Zusätzlich besteht die Möglichkeit einem Objekt Referenzen auf andere Objekte mitzugeben. Objekte werden in Files gruppiert. Die Schlüssel sind nur eindeutig innerhalb eines Files; sollten mehrere Files gleichzeitig geöffnet sein, so wird übergeordnet und dynamisch ein weiterer Schlüssel vergeben. Man klassifiziert Objekte außerdem in physikalische Segmente, diejenigen Einheiten, die zwischen der Festplatte und dem Hauptspeicher übertragen werden, und logisch in Pools. Pools definieren die Größe der physikalischen Segmente, die Konstruktoren, das Objektformat und wo Objekte in einem File abgelegt werden. Durch die Entwicklung von neuen Poolroutinen wird das System den individuellen Bedürfnissen angepaßt. Das Basissystem beinhaltet eine Menge von Funktionen zum Implementieren von Poolroutinen, sowie einige Standardpoolroutinen. Logische Segmente, die 255 Objekte zusammenfassen, ermöglichen den Zugriff. Dabei wird über eine Hash-Tabelle, in der sich die Objekt-Schlüssel befinden, geprüft, ob sich ein Objekt im Hauptspeicher befindet.

5.4.4 Das integrierte System

Mneme erzeugt für jede invertierte Liste im B-Baum ein Objekt. Der Mneme Index wird in die Hash-Tabelle von Inquery eingetragen. Bei einer Anfrage wird der Index ausgelesen und das gewünschte Objekt angefordert. Basierend auf den obigen Vorüberlegungen wurden drei Klassen von Objekten zur Abbildung von invertierten Listen geschaffen.
Small Object Pool: 50% der invertierten Listen hatten eine Größe kleiner gleich 12 Byte. Geht man von 16 Byte-Objekten aus (+4 Byte Verwaltungsoverhead), ergibt sich bei 255 Objekten eine Größe für ein logisches Segment von 4 KB.
Large Object Pool: Einige invertierte Listen sind so groß, daß es sich nicht lohnt, sie zu clustern. Alle invertierten Listen über 4 KB erhalten ihr eigenes pysikalisches Segment zugewiesen.
Medium Object Pool: Alle diejenigen Listen, die zu klein oder nicht groß genug für eine der beiden anderen Klassen sind, werden in 8KB große physikalische Segmente aufgeteilt.

Jeder dieser drei Gruppen wird ein eigener Puffer zugewiesen, so daß der globale Puffer optimal aufgeteilt werden kann. Die Pufferstrategie ist Last Recently Used mit einer kleinen Abweichung. Bei einer Abfrage in Inquery, wird intern ein Baum aufgebaut, der, bevor die eigentliche Abarbeitung beginnt, durchgesehen wird, um Objekte, die von der Anfrage benötigt werden, vor einer Verdrängung zu bewahren.

5.5 Bewertung

DBMS ja oder nein? Wie sooft lautet auch hier die Antwort: vielleicht! Dafür sprechen die Sicherheit dieser erfahrenen Systeme, Mechanismen, die eine Wiederherstellbarkeit der Daten gewährleisten. Durch einen sauberen Entwurf in einem Datenmodell erreicht man einen hohen Grad an Transparenz und damit eine leichtere Wart- und Skalierbarkeit. Hier wird auch klar, daß der abzubildende Ausschnitt der Welt für die Realisierung in einem DBMS wie prädestiniert ist. Es stehen sehr flexible Abfragewerkzeuge zur Verfügung. Außerdem existieren diese Systeme schon, man muß sie nicht neu bauen. Einzig und allein die Performanz gibt den Ausschlag dafür, daß IRS immer noch "from the scratch" entstehen.
Im relationalen Bereich wird es, wie oben schon erwähnt, Probleme geben, wenn Abfragen Boolsche Operatoren nachbilden. Durchsucht man innerhalb einer Abfrage die Datenbank nach mehreren Stichworten, so erhöht jeder weitere Suchbegriff die Anzahl der zu prüfenden Sätze multiplikativ um den Faktor |Stichwort|. Desweiteren sind wir bisher von sechs bis zehn Stichworten pro Dokument ausgegangen. Man kann sich leicht vorstellen, daß eine Indizierung aller Begriffe in Texten den Rahmen sprengt.
Trotz allem werden RDBMS heute in IRS eingesetzt. Wenn beispielsweise die Anforderungen nur eine Abfrage mit maximal zwei Stichworten gleichzeitig vorsehen, zudem vielleicht schon ein RDBMS beim Kunden vorhanden ist, wird man sich auch für eine Realisierung in der relationalen Welt entscheiden.
Bei der Integration von Mneme in Inquery war die Aufgabenstellung keineswegs ein IRS auf Basis eines OODBMS aufzubauen, sondern umgekehrt unter ein bestehendes System Mneme unterzusetzen. Das OODBMS sollte ausschließlich den Inverted File Index abbilden. Es gab also keinen objektorientierten Entwurf, der dann umgesetzt wurde in ein Datenbasisschema. Demzufolge wird hierbei der Aspekt der Transparenz nicht erfüllt. Mneme sind zwar gewisse objektorientierte Züge nicht abzusprechen, dennoch erfüllt es nicht die Kriterien für ein klassisches DBMS. Aus diesem Grund ist der Beispielcharakter im Zusammenhang "OODBMS und IRS" fragwürdig, wobei andere Unternehmungen in diese Richtung in der objektorientierten Welt nicht stattgefunden haben, bzw. zumindest nicht veröffentlicht wurden.

Kapitel 6

Datenmodelle und Anfragesprachen für semi-strukturierte Daten

Ulrike Zintz

6.1 Problembeschreibung und Lösungsansätze

6.1.1 Einführung

In den letzten Jahren hat die Flut von Daten, die in kommerziellen Datenbanken verwaltet werden, immer stärker zugenommen, so daß der Mensch nicht mehr in der Lage ist, diese Daten zu interpretieren und zu verarbeiten.

Daraus entstanden Forschungsrichtungen, die man z.B. mit den Begriffen *Data Mining* oder *Data Warehouse* verbindet. Durch die weltweite Vernetzung sieht man sich einer neuen Herausforderung gegenübergestellt:
Wie kann aus der Fülle von Daten, die über das *World Wide Web* verfügbar sind, die gewünschte Information (genauer die gewünschten Daten) herausgefiltert werden?

Die Daten liegen in unterschiedlichen Formen vor, von unstrukturierten Daten (einfache Textdateien) bis hin zu Daten die in Datenbanksystemen (also strukturiert) gespeichert sind. Es ist bekannt, daß über Anfragesprachen (z.B. SQL) Daten aus strukturierten Datenbeständen selektiert werden können. Probleme entstehen erst, wenn die Datenbestände, in denen gesucht wird, keine vorgegebene oder bekannte Struktur aufweisen. Man unterscheidet zwischen *raw data*, d.h. Daten, die keine Struktur aufweisen (z.B. Bilder) und Daten in DB-Systemen, denen ein Datenbasis-Schema zugrunde liegt. Dazwischen liegen die sogenannten *semi-strukturierten Daten*, die weder unstrukturiert noch tabellenorientiert (Relationales Modell) oder typisiert sind. Eine solche Definition ist sicherlich nicht präzise genug, um genaue Aussagen darüber zu treffen, ob bestimmte Daten unstrukturiert, semi-strukturiert oder strukturiert sind.

6.1.2 Gliederung

Die folgenden Abschnitte zeigen die Probleme auf, die sich beim Umgang mit semi-strukturierten Daten ergeben. In Abschnitt 6.1.3 wird der Begriff der semi-strukturierten Daten anhand eines Beispiels erläutert. Im folgenden Abschnitt 6.1.4 werden mögliche Datenmodelle und Anfragesprachen für semi-strukturierte Daten mit ihren Vor- und Nachteilen vorgestellt. Im Kapitel 6.2 wird eine speziell für semi-strukuturierte Daten entwickelte, Anfragesprache vorgestellt.[1]

6.1.3 Der Begriff *semi-strukturierte Daten*

Die folgenden Beispiele sollen verdeutlichen, was unter semi-strukturierten Daten zu verstehen ist:

Einführende Beispiele

In wissenschaftlichen Veröffentlichungen ist die Angabe eines Literaturverzeichnisses eine Selbstverständlichkeit. LaTeX unterstützt durch das Programm *BibTeX* die automatische Generierung einer Literaturdatenbank, die aus einem oder mehreren Files, den sogenannten *BibTeX files* besteht. Die allgemeine Syntax für einen Literatureintrag in der Datenbank lautet:

Eingabetyp{Schlüsselwort,
Zwingende Felder
Feldname = {Feldwert},Feldname = {Feldwert},...
Optionale Felder
Feldname = {Feldwert},Feldname = {Feldwert},...
}

Ein BibTeX file gehört wegen seiner Strukturierung zu den semi-strukturierten Daten. Trotz der vorhandenen Struktur können ohne Vorbearbeitung keine Anfragen auf diesen Daten durchgeführt werden. Durch die Möglichkeit, gewisse Felder nicht zu belegen (optionale Felder), besitzen die Einträge eines *BibTeX files*, obwohl strukturiert, nicht alle die gleiche Struktur. Trotz dieser *Unregelmäßigkeit* wäre es durchaus möglich, jeden Eintrag in einem Tupel abzuspeichern, um Anfragen auf diesen Daten zu ermöglichen. Dieses würde aber voraussetzen, daß alle unbelegten Felder mit NULL aufgefüllt werden, was einen erhöhten Speicherbedarf zur Folge hätte.

Das World Wide Web bietet ebenfalls zahlreiche Beispiele semi-strukturierter Daten. Web-Seiten z.B. besitzen ein vorgegebenes Format, das den Benutzer zwingt, sich in der Gestaltung einer Seite an gewisse Regeln zu halten. Trotz dieser auferlegten Struktur

[1]Die vollständige Syntax der Anfragesprache ist in [ADM96] zu finden

werden Web-Seiten zu den semi-strukturierten Daten gezählt, da keine Struktur ähnlich der in Datenbanksystemen vorliegt.

Merkmale semi-strukturierter Daten

Aus den obigen Beispielen lassen sich einige typische Charakteristika semi-strukturierter Daten ableiten::

- Die Daten besitzen keine einheitliche Struktur.
  Da die Daten aus unterschiedlichen Quellen stammen, besitzen sie unterschiedliche Strukturen, auch wenn sie eigentlich den gleichen Gegenstand oder Sachverhalt repräsentieren.

- Die Struktur der Daten ist implizit (Beispiel: BibTeX files, HTML Dokumente).
  Obwohl die Daten keine Struktur ähnlich der in relationalen oder objektorientierten Datenbanken aufweisen, kann aus den Daten mit Hilfe geeigneter Parser eine Struktur abgeleitet werden, d.h. die Daten sind implizit strukturiert.

- Die Struktur ist nur teilweise vorhanden.
  Charakteristisch für semistrukturierte Daten ist die Tatsache, daß die aus den Daten abgeleitete Struktur nicht immer vollständig ist. Dies' hat wiederum zur Folge, daß die Datensätze nicht die gleiche Struktur aufweisen, wodurch das Extrahieren von Daten aus der Datenbank wesentlich komplizierter ist.

- Das Datenbasis-Schema ist sehr groß (wächst mit der Größe des Datenbestandes).
  In klassischen Datenbanksystemen, wird die Struktur der Daten durch das Datenbank-Schema vorgegeben. Daten die dieser Strukturierung nicht entsprechen werden nicht in die Datenbasis aufgenommen. Schemaerweiterungen erfolgen nur durch speziell dafür vorgesehene Befehle (DDL) und können nicht durch Daten erzwungen werden. Datenbanksysteme die heterogene Daten aufnehmen, müssen in dieser Hinsicht flexibler sein, indem sich das Schema an die Daten anpaßt (d.h. entsprechend erweitert wird) und nicht umgekehrt.

- Das Schema entwickelt sich sehr schnell.
  Im Gegensatz zu Standard Datenbank-Systemen, wo eher selten Änderungen am Schema vorgenommen werden, ist das Schema eines Datenbanksystems für semi-strukturierte Daten einem ständigen Anpassungsprozeß unterworfen.

- Es existiert keine Typisierung der Daten.
  Eine weitere Folge der Heterogenität der Datenbestände ist die unterschiedliche Typisierung der Daten. So z.B. wird ein Merkmal in einem Datensatz als String und in einem anderen Datensatz als Integer definiert. Dadurch entstehen, wie bereits oben erwähnt, Schwierigkeiten bei der Formulierung von Anfragen.

- Es existiert keine strikte Unterscheidung zwischen Datenbasis-Schema und Daten. Durch die Anpassung des Schemas an die eingefügten Daten ist eine solche Unterscheidung nicht mehr notwendig.

6.1.4 Datenmodelle und Anfragesprachen

Problemstellung

Liegt einem Datenbanksystem ein *klassisches* Datenmodell zugrunde, so existieren Anfragesprachen, um auf relativ einfache Art und Weise die benötigten Daten anzufragen. Steht z.B. ein relationales Datenbanksystem zur Verfügung, können mit Hilfe des Datenbankschemas und geeigneter SQL-Statements, gezielt Daten aus der Datenbank selektiert werden. Hat man es hingegen mit semi-strukturierten Daten zu tun, kommen die folgenden Fragen auf:

- Welches Datenmodell eignet sich für semi-strukturierte Daten?
- Wie muß eine Anfragesprache für semi-strukturierte Daten aussehen? Welchen Anforderungen muß sie genügen?

In den folgenden Abschnitten werden zwei Ansätze vorgestellt.

Strukturgewinnung: Ziel ist es, die semi-strukturierten Daten in einer Form abzuspeichern, die es erlaubt, auf bereits bekannte Datenmodelle und Anfragesprachen zurückzugreifen.

Angenommen, es sollen Dateien so abgespeichert werden, daß eine möglichst effiziente Suche auf den Daten möglich ist. Da die Daten eine implizite Struktur aufweisen (Titel, Autor, Erscheinungsjahr usw.) besteht die Möglichkeit, eine Relation (Tabelle) mit den entsprechenden Attributen anzulegen. Die einzelnen Tupel werden aus den Dateien konstruiert, wobei auch auf Duplikateliminierung und ähnliche Probleme geachtet werden muß. Für die nicht vorhandenen Attributwerte werden in den entsprechenden Tupeln die Felder mit NULL belegt. Neben den erwähnten Attributen enthalten die Tupel Verweise auf Ausschnitte der Dateien, aus den sie *entstanden* sind. Dadurch geht derjenige Teil der Daten, der keine Struktur aufweist nicht verloren und kann über Verweise erreicht werden.

Auf diese Weise ist es möglich, eine relationale Datenbank aufzubauen, und darauf SQL-ähnliche Anfragen auszuführen. Abbildung 6.1 verdeutlicht, daß die relationale Datenbank als *strukturierter Einstiegspunkt* für das Durchsuchen semi-strukturierter Daten angesehen werden kann.

Data Guide: Der Ansatz des *Data Guide* geht von der Annahme aus, daß die Daten sehr unterschiedlich strukturiert sind, sowohl was die Anzahl der *Attribute* anbelangt, also auch bzgl. der Typisierung der einzelnen Attribute. Dadurch ist der Ansatz der Strukturgewinnung nur beschränkt einsetzbar.
Ein Data Guide enthält beispielsweise Informationen über die *möglichen* Attribute eines

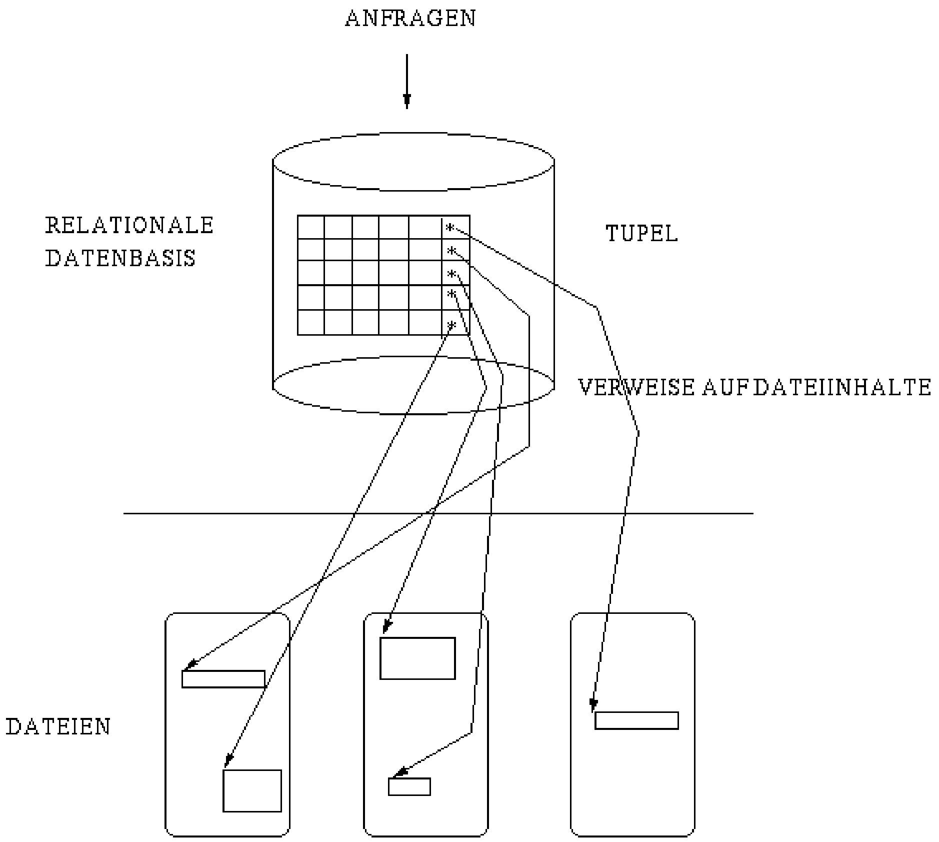


Abbildung 6.1: Strukturierter *Aufsatz* auf semi-strukturierte Daten

Gegenstandes, wobei eine Erweiterung des Data Guides durch Hinzufügen weiterer Datensätze durchaus möglich ist. Die Information des Data Guide dient in erster Linie dem Retrieval, d.h. der Bearbeitung von lesenden Anfragen.

Datenmodelle für semi-strukturierte Daten

Welches sind die Anforderungen, die an ein semantisches Datenmodell zur Darstellung semi-strukturierte Daten gestellt werden? Sollte es eher einfach, d.h. nicht restriktiv sein oder wäre ein komplexes, strukturiertes Datenmodell die bessere Lösung?

Ein einfaches Datenmodell hat den Vorteil, daß neue Daten, trotz ihrer Unstrukturiertheit, in die Datenbank aufgenommen werden können, ohne besondere Software-Tools für eine Vorbearbeitung der Daten zu benötigen.

Oftmals werden aber ganze Relationen mit zugehörigen Indexstrukturen in die Datenbasis importiert. Um Anfragen auf diesen Daten möglichst schnell zu beantworten, muß die Tatsache, daß es sich hier um eine Relation und Indexe handelt, berücksichtigt werden können. Es besteht somit die Notwendigkeit eines komplexen, strukturierten Datenmodells.

Die beiden Forderungen nach einem einfachen und gleichzeitig strukturierten und damit komplexen Datenmodell lassen erkennen, daß ein für semi-strukturierte Daten geeignetes Datenmodell flexibel (anpassungsfähig) sein muß, um beiden Anforderungen zu genügen.

Es existieren zwei Ansätze um semi-strukturierte Daten zu modellieren. Sie unterscheiden sich in der Komplexität des Datenmodells, d.h. in der Anzahl unterschiedlicher Konstrukte die zur Modellierung eingesetzt werden können. Der sogenannte *Maximalansatz* sieht als Modellierungskonstrukte u.a. Objekte, Klassen, Typhierarchien, Mengen, Listen und Arrays vor. Durch die Heterogenität der Datenbestände ist es aber recht schwierig, alle möglichen Datenstrukturen zu berücksichtigen, so daß die Notwendigkeit eines *Minimalansatzes* gegeben ist. Im nächsten Abschnitt wird ein semantisches (und in diesem Fall auch gleichzeitig logisches) Datenmodell vorgestellt, daß sich für semi-strukturierte Daten als besonders geeignet erwiesen hat.[2]

Das OEM (Object Exchange Model)

OEM ist ein Datenmodell, daß dem Minimalansatz zugeordnet wird. Daten werden im *OEM* als Graph dargestellt, wobei die Knoten des Graphen Objekte repräsentieren und die Kanten (labels) des Graphen Attribute (d.h. die Struktur) des Objektes darstellen. Im OEM besitzt jedes Objekt einen eindeutigen Identifikator vom Typ oid und einen Objekttyp der atomar (z.B. integer, real, string, gif, html, audio, java) oder komplex sein kann. Ein komplexer Datentyp ist in diesem Zusammenhang eine Menge von Objektreferenzen der Form (label,oid). Ein Beispiel eines OEM-Graphen ist in Abbildung 6.2 dargestellt.

Mit Hilfe dieses Datenmodells können, neben semi-strukturierten Daten auch Daten aus (einfachen) Relationen oder auch Daten aus NF^2-Relationen modelliert werden. Wie bereits in den vorigen Abschnitten erwähnt wurde, müssen semi-strukturierte Daten keine Struktur ähnlich der in klassischen Datenmodellen aufweisen. Dadurch entsteht ein

[2]Quelle: [AQM96]

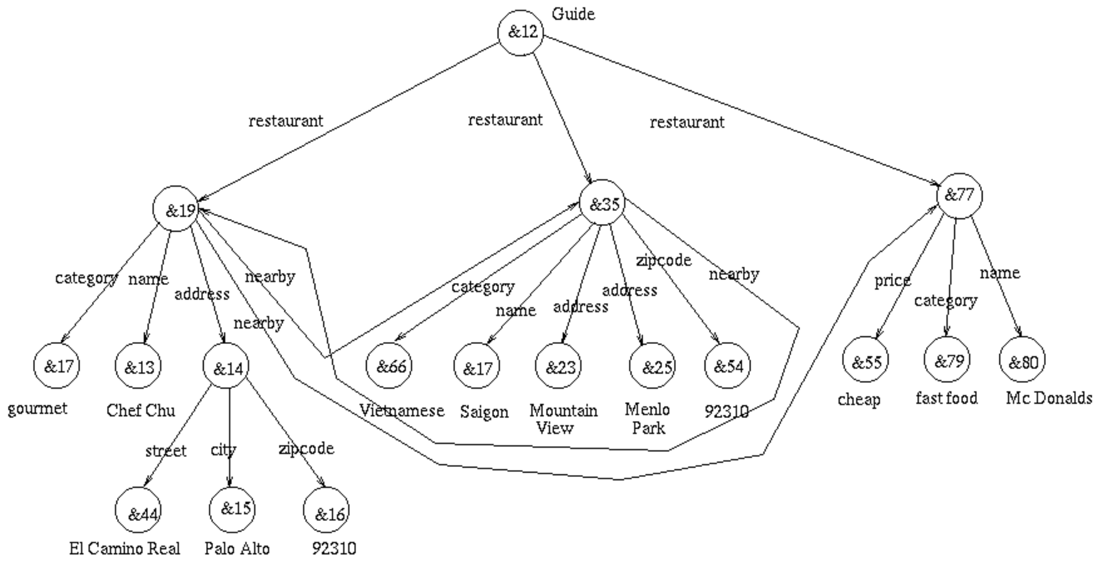


Abbildung 6.2: Beispiel eines OEM-Graphen

gewisser Spielraum, der es ermöglicht, Daten aus unterschiedlichen Quellen (unterschiedliche Systeme und Dateiformate) in einer Datenbank zusammenzufassen. Im Graph aus Abbildung 6.2 sind die erwähnten Spielräume deutlich zu erkennen:

- Restaurants könne keine, eine oder mehrere Adressen besitzen
- Eine Adresse eines Restaurants kann vom Typ *string* oder auch ein komplexer Datentyp sein
- Ein *zipcode* kann vom Typ *string* oder vom Typ *integer* sein
- *Zipcode* kann entweder Attribut eines Restaurants sein oder in der Hierarchie weiter unten liegen (d.h. Attribut eines Unterobjektes)
- Preisangaben können vorhanden sein, sind aber nicht zwingend notwendig

Faßt man die Blattknoten eines OEM-Graphen als Tupel der Form VAL(oid, atomarer Typ) auf und die internen Knoten des Graphen als Tupel der Form MEMBER(oid, label, oid), so können die Daten einer OEM-Datenbasis in einer relationalen Datenbank abgespeichert werden. Das Problem der Suche in semi-strukturierten Daten scheint auf den ersten Blick damit gelöst zu sein. Da aber im relationalen Datenmodell oids nicht vorhanden sind und Anfragesprachen auf relationalen Datenbanken keine Rekursivität anbieten (wäre notwendig, um den gesamten Graphen durchsuchen zu können), sind noch einige Probleme ungelöst.

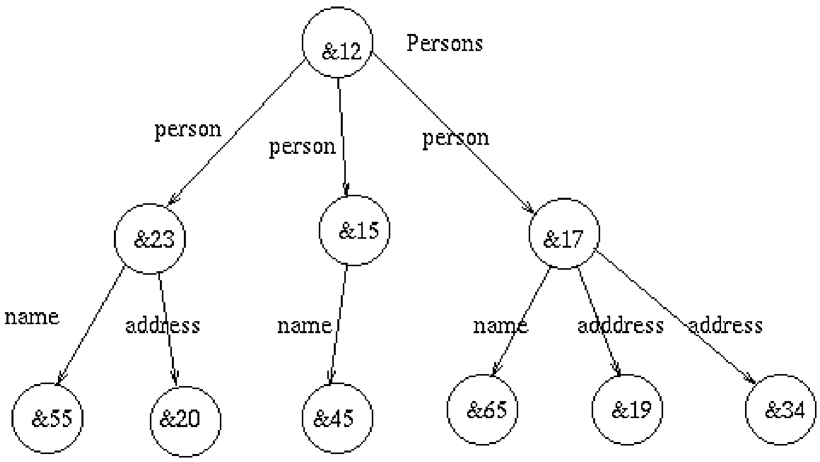


Abbildung 6.3: OEM-Graph

Anfragesprachen für semi-strukturierte Daten

Objektorientierter Ansatz: Der objektorientierte Ansatz geht davon aus, daß die Daten eines OEM-Graphen in der in Abschnitt 6.1.4 vorgestellten relationalen Form vorliegen.

Beispiel: Gegeben sei der OEM-Graph aus Abbildung 6.3. Nach Abschnitt 6.1.4 ergeben sich daraus die folgenden Relationen:

PERSON(oid)
MEMBER(oid_person, name, oid_name)
VALUE(oid_person, address, oid_address)

Trotz fehlender Informationen bezüglich der Struktur der Objekte vom Typ PERSON können einfache Anfragen formuliert werden. Die Anfrage nach der Adresse der Personen (mit Namen) *Smith* sieht wie folgt aus:

SELECT	unique VA.2
FROM	PERSON P, MEMBER N, MEMBER A, VAL VN, VAL VA
WHERE	(P = N.1) AND (N.3 = VN.1) AND (VA.2 = *Smith*) AND
	(A.2 = *address*) AND (A.3 = VA.1)

Die Art der Formulierung solcher Anfragen ist sicherlich gewöhnungsbedürftig. Ein weiterer Nachteil dieses Ansatzes ist der Verlust an Information, der durch die Umformung des OEM-Graphen in Relationen entsteht. Der gesamte Zusammenhang geht dabei verloren. Die Relationen enthalten lediglich Verweisinformationen und dienen ausschließlich dem Navigieren von der Wurzel zu den Blattknoten. Eine wesentlich einfachere und kompaktere Formulierung obiger Anfrage sieht wie folgt aus:

SELECT	A FROM PERSON P, P.address A
WHERE	*Smith* = P.name

Sie setzt aber voraus, daß die Objekte des Typs PERSON alle dieselbe Struktur (d.h. Name und Adresse) aufweisen. Diese Forderung ist aber für semi-strukturierte Daten nicht immer durchsetzbar.

Typwandlung: Die Schwierigkeiten im Umgang mit semi-strukturierten Daten entstehen durch die Tatsache, daß über die Struktur der Objekte nichts genaues bekannt ist. So kann z.B. die Adresse einer Person ein atomarer Typ, eine Menge oder ein Record sein. Jeder der Fälle müßte in Anfragen berücksichtigt werden, was bei der großen Anzahl unterschiedlicher Datentypen unmöglich ist. Um alle Fälle durch ein einziges Konstrukt abzudecken, bietet sich folgende Lösung an: Alle Attribute werden als Mengen aufgefaßt, wobei fehlende Attribute als leere Menge dargestellt werden und Attribute mit atomarem Typ als einelementige Mengen interpretiert werden. Unter dieser Annahme, sieht obige Anfrage wie folgt aus:

SELECT	A FROM PERSON P, P.address A
WHERE	*Smith* IN P.name

Pfadausdrücke: Eine Anfragesprache für semi-strukturierte Daten (die auf einem objektorientierten Ansatz aufbaut) muß das Navigieren zwischen Objekten unterstützen. Dies geschieht mit Hilfe vom Pfadausdrücken, wobei ein Pfad eine Kette von Objekten ist, die über Referenzen miteinander verbunden sind. Anwendungen navigieren durch den OEM-Graphen, indem sie Pfade durchlaufen und dabei auf die entlang des Pfades liegenden Objekte zugreifen.
Beispiel: Guide.restaurant.address.zipcode (siehe OEM-Graph aus Abbildung 6.2)

Erweiterte Pfadausdrücke: Im Umgang mit semi-strukturierten Daten kann es durchaus vorkommen, daß Instanzen desselben Objekttyps unterschiedliche Strukturen aufweisen. Dies erschwert das Formulieren von Anfragen mit Hilfe von Pfadausdrücken, da eine genaue Angabe des Pfades nicht immer möglich ist. Um dieses Problem zu beheben, werden sogenannte *wild cards* verwendet. Sie können nicht nur bei der Formulierung von Pfadausdrücken eingesetzt werden, sondern auch für Attributbezeichnungen (z.B. na% statt name), da diese vom Typ *string* sind.
Beispiel: Gesucht sind die Namen und zipcodes aller billigen (d.h. *cheap*) Restaurants anhand des OEM-Graphen aus Abbildung 6.2:

SELECT	R.name, R(.address)?.zipcode
FROM	Guide.restaurant R
WHERE	R.% grep *cheap*

Das Fragezeichen im Pfadausdruck der SELECT-Klausel bedeutet, daß die Adresse optional ist. Durch die Angabe von % in der WHERE-Klausel werden alle Unterobjekte von *R* nach dem Wort *cheap* durchsucht. Durch die Einführung von *wild cards* ist die Anfragesprache um einiges mächtiger als beispielsweise SQL oder OQL, wo weder reguläre Ausdrücke noch *wild cards* unterstützt werden.
Folgende Anfrage erlaubt das Durchsuchen *aller* Unterobjekte eines Objektes, unabhängig davon, wie tief die Verschachtelung ist:

SELECT t FROM MyReport.#.title t

Ergebnis der Anfrage sind die Titel aller Reports samt den Titeln all' ihrer Kapitel, Unterkapitel usw.[3]

Heterogene Datenbestände: Semi-strukturierte Daten ergeben sich meistens durch Zusammenfassung von Daten aus heterogenen Datenbeständen und weisen somit unterschiedliche Strukturen auf. Möchte man z.B. diejenigen Datensätze selektieren, die sowohl einen Namen als auch eine Adresse (= *Palo Alto*) besitzen, könnte folgendes Konstrukt geeignet sein:

res(name:X, address:Y; REST1) :- r(name:X, address:Y; REST1),
Y = (city: Palo Alto; REST2)

Dabei ist r eine Menge heterogener Tupel, die aber mindestens die Attribute *name* und *address* besitzen. Zusätzliche Attribute sind erlaubt (REST1). Das Attribut *address* muß zumindest das Attribut *city* besitzen. Auf diese Art können Tupel mit einer gewissen Struktur aus der Datenbank selektiert werden.[4]

6.2 Lore: Ein DBMS für semi-strukturierte Daten

6.2.1 Einführung

Lore steht für **L**ightweight **O**bject **Re**pository und ist ein DBMS für semi-strukturierte Daten, das an der Stanford University entwickelt wurde. Bei *Lore* sind die Daten nicht den Beschränkungen eines Schemas unterworfen, sondern können durchaus unterschiedliche Strukturen aufweisen. Existiert Wissen über die Struktur der Daten, kann dieses Zusatzwissen bei der Formulierung von Anfragen eingebracht werden.

6.2.2 Die Systemarchitektur von *Lore*

Lore besitzt zwei Benutzeroberflächen: eine graphische und eine textuelle (Stand 1996). Die textuelle GUI dient in erster Linie dem Debugging (Entwickler), die graphische UI ist die erste Oberfläche, die dem Benutzer zur Verfügung gestellt wurde. Die graphische Benutzeroberfläche enthält eine Vielzahl leistungsfähiger Tools zur Unterstützung der Suche und Darstellung semi-strukturierter Daten:

- Ein *Data Guide* zur Darstellung der Struktur der Daten der Datenbasis (OEM-Graph, siehe Abschnitt 6.2) und gleichzeitig zur Formulierung einfacher Anfragen.

- Ein Tool zur Speicherung häufig vorkommender Anfragen.

[3]Weitere Informationen zu Pfadausdrücken für semi-strukturierte Daten können [KKS92] entnommen werden

[4]Literatur: [PGW95]

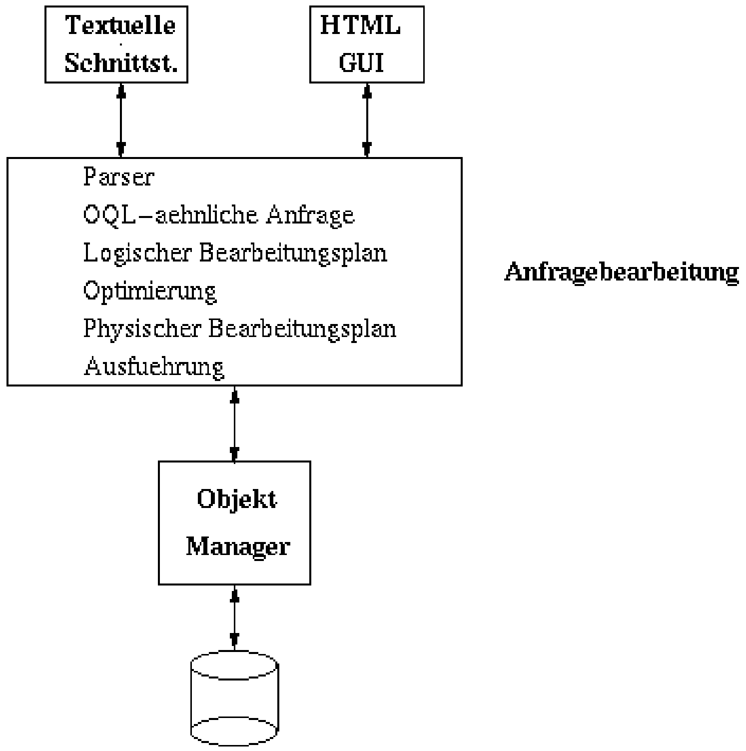


Abbildung 6.4: Die Architektur von *Lore*

- Möglichkeiten zur Darstellung von Daten vom Typ *video*, *audio* und *java* (zum Beispiel als Ergebnis einer Anfrage).

In Abbildung 6.4 ist die (vereinfachte) Architektur von *Lore* dargestellt. Der Objekt-Manager stellt die Schnittstelle zwischen dem persistenten Speichermedium und der Anwendung die sich mit der Anfragebearbeitung befaßt, dar. Seine Aufgaben sind die Bereitstellung von Objekten, der Vergleich von Objekten, Umformungsoperationen unterschiedlicher Art (z.B. Objekt in Menge von Objekten, siehe Abschnitt 6.1.4), das Iterieren über alle Unterobjekte eines (komplexen) Objektes. Zusätzlich sind im Objekt-Manager Cache-Mechanismen zur Speicherung häufig benutzter Objekte implementiert.

Die Bearbeitung von Anfragen übernimmt der sogenannte *query processor*, der die Aufgaben des Anfragetransformators, des Anfrageoptimierers und des Anfragecode-Generators erfüllt. Eine Benutzer-Anfrage durchläuft dabei folgende Stadien:

1. Überprüfung der Korrektheit der Anfrage (parser)

2. Umformung der Anfrage in eine OQL-ähnliche Syntax

3. Erstellung eines logischen Bearbeitungsplans

4. Optimierung

5. Erstellung eines Auswertungsplans auf physischen Datenstrukturen
6. Ausführung des physischen Bearbeitungsplans

In Abschnitt 6.2.4 werden anhand eines Beispiels (und des in Abbildung 6.2 dargestellten OEM-Graphen) die weiter oben vorgestellen Bearbeitungsschritte einer Benutzeranfrage durchlaufen.

6.2.3 Datenmodell und Anfragesprache von *Lore*

Datenmodell

Die Heterogenität der Daten führt dazu, daß klassische Datenmodelle für semi-strukturierte Daten nicht einsetzbar sind. Trotz der unterschiedlichen Struktur muß aber versucht werden, die Objekte als in irgendeiner Weise gleich anzusehen und damit auch gleich zu behandeln. In diesem Sinn' geht *Lore* von der Annahme aus, daß alle Objekte der Datenbasis vom Typ OEM sind. Dabei wird der Typ OEM wie folgt definiert:

type OEM = OEMcomplex + OEMstring +
OEMint + ... + OEMnil
type OEMcomplex =
(a_1:set(OEM), ..., a_n:set(OEM))
type OEMstring = string
...
type OEM = OEMnil()

wobei $a_1, ..., a_n$ die Liste der in der Datenbasis vorkommenden *labels* (Attribute) darstellt.
Beispiel: Das Objekt mit oid &12 hat ein Attribut vom Typ *restaurant*, daß die Menge der oids vom Typ *restaurant* enthält, die von diesem Objekt aus erreichbar sind ({&19,&35,&77}). Alle anderen Attribute sind leer.

Merkmale der Anfragesprache *Lorel*

Anhand von Beispielen werden in den weiteren Unterabschnitten die wichtigsten Merkmale von *Lorel* erläutert.

Umformungen: Ein DBMS für semi-strukturierte Daten muß Mechanismen zur Typumwandlung zur Verfügung stellen, um möglichen Typisierungsfehlern entgegenzuwirken.
Beispiel: Gesucht sind die Adressen aller Restaurants, deren Zipcode gleich 92310 ist.

```
SELECT   X.address
FROM     Guide.restaurant X, X.zipcode Y
WHERE    Y = 92310
```

Obige Anfrage würde in OQL zu einem *type error* führen, da in der WHERE-Klausel ein Objekt mit einem *Integer* verglichen wird. Nicht so in Lorel, in diesem Fall wird das Objekt Y derefenziert und es erfolgt der Vergleich mit dem so erhaltenen Wert. Ein ähnlicher Fall tritt auf, wenn ein Objekt (lt. Anfrage) mit einer Menge von Objekten verglichen werden soll. Im Normalfall würde das auch zu einem Fehler führen, Lorel formt diese Gleichheit aber in eine Existenzbedingung um und umgeht dadurch das Problem.

Pfadausdrücke: Pfadausdrücke werden wie bereits in Abschnitt 6.1.4 vorgestellt verwendet.
Beispiel: Gesucht sind die Zipcodes aller Restaurants der Datenbasis, die Unterobjekt von *restaurant* und diese wiederum Unterobjekte von *Guide* sind.

```
SELECT   Z
WHERE    Guide.restaurant.zipcode Z
```

Erweiterte Pfadausdrücke: Erweiterte Pfadausdrücke werden wie in Abschnitt 6.1.4 eingesetzt.
Beispiel: Guide.restaurant(.address)?.zipcode
Beispiel: Guide.restaurant(.nearby)*R.name

Im ersten Beispiel werden, ausgehend vom Objekt *Guide*, alle Objekte mit label *restaurant* durchlaufen, von diesen Objekten werden diejenigen selektiert, die ein label *zipcode* besitzen, oder diejenigen die ein *address* label aufweisen, das ein label *zipcode* besitzt. Im zweiten Beispiel werden die Namen aller Objekte ausgegeben, die sich auf einem Pfad der Form: Guide.restaurant.nearby befinden.

Anfrageergebnisse : Anfrageergebnisse werden in *Lore*, anders als in klassischen DBMS, abgespeichert, und können somit wiederverwendet werden. In SQL beispielsweise werden Ergebnisse von Anfragen in der Regel nur temporär gespeichert. Bei *Lore* hingegen werden die Ergebnisse in einem neuen Objekt vom Typ OEM abgelegt und dem OEM-Graphen hinzugefügt.
Beispiel: Gesucht sind die Unterobjekte von Guide vom Typ (label) r*restaurant*.

Anfrage		Ergebnis
SELECT	X	answer &155
WHERE	Guide.restaurant X	restaurant &19
		restaurant &35
		restaurant &77

Ergebnis dieser Anfrage ist ein neu erzeugtes Objekt (*answer*) vom Typ OEM, dessen Unterobjekte (label) jeweils Objekte vom Typ *restaurant* sind. Das neue Objekt wird dem OEM-Graphen hinzugefügt.

6.2.4 Anfragebearbeitung bei *Lore*

Beispiel: Es sollen die Adressen derjenigen Restaurants ausgegeben werden, für die category = *gourmet* gilt. Die Anfrage in *Lorel* (**Lore l**anguage) lautet:

SELECT Guide.restaurant.address
WHERE Guide.restaurant.category = *gourmet*

Im ersten Schritt überprüft der Parser die Korrektheit (Syntax) der Anfrage. Anschließend wird die Anfrage in eine OQL-ähnliche Syntax umgeformt:

SELECT Y
FROM Guide.restaurant X, X.address Y
WHERE exists Z in category : Z = *gourmet*

Um die Anfrage bearbeiten zu können, ist es erforderlich die Menge von Objekte des OEM-Graphen soweit zu *ordnen*, daß relationale Operatoren darauf angewandt werden können. Dafür wird eine einfache Datenstruktur eingeführt, die sich an der zu bearbeitenden Anfrage orientiert. Die Datenstruktur enthält eine endliche Anzahl sogenannter *slots*, die je einem Pfadausdruck zugeordnet sind und den oid eines über den Pfad erreichbaren Objektes enthalten. Zur Speicherung von oids die zur Bearbeitung der Anfrage notwendig sind (aber keinem Pfadausdruck entsprechen), können weitere *slots* eingeführt werden. Für unsere Anfrage ist folgende Datenstruktur notwendig:

OA_0 Guide	OA_1 OA_0.restaurant	OA_2 OA_1.address
OA_3 OA_1.category	OA_4 Aggr	

Der logische Bearbeitungsplan für obige Anfrage ist in Abbildung 6.5 dargestellt. Der Bearbeitungsplan wird von den Blättern in Richtung Wurzel abgearbeitet. Als erstes werden ausgehend von der Wurzel des OEM-Graphen alle Unterobjekte vom Typ *restaurant* durchlaufen und mit dem einattributigen Tupel gejoint. Dadurch entstehen eine Menge von Tupeln die jeweils den gleichen Eintrag unter (OA_0 und unterschiedliche Einträge unter (OA_1 aufweisen. Anschließend erfolgt ein Join dieser Tupel mit allen Unterobjekten vom Typ *address*. Die erhaltenen Tupel enthalten für jedes Restaurant die gesamte benötigte Information.
Im rechten Teilbaum wird in einer Datenstruktur nur die Information zu AO_1 (restaurant) und AO_3 (category) festgehalten. Im ersten Schritt werden diejenigen Tupel selektiert, für die die Bedingung: category = *gourmet* erfüllt ist. Anschließend werden die erhaltenen Tupel nach ihren restaurant.oids (OA_1) gruppiert. Im Slot AO_4 wird vermerkt, ob sich für ein bestimmtes Restaurant mindestens ein Tupel qualifiziert hat (OA_4 = true oder false, d.h. Restaurant besitzt die Kategorie *gourmet* oder nicht).
Im nächsten Schritt erfolgt der Join mit den Tupeln des linken Teilbaums. Anschließend werden diejenigen Tupel selektiert für die OA_4 = true (d.h. category = *gourmet*). In einem letzten Schritt erfolgt die Projektion auf OA_2 um die Adresse der gefundenen Restaurants auszugeben.

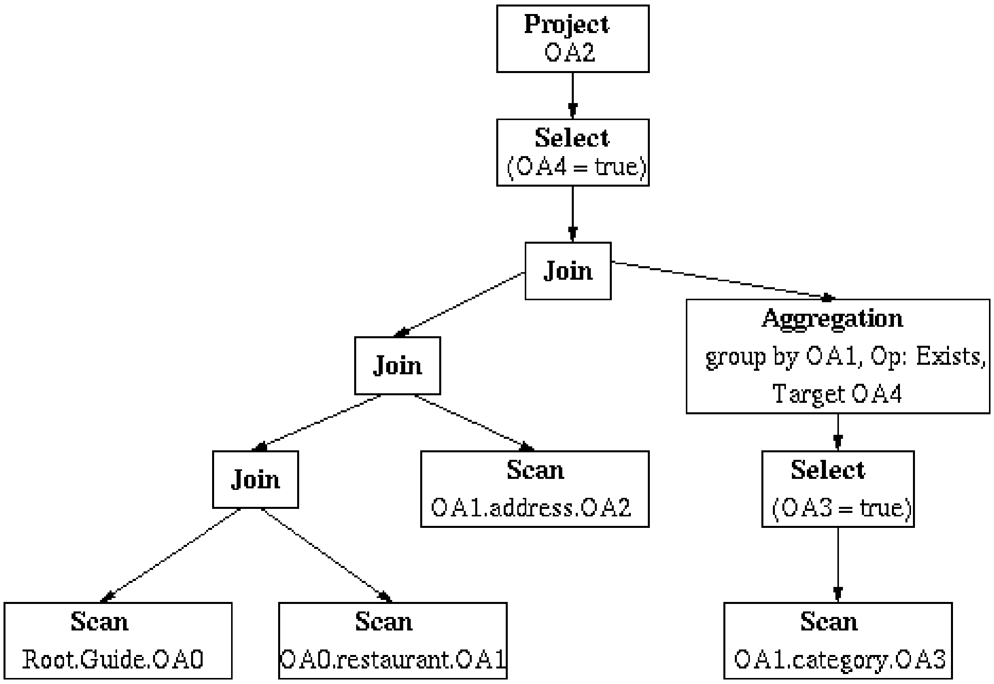


Abbildung 6.5: Logischer Bearbeitungsplan einer Anfrage in *Lorel*

Auf den ersten Blick fällt auf, daß dieser logische Bearbeitungsplan keineswegs optimiert ist. Um eine Anfragebearbeitung zu beschleunigen, muß die Anzahl der Hintergrundspeicher-Zugriffe minimiert werden. Dies erreicht man, indem man darauf achtet selektive Prädikate möglichst *früh* auszuführen, um damit die Kardinalität von Zwischenergebnissen zu reduzieren. In unserem Fall wäre es möglich die Selektion $OA_4 = \text{true}$ schon nach dem Gruppierungsknoten durchzuführen, um den Aufwand des folgenden Joins zu verringern. Auch die Projektion kann weiter nach unten verschoben werden.

6.2.5 Anfrageoptimierung bei *Lore*

In klassischen DBMS existieren Mechanismen zur Anfrageoptimierung, in *Lore* hingegen sind lediglich einfache Techniken der Optimierung vorhanden (Stand Februar 1997). Dazu zählt in erster Reihe die heuristische Optimierung, deren Hauptregel besagt, daß versucht werden muß Zwischenergebnisse möglichst klein zu halten. In *Lore* werden Selektionen so früh wie nur möglich ausgeführt um das Datenvolumen (für eventuelle Joins oder sonstige aufwendige Operationen) möglichst klein zu halten.
Zusätzlich werden Indexe eingesetzt, ähnlich denen in klassischen DBMS, um den direkten Zugriff auf Datensätze zu beschleunigen. Die Indexe werden als B*-Bäume (physisch) abgelegt, um sowohl EMQ als auch RQ gut zu unterstützen. Geplant sind u.a.:

- Weitere Mechanismen zur Anfrageoptimierung,

- Recoverymechanismen,
- Sperrmechanismen,
- Implementierung von Funktionen zur Definition von Sichten.

Kapitel 7

Datenbankanfragen ans WWW: WebSQL

Matthias Wagenmann

7.1 Kurzfassung

WebSQL ist eine Sprache, die es ermöglicht, Informationen aus dem World-Wide-We b zu extrahieren. Dabei wird eine ähnliche Syntax verwendet, wie sie bei SQL verwend et wird. Der Funktionsumfang ist jedoch im Vergleich zu SQL stark reduziert. Entwickelt wird die Sprache im Rahmen des **SQL-Projekt**s an der **University of Toronto** vom **Depart ment of Computer-Science**.
Implementiert wurde WebSQL als eine Library von Java-Klassen, wodurch Anfragen i n WebSQL auf ähnliche Weise in Java eingebettet werden können, wie SQL-Anfragen in C-Programme. Dies erlaubt es, auf einem viel höheren Abstraktionsniveau auf das Web zuzugrei fen, als dies durch reine HTTP-Anfragen möglich wäre. Außerdem bietet WebSQL einen transparenten Zugriff auf Suchmaschinen, die mitt els CGI [1] abgefragt werden können.

Die fertig übersetzten Java-Klassen sowie eine Online-Dokumentation in Englisch stehen zum Download auf der Seite [WH97b] bereit.

7.2 Einführung

Die bisherige Praxis, interessante Dokumente im Web zu finden, ist es, sich entw eder durch endlos lange Hyperlink-Ketten zu klicken, oder eine Suchmaschine zu verwenden. Beide Vorgehensweisen erschlagen meist den Suchenden mit irrelevanten Informationen un d dieser findet vor lauter Bäumen den Wald nicht.

[1] Common Gateway Interface

Dieses Problem hat seine Ursache im Aufbau von HTML: Einzelne Dokumente sind üb er Links beliebig untereinander vernetzt - eine hierarchische Struktur existiert nicht. D adurch ist es nicht möglich, von einem allgemeineren Themengebiet ausgehend die Suche immer weiter einzuengen. Die hierarchische Struktur des Internets wird bei den herkömmlichen

Suchmaschinen nicht berücksichtigt [Hal88]. Bild 7.2 und Bild 7.2 veranschaulichen diese beiden Sichtweisen auf das Internet.

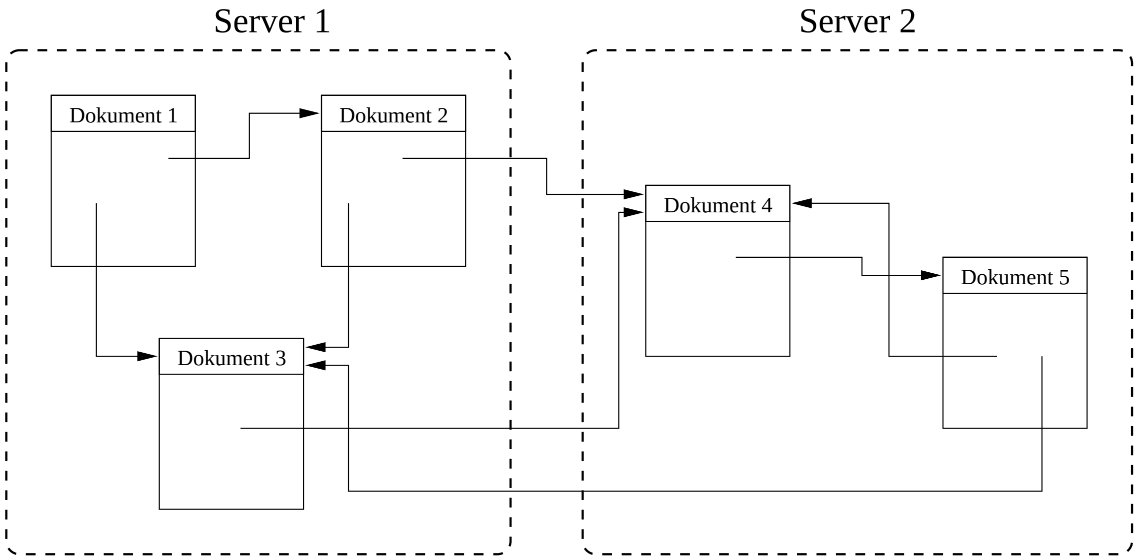


Abbildung 7.1: Das Internet als HTML-Struktur

WebSQL umgeht dieses Problem, indem des die Suche über Stichwörtern mit einer automatisierten Navigation durchs Netz kombiniert. Erste Spezifikationen dieser Sprache und deren Implementierung sind in [Mih96] zu finden.
Trotz der Vorteile, die WebSQL gegenüber herkömmlichen Suchmaschinen hat, wird der Endbenutzer des WWW sich in der Regel nicht die Mühe machen, eine WebSQL-Anfrage zu program mieren, um eine bestimmte Seite zu finden. Doch genauso, wie SQL dazu benutzt wird, Date nbanken zu programmieren, läßt sich WebSQL dazu benutzen, anwenderfreundliche Programm e fürs Web zu programmieren. So könnte z.B. mit WebSQL ein Programm geschrieben werden , das in regelmäßigen Abständen Homepages durchforstet, tote Links und Seiten, auf die kein Verweis existiert, registriert, und diese per E-Mail dem Betreiber der Homepage mitteilt. Provider könnten auf diese Art einiges an Festplattenspeicher einsparen.

7.3 Das Prinzip von WebSQL

WebSQL modelliert das Web als relationale Datenbank mit zwei virtuellen Relation en: **Document** und **Anchor**. Die Relation **Document** besitzt ein Tupel p ro Dokument im

Web, die Relation **Anchor** ein Tupel pro Link in jedem Dokument im Web. Die einzelnen Felder der beiden Relationen sind in den Abbildungungen 7.3 und 7.3 dargestellt.

Wären diese beiden Relationen komplett verfügbar, könnte man SQL benutzen, um Anfragen zu schreiben. Die folgende Anfrage würde die URLs aller Paare von Dokumenten mi t gleichem Titel zurückgeben:

```
SELECT d1.url, d2.url
FROM Document d1, Document d2
WHERE d1.title = d2.title
AND NOT (d1.url = d2.url);
```

Diese Anfrage kann jedoch nicht effektiv behandelt werden, da die Relation **Dokument** virtuell ist, und es keine Möglichkeit gibt, alle Dokumente im Web zu materiali sieren. Die obige Anfrage müßte z.B. jedes Dokument im Web mit jedem anderen Dokument verg leichen, was wohl ein hoffnungsloses Unterfangen wäre.
Deshalb operiert WebSQL nicht direkt mit der virtuellen Relation, sondern materi alisiert Teile davon. Damit dies gelingt, muß der Kreis der in Frage kommenden Dokumente reduz iert werden. Um dies zu erreichen, müsste die obige Anfrage in WebSQL ungefähr so a ussehen:

```
SELECT d1.url, d2.url
FROM Document d1 SUCH THAT d1 MENTIONS "wasauchimmer",
     Document d2 SUCH THAT d2 MENTIONS "wasauchimmer",
WHERE d1.title = d2.title
AND NOT (d1.url = d2.url);
```

Der einzige Unterschied von dieser Anfrage zur vorhergehenden ist der Zusatz f SUCH THAT ... MENTIONS ... bei jeder Variablen. Dies bedeutet, daß nur Dokumente betrach tet werden, in denen das angegebene Schlüsselwort gefunden wird. Dadurch sinkt die Anzahl d er Dokumente, und sie können materialisiert werden. Um die Seiten, die das Schlüs selwort enthalten, zu extrahieren, werden die bekannten Suchmaschinen kontaktiert ([AS97], [ES97],[HC97], etc.).

7.4 Einige Beispiele

7.4.1 Intelligente Suche

Wie bereits erwähnt, sind Anfragen an Suchmaschinen eine der zwei Wege, wie mom entan Informationen im Web gesucht werden. Die andere ist die manuelle Navigation durc hs Netz. WebSQL erlaubt die Automation dieses Prozesses. Die Anfrage

```
SELECT d.url, d.title
FROM   Document d SUCH THAT "http://www.somewhere.com" -> d;
```

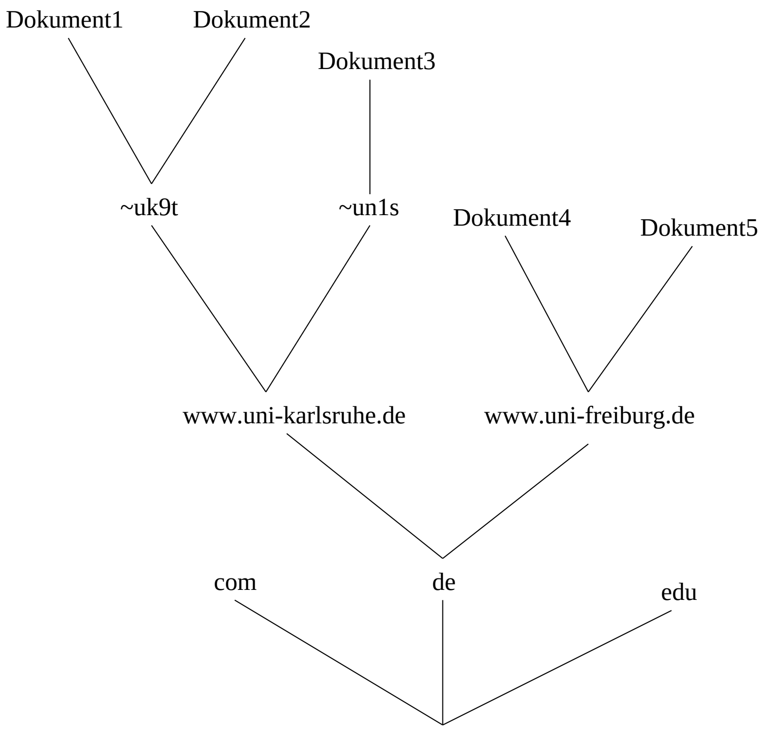


Abbildung 7.2: Ein Ausschnitt aus der Internet-Hierachie

url	**title**	**text**	**length**	**type**	**modif**
http://www...	title 1	text 1	1234	text	1-1-97
http://www...	title 2	text 2	5678	text	1-2-97
http://www...	title 3	text 3	9999	text	1-3-97
...	...	...	...	...	...

Abbildung 7.3: Dokumente des WWW als Relation **Document** von WebSQL

base	**label**	**href**
http://www...	label 1	http://www...
http://www...	label 2	http://www...
http://www...	label 3	http://www...
...	...	...

Abbildung 7.4: Links des WWW als Relation **Anchor** von WebSQL

ergibt den Titel und die URL von jedem Dokument, auf das von der angegebenen URL verwiesen wird, und das sich auf dem selben Server befindet.

Der Pfeil zwischen der URL und der Variablen spezifiziert den Pfad, nachdem das Dokument zu finden sein muß. Der in diesem Beispiel verwendete Pfeil besagt, daß das Do kument über einen lokalen Link zu erreichen sein muß, das heißt, es muß auf demselb en Server liegen. Im Gegensatz dazu gibt es globale Links. Diese verweisen von einem Serve r auf einen anderen. Die Tabelle 7.4.1 zeigt einige mögliche Pfadangaben.

$->->=>$	Ein Pfad der Länge 3, der sich aus zwei lokalen und einem globalen Link zusammensetzt. Dabei ist die Reihenfolge von Bedeutung, das heißt , es werden genau die Seiten angesprochen, die vom Startdokument aus über einen lokalen Lin k, noch einem lokalen Link und daraufhin folgend mit einem globalen Link zu erreichen sind.
$->\|=>$	Ein Pfad der Länge eins, egal ob lokal oder global
$->*$	Ein lokaler Pfad beliebiger Länge
$=>*$	Ein globaler Pfad beliebiger Länge. Diese Angabe würde jedoch das System höchstwahrscheinlich zum Überlaufen bringen
$=>->*$	Ein globaler Link gefolgt von beliebig vielen lokalen Links.

Abbildung 7.5: einige mögliche Pfadangaben unter WebSQL

Mit Hilfe dieser Pfadangaben können interessante Suchmuster spezifiziert werden . So sucht z.B. die nächste Anfrage nach Seiten der Universität Karlsruhe, die das Stichw ort **Mensa** im Titel oder im Text selbst enthalten:

```
SELECT d.url
FROM   Document d SUCH THAT "http://www.uni-karlsruhe.de" ->* d
WHERE  d.text CONTAINS "Mensa" OR d.title CONTAINS "Mensa";
```

Eine weitere interessante Variation einer Suchanfrage ist folgende, die alle Sei ten liefert, auf denen Stellenangebote für Informatiker enthalten sind:

```
SELECT d2.url, d2.title
FROM   Document d1 SUCH THAT d1 MENTIONS "Stellenangebot",
       Document d2 SUCH THAT d1 =|->|->-> d2
WHERE  d2.text CONTAINS "Informatiker";
```

Ausgehend von allen Seiten, die das Schlüsselwort **Stellenangebot** enthalte n, werden alle Seiten, die **Informatiker** enthalten und über einen Pfad der Länge null, e ins oder zwei zu erreichen sind, gesucht. Eine Anfrage an eine herkömmliche Suchmaschine mit den Sichworten **Stellenangebot** und **Informatiker** hätten nur Seiten geliefert, auf de nen beide Suchbegriffe gleichzeitig auftreten. Dadurch wären wohl die meisten Angebote verloren gegang en. Bisher wurde die Relation **Document** verwandt, die Relation **Anchors** ka nn jedoch auch sehr nützlich beim Suchen im Netz sein. Bei einer Anfrage können so zum Beispi el nur Links betrachtet werden, die sich auf einem bestimmten Dokument befinden (mit dem **base**-Attribut) oder die auf bestimmte Dokumente verweisen (mit de m **href**-Attribut. Folgende Anfrage liefert alle Papers, die von der Fraunhofer-Gesellschaft IITB online angeboten werden:

```
SELECT d1.url, d1.title
FROM   Document d1 SUCH THAT "http://www.fhg.ittb.de" ->* d1,
       Anchor a1 SUCH THAT base = d1,
WHERE  filename(a1.href) CONTAINS "ps.gz"
   OR  filename(a1.href) CONTAINS "ps.Z";
```

7.4.2 Homepage-Verwaltung

Eine weitere interessante Anfrage liefert alle Links einer Homepage, die nicht mehr aktuell sind, d.h. die auf Dokumente verweisen, die gelöscht wurden:

```
SELECT d.url, a.href
FROM   Document d SUCH "http://server/index.html" ->* d,
       Anchor a SUCH THAT base = d
WHERE  protocol(a.href) = "http" AND doc(a.href) = null;
```

Durch diese Anfrage werden ausgehend von einer Startseite alle Seiten, auf die lokal verwiesen wird, gesucht. In der Regel sind dies alle Seiten einer kompletten Homepage. Auf jeder dieser Seiten werden nun alle Links daraufhin untersucht, ob sie auf eine HTTP-Seite verweisen,

die nicht vorhanden ist.

Will ein Homepage-Besitzer, der Links auf die Homepages seiner Freunde hat, überprüfen, ob diese auch ein Link auf seine Homepage haben, kann er dies mit folgender Anfrage feststellen:

```
SELECT d.url
FROM   Document d  SUCH THAT "Link-Page" => d,
       Document d1 SUCH THAT d=>|->=> d1,
       Anchor   a  SUCH THAT base = d1
WHERE  server(a.href) = "mein.server";
```

Hier werden ausgehend von einer Linkpage, also einer Seite die lauter globale Links enthält, alle anderen Server besucht, und diese ausgegeben, die entweder direkt, oder über einen lokalen Link auf den gewünschten Server zeigen.

7.5 Die DEFINE-Anweisung

7.5.1 Benutzerdefinierte Links

Oft werden Dokumente als verkettete Listen organisiert, die beispielsweise mit einem **Weiter**-Button untereinander verknüpft sind. Folgende Anfrage sucht eine derartige Liste nach einem Begriff durch:

```
DEFINE LINK [next] AS label CONTAINS "Weiter";
FROM   Document d SUCH THAT "Startseite" [next]* d,
WHERE  d.text CONTAINS "Begriff";
```

Sollen bei der Suche nur Links von einem Dokument berücksichtigt werden, die sich in einem Unterverzeichnis des Dokumentenverzeichnis befinden, hilft folgende Definition:

```
DEFINE LINK [Deeper] AS server(hrref) = server(base)
                     AND path(href) CONTAINS path(base);
```

7.5.2 Server-Definition

Soll eine andere Suchmaschine als AltaVista verwandt werden, so kann diese mit der Definition

```
DEFINE INDEX = "Suchmaschine";
```

als Lieferant für die Dokumente, die mittels der MENTION-Klausel, verifiziert werden, eingestellt werden.

7.5.3 Kontext-Definition

Hier kann angegeben werden, auf welche Weise das Attribut **context** für die Relation **Anchors** definiert wird. Dabei kann angegeben werden, wieviel Text vor dem Tag $< A\ HREF = ... >$ und wieviel Text nach dem $< /A >$-Tag im Kontext zu dem Link betrachtet werden soll.
Beim folgenden Beispiel wird als Kontext zu jedem Link alles interpretiert, was zwischen dem vorhergehenden HR-Tag und dem ersten ihm nachfolgenden P-Tag steht:

```
DEFINE context begin=HR,end=P;
```

Eine kleine Variante stellt das folgende Beispiel dar. Der einzige Unterschied zum vorhergehenden Beispiel ist der Punkt vor dem P-Tag. Dies bewirkt, daß der erste P-Tag, der dem HR-Tag folgt, als Endmarkierung für den Kontext des Links diehnt, und nicht das erste P-Tag nach dem $< /A >$-Tag:

```
DEFINE context begin=HR,end=.P;
```

Eine andere Möglichkeit besteht darin, die Anzahl der Tags anzugeben, die zu berachten sind. Hier werden alle Zeichen des HTML-Scripts betrachtet, die zwischen dem vorhergeh enden HR-Tag und dem vierten Tag nach dem $< /A >$-Tag liegen:

```
DEFINE context begin=HR,end=4;
```

7.6 Realisierung

In diesem Kapitel wird die aktuellen Implementierung der WebSQL-Sprache vorgeste llt. Alle Teile des Systems wurden als eine Library von Java-Klassen konzipiert und lassen sich somit von jedem Java-Programm problemlos benutzen.

Der allgemeine Aufbau dieser Library ist in der Abbildung 7.6 zu sehen.

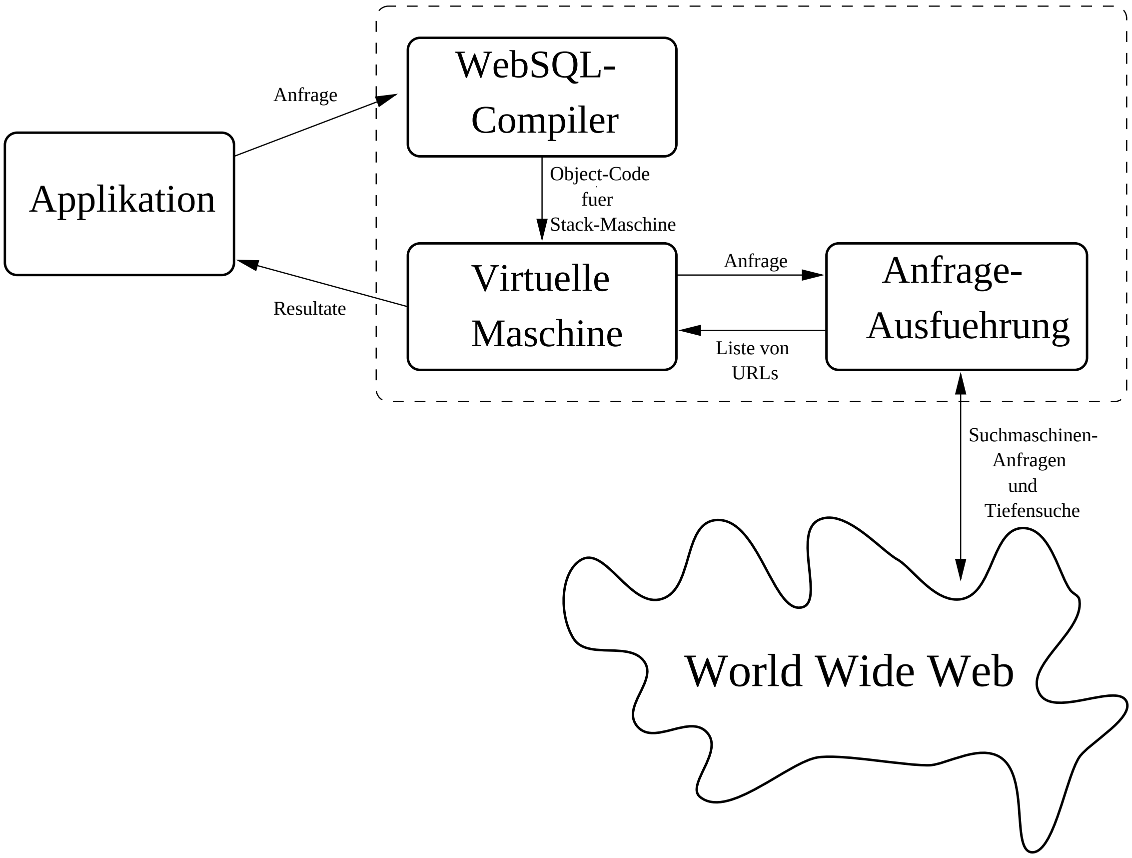


Abbildung 7.6: Die Architektur der WebSQL-Systems

7.6.1 Der Compiler und die virtuelle Maschine

Der WebSQL-Compiler übersetzt den Anfrage-String in ein Programm für eine Stac k-Maschine. Diese wird als virtuelle Maschine bereitgestellt. Die Sprache WebSQL wird also m ittels des Compilers auf eine niedrigere Ebene, eben die einer Stack-Maschine, transferiert . Auf dieser Ebene wird dann der erzeugte Programmcode interpretiert. Das Konzept, wie WebSQL Anfragen behandelt, ist also mit dem Compilierungs- und Ausführungsprozess eines Java-Pr ogramms vergleichbar.

7.6.2 Die Anfrage-Ausführung

Findet der Interpreter bei der Abarbeitung der Anfrage eine **FROM**-Anweisung , wird die Anfrage-Ausführung kontaktiert. Abhängig vom Typ der Anweisung wird entweder e ine Anfrage an eine Suchmaschine getätigt oder eine Tiefensuche im Dokumenten-Netzwerk gest artet. Als Ergebnis liefert sie eine Liste von URLs zurück.

7.7 Andere WWW-Anfragesprachen: W3QL

Der grundlegende Aufbau von W3QL ist, wie auch der von WebSQL, an SQL angelehnt. Allerdings haben die beiden Sprachen eine etwas unterschiedliche Syntax. An einem Beispiel sollen diese Unterschiede erläutert werden. Die folgende Anfrage soll alle Bilder, die über lokale Links von der Startseite der Uni-Karlsruhe aus erreichbar sind, ausgeben:

```
Select n3:
From n1,l1,(n2,l2),l3,n3
Where
n1 in {http://www.uni-karlsruhe.de};
l1 in {/.uni-karlsruhe\.de/};
l2 in {/.uni-karlsruhe\.de/};
n3: PERLCOND 'n3.format=~/image/';
n3 in {/.uni-karlsruhe\.de/}
Using iSEARCHd -d 5 -l 1000
```

Anders als bei WebSQL wird hier der Pfad schon bei der FROM-Anweisung spezifizie rt. Die Angabe hier besagt, daß von dem Dokument **n1** ausgehend über den Link **l1** und beliebig vielen Dokumenten **(n2,l2)** das Dokument **n3** erreichbar sein muß. Zwischen lokalen und globalen Links wird nicht unterschied en. Als Startdokument wird **http://www.uni-karlsruhe.de** angegeben. Die nächste n Zeilen besagen, daß die weiterführenden Links **l1** und **l2** nur besuch t werden, wenn sie lokale Links sind. Bei den Angaben in den geschweiften Klammern handelt es sich um reguläre PERL-Ausdrücke.Die letzte Zeile wird das Programm spezifiziert, welches als Suchmaschine verwendet werden soll. Die Übergabeparameter besagen, daß maximal 1000 HTTP-Anfragen gestartet werden sollen, und Pfade nur bis zu einer maximalen Länge von 5 verfolgt werden .

Eine Stärke von W3QL ist es, automatisch Formulare, die in HTML-Seiten eingebet tet sind, ausfüllen zu können. Eine derartige Anfrage in einer HTML-Struktur stell t für WebSQL eine unüberwindbare Barriere dar. Folgendes Beispiel überwindet di e User-Abrage der Uni-Bibliothek Karlsruhe:

```
Select n2:

From n1,l1,n2
Where
n1 in {http://www.ubka.uni-karlsruhe.de/hylib-bin/ausleihe.cgi};
```

```
Fill n1 as in Report;
Run learnform n1 colorado:0 if n1 unknown in Report;
Using iSEARCHd
```

Wird hier das Formular erreicht, versucht W3QL, es so auszufüllen, wie es in de r Datei **Report** festgehalten ist. Wurde das Formular bisher noch nicht aufger ufen, tritt die nächste Zeile (**Run learnform...**) in Aktion: W3QL öffnet ein Fenster auf dem Computer **colorado**, fragt die Eingaben ab, und speichert di ese in der Datei **Report**. Wird die Seite das nächste mal abgefragt, füllt W3Q L das Formular automatisch aus.

W3QL verwendet nicht wie WebSQL automatisch eine Suchmaschine, um Dokumente im W WW zu finden. Um dennoch nicht bei jeder Anfrage das Web durchstöbern zu müssen, besitzt W3QL einen lokalen Speicher. Als Besonderheit kann bei W3QL angegeben we rden, wie oft eine Anfrage neu, das heißt nicht aus dem lokalen Speicher, ausgewertet wer den soll. Die folgende Anfrage liefert beispielsweise jeden Tag die neue Version der New-Y ork Times:

```
Select n1:
From n1
Where
N1 in {http://nytimesfax.com/time.pdf};
Using ISEARCHd
Evaluated every day
```

Auf der Seite [WH97a] ist ein CGI-Script eingerichtet, durch das Anfragen an W3QL gestartet werden können.

7.8 Bewertung

Sprachen wie WebSQL sind eine echte Alternative zu den bestehenden Arten, das WW W nach Informationen zu durchsuchen. Dabei ist der Funktionsumfang von WebSQL im Vergle ich zu W3QL kleiner, jedoch ist WebSQL intuitiver zu bedienen. Durch den Umstand, da ß WebSQL bei vielen Anfragen Suchmaschinen kontaktieren, und nicht wie W3QL die Ad ressen wirklich besucht, ist WebSQL beim ersten Aufruf im Vergleich um einiges schnelle r, jedoch auch unflexibler. WebSQL unterstützt leider den Service vieler Suchmasch inen

noch nicht, nach Links in einem Dokument zu suchen. So ist es weder WebSQL noch W3QL möglich, sämtliche Seiten, die eine bestimmte Seite adressieren, in einer Anfr age zu berücksichtigen.

Es bleibt abzuwarten, ob sich derartige Abfragesprachen im Benutzerbereich durch setzen werden. Für die meisten wird wahrscheinlich eine Anfrage an die traditionellen Suchmaschinen zu schnelleren Ergebnissen führen, insbesondere da diese immer me hr Optionen bieten, um die Suche im Netz zu spezialisieren. Für den Netzwerkmanage r oder den Webmaster bieten die Sprachen jedoch die Möglichkeit, sich oft wiederholend e Tätigkeiten zu automatisieren.

Wahrscheinlich werden in Zukunft die bestehenden Suchmaschinen um eine derartige Sprachen erweitert werden, und dem Benutzer neben der normalen Stichwortsuche einen Exper tenmodus anbieten zu können.

7.9 Sprachreferenz

Im Folgenden wird der aktuelle Befehlssatz von WebSQL aufgeführt. Eckige Klamme rn bedeuten, daß der darin enthaltene Ausdruck vorkommen kann, aber nicht muß. Geschweifte Klammern besagen, das das Innere beliebig oft (auch garnicht) auftre ten kann. Ausdrücke, die in Anführungszeichen stehen, stellen Klartextangaben dar. Sie bilden also kein Verweis auf eine andere Definition, sondern sind genau so einzugeben.

7.9.1 Definitionen

```
Program          := [Definitions] Query
Definitions      := {LinkDefinition} [IndexDefinition] [Contextdefinition]
LinkDefinition   := ‘define link‘ LinkSymbol ‘as‘ Condition ‘;‘
LinkSymbol       := [STRING]
IndexDefinition  := ‘define index ="STRING";‘
ContextDefinition:= ‘define context begin =‘ Endpoint
                    ‘, end =‘ Endpoint ‘;‘
Endpoint         := [.] [TAGNAME | INT]‘;‘
```

7.9.2 Anfrage

```
Query    := ‘select‘ AttrList ‘from‘ RangeExpressionList
            [‘where‘ Condition]‘;‘
Attribute:= TableVar.Field|Field
Field    := ‘url‘ | ‘title‘ | ‘length‘ | ‘modif‘ | ‘type‘ |
            ‘base‘ | ‘label‘ | ‘href‘ | ‘context‘
TableVar := IDENTIFIER

AttrList := Attribute {,Attribute}
Node     := "URL" | TableVar
```

7.9.3 Bereichsangabe

```
RangeExpression    := Table TableVar ‘such that‘ RangeCondition
Table              := ‘Document‘|‘Anchor‘
RangeCondition     := Node PathRegExp TableVar |
                      Node ‘mentions‘ "PATTERN" |
                      ‘base=‘ Node
```

```
RangeExpressionList:= RangleExpression {,RangeExpression}
```

7.9.4 Pfadangabe

```
PathRegExp:= ‘=‘ | Link | PathRegExp‘*‘ | PathRegExp PathRegExp |
             PathRegExp | ‘(‘PathRegExp‘)‘
Link       := ‘=‘ | ‘->‘ | ‘=>‘ | LinkSymbol
```

7.9.5 Bedingungen

```
Condition:= Condition {‘or‘ Condition} | Condition { ‘and‘ Condition} |
            ‘not‘ Condition | ‘(‘ Condition ‘)‘ |
            Expression RelOp Expression | Expression ‘contains‘ "STRING"
RelOp    := ‘=‘ | ‘<‘ | ‘>‘ |‘ <=‘ | ‘>=‘
```

7.9.6 Ausdrücke

```
Expression:= Attribute | Function‘(‘Attribute‘)‘[‘.‘Field] |
             "STRING" | <null>
Function  := ‘protocol‘ | ‘server‘ | ‘pathname‘ | ‘path‘ |
             ‘file‘ | ‘extension‘ | ‘doc‘
```

Kapitel 8

Datenbankanbindungen an das WWW

Christian Sy

Kurzfassung *Dieser Artikel behandelt verschiedene existierende Techniken der Datenbankanbindung an das World Wide Web (WWW). Die bestehenden unterschiedlichen Techniken werden auf ihre Stärken und Schwächen analysiert. An die Datenbanksysteme (DBS) ergeben sich für WWW-basierte Anwendungen neue Anforderungen, welche bei der Entwicklung zukünftiger Techniken berücksichtigt werden müssen. Dies wird am Ende des Artikels behandelt.*

8.1 Inhalt

Zunächst wird ein kurzer Überblick über das WWW und die Programmierprache Java gegeben. Anschließend werden einige Beispiele zu sinnvollen Datenbankanwendungen im WWW vorgestellt. Daraufhin folgt der Schwerpunkt des Artikels, die Darstellung der existierenden Techniken der Datenbankanbindung. Am Ende wird ein Ausblick auf kommende Entwicklungen gegeben. Der Artikel lehnt sich in Inhalt und Gliederung sehr stark an [Loe97] an.

8.2 Grundlagen

Um die folgenden Ausführungen besser verstehen zu können, werden zunächst einige einleitende Anmerkungen zum WWW und zu Java gemacht.

8.2.1 Das World Wide Web

Das World Wide Web (WWW) besteht aus einem weltweiten Netz von Server-Rechnern. Diese Web-Server können von Client-Rechnern über das Internet angesprochen werden. Um als Client eine Verbindung zu einem Web-Server aufbauen zu können, genügt ein Anschluß zum Internet über die Telefonleitung und ein Computer mit Client-Software (einem

Browser). Der Server stellt Hypertext-Dokumente in HTML-Form (Hypertext Markup Language) zur Verfügung. Zum Übertragen wird das zustandslose HTTP-Protokoll (Hypertext Transfer Protocol) verwendet. Der Browser interpretiert die HTML-Dokumente und stellt sie für den Benutzer am Bildschirm dar. Das gewünschte Dokument wird im Browser durch eine URL (Uniform Resource Locator) angegeben, welche die Protokollart, den Servernamen sowie das Verzeichnis und den Dateinamen enthält, z.B.
http://www.iona.com/Orbix/Java/index.html.

Um nicht nur Seiten mit statischem Inhalt laden zu können, wurde das Common Gateway Interface (CGI, [CGI95]) entwickelt. Mit dessen Hilfe ist es möglich, ein Programm auf dem Server mit optionalen Parametern ausführen zu lassen. Anstatt dem Namen einer HTML-Seite wird der Namen des Programms angegeben. Die Parameter für das Programm können beispielsweise von einem HTML-Formular, welches vom Benutzer ausgefüllt wurde, stammen. Auf diese Weise arbeiten die meisten Suchmaschinen. Auf der Server-Seite kann nun, falls nötig, eine Datenbank angesprochen werden, welche die gewünschten Daten bereitstellt. Das Ergebnis wird nach Beendigung des Programms in Form einer dynamisch erzeugten HTML-Seite an den Client gesendet.

Es gibt zur Zeit zwei Protokolle, welche für Übertragungssicherheit zwischen Browser und Server sorgen können, zum einen SHTTP (Secure HTTP, [RS95]), eine Erweiterung von HTTP, zum anderen SSL (Secure Socket Layer, [FKK96]), 'ein von Netscape entwickeltes Übertragungsverfahren, welches die Verschlüsselung auf der Socket-Ebene einführt und somit auch von anderen Protokollen benutzt werden kann. [..] Da SSL schon in vielen Browsern und Servern eingesetzt wird, wird es wohl bald als Standard akzeptiert werden und SHTTP völlig verdrängen' [Loe97]

8.2.2 Java

1995 erhielt das WWW durch die von Sun entwickelte Programiersprache Java ([Gos96]) einen enormen Aufschwung. Java ist eine plattformunabhängige Sprache, die von einer virtuellen Maschine interpretiert wird. Es besteht die Möglichkeit, ein Java-Programm ('Applet') von einem Server zu laden und auf dem Client-Rechner mithilfe einer virtuelle Maschine im Browser auführen zu lassen. Applets können in HTML-Seiten integriert werden und bieten dadurch neue Möglichkeiten der Benutzerinteraktion. Mittlerweile wurde in den Sprachumfang von Java das JDBC (Java Database Connectivity), ein Teilpaket zur Ansteuerung von Datenbanken, aufgenommen. Damit bietet sich nunmehr eine elegante Lösung zur WWW-Anbindung von Datenbanken an.

8.3 DB-Anwendungen und das WWW

Im folgenden wird erläutert, warum der Einsatz des WWW für Firmen interessant ist. Einige sinnvolle DB-Anwendungen werden kurz vorgestellt.

Immer mehr Firmen erkennen mittlerweile, daß das WWW für ihr Unternehmen eine wichtige Bedeutung besitzt. Der erste Schritt besteht meist im Aufbau eines eigenen

Web-Servers, welcher als preiswertes Werbemedium dienen kann. Es werden dort Produktinformationen angeboten. Weiterhin können potentielle Kunden erreicht werden, zum Beispiel durch Bereitstellung von Demo-Versionen: Der Interessent füllt online ein Formular mit seinen Adressdaten aus und erhält im Gegenzug die Demo-Version eines Produktes. Registrierte Kunden haben auf einfache Weise die Möglichkeit, Kommentare zu den Produkten abzugeben; die Firma erhält also für die Weiterentwicklung ihrer Produkte wichtiges Feedback der Kunden.

Bei der Bereitstellung obiger Funktionalität fallen meistens viele Daten an (Kundendaten, Produktdaten). Diese werden sinnvollerweise durch ein DBS verwaltet. Die verschiedenen Anbindungsmöglichkeiten werden im nächsten Kapitel beschrieben. Im folgenden werden einige Beispielanwendungen aufgelistet:

- **Bereitstellung aktueller Börsendaten**
- **Nachrichten-Ticker**
- **Verfolgung von Vorgängen**: Falls das Unternehmen eine elektronische Auftragsbearbeitung einsetzt, so kann über das WWW der aktuelle Bearbeitungsstand abgefragt werden. Das kann Personalkosten (telefonische Auskunft) sparen.
- **Bestell-Center**: Produkte können über das WWW bestellt werden, oder es können Zusatzinformationen über die Produkte abgefragt werden.

Es sind weitere Anwendungsmöglichkeiten denkbar, die momentan noch kaum, in der Zukunft aber sehr wahrscheinlich stark genutzt werden:

- **Intranet**: Dieselbe Netzstruktur und Technologie, welche das Internet verwendet, kann genutzt werden, um ein firmeninternes Netz, ein Intranet, aufzubauen. Dadurch wird es möglich, firmeninterne Server analog den WWW-Servern einzurichten, welche den Mitarbeitern (Clients) ihre Dienste anbieten. Auf dem Firmenserver können unternehmensweite Anwendungen plattformunabhängig mit Java-Programmen realisiert werden. Somit ist es auf recht einfache Weise möglich, selbst in einem heterogen Netzwerk Enterprise-Informations-Systeme (EIS) oder Workflow-Management-Systeme (WMS) zu implementieren, da auf den Clients nur ein Browser benötigt wird, welcher die Java-Programme ausführen kann. Es ergibt sich ein enormes Einsparungspotential, da die Software-Systeme zentral erstellt und gepflegt werden können und nicht mehr wie bisher für verschiedene Zielplattformen entwickelt werden müssen.
- **Electronic Commerce**: 'Wird das Vertrauen bei den Geschäftspartnern, d.h. Kunden und Unternehmen in die Sicherheit der eingesetzten Technik weiter gestärkt, so wird der elektronische Handel sowie die Abwicklung von Bankgeschäften drastisch zunehmen. Neben der Einführung von elektronischem Geld ist besonders die Unterstützung von Kreditkarten interessant' ([Loe97]). Bisher sind die Sicherheitsanforderungen für das Internet noch nicht ausreichend erfüllt, die großen Kreditkartenunternehmen arbeiten jedoch mit namhaften Softwarefirmen zusammen an einem neuen Sicherheitsstandard ([VM96]).

Gerade diese beiden Einsatzmöglichkeiten des WWW in Verbindung mit Datenbanken bergen ein enormes Potential zur Produktivitätssteigerung und werden mit großer Sicherheit zu einer ständigen Weiterentwicklung der DB/WWW-Anbindungstechniken führen.

8.4 DB-Anbindungen an das WWW

Trotz der kurzen Entwicklungsgeschichte des WWW sind mehrere Stufen bezüglich der Datenbankanbindung zu erkennen. Im folgenden werden diese Stufen mit ihren Vor- und Nachteilen dargestellt und anhand von bestimmten Kriterien wie mögliche Transaktionsdauer oder Erstellungsaufwand einer HTML-Seite bewertet.

8.4.1 Die erste Generation

'Als Datenbankanbindungen der ersten Generation bezeichnen wir die vom Informationsanbieter für die einzelnen DB-Anfragen selbst erstellten Skripte oder Programme (meist Perl, C oder Shell-Skripte). Sie werden durch den CGI-Mechanismus des WWW-Servers aufgerufen und mit entsprechenden Parametern versorgt' [Loe97]. Die zu übertragende HTML-Seite und auch die DB-Anfrage ist im Skript fest kodiert. Somit muß für jede Aufgabe ein neues Skript erstellt werden. Nachfolgend ist beispielhaft ein in Perl geschriebenes CGI-Programm angegeben, welches Operationen auf einer mSQL-Datenbank ausführt. Daran wird deutlich, daß die Erzeugung einer HTML-Seite auf diese Weise recht mühselig und unübersichtlich ist.

```
#!/usr/bin/perl
# Msql-Package laden
use Msql;
# Seitenkopf ausgeben
print ”Content-type: text/html\ n \ n”;
# [..]
# Verbindung mit dem DB-Server herstellen:
$testdb = Msql->connect;
# Datenbank auswählen:
$testdb->selectdb (”bookmarks”);
# Datenbankanfrage stellen:
$sth = $testdb->query (”select name, url from bookmarks where url <> ” order by name”);
# Resultat ausgeben: print ”<TABLE BORDER=1> \ n”;
print ”<TR> \ n<TH>Name<TH>URL</TR>”;
$rows = $sth->numrows;
while ($rows>0)
{
@sqlrow = $sth->fetchrow;
print ”<tr><td>”,@sqlrow[0],”</TD><td>>A HREF=
```

```
\””, @sqlrow[1],”\”>”, @sqlrow[1],”</A></td></TR> \ n”;
$rows–;
}
print ”</TABLE> \ n”;
# Seitenende ausgeben
# [..]
```

Auf diese Art erstellte CGI-Programme besitzen viele Nachteile:

- **Zustandslosigkeit des HTTP**: Das verwendete Protokoll zwischen Server und Browser ist zustandslos. Dies hat zur Folge, daß die Dauer von Transaktionen auf die kurze Abarbeitung des CGI-Programms beschränkt ist, da die Verbindung zwischen Server und Browser nur für diesen Zeitraum besteht. Es können keine Sperren nach Abarbeitung des CGI-Programms gehalten werden.

- **Mehrstufige Aufrufhierarchie**: Für jede Benutzeranfrage muß zunächst das CGI-Programm gestartet werden, welches auf dem WWW-Server direkt oder in einem anderen Adreßraum liegen kann. Das CGI-Programm baut dann eine Verbindung zur Datenbank auf dem DB-Server auf. Dieser bearbeitet die Anfrage und schickt das Ergebnis zurück an das CGI-Programm. Dies wiederum leitet die Daten in Form einer HTML-Seite an den WWW-Server weiter. Da auch die normalen HTML-Seiten-Anforderungen über den WWW-Server abgwickelt werden, kann dieser zum Flaschenhals werden.

- **Unsicherheit über Ausführung**: Es gibt für den Benutzer keine Möglichkeit festzustellen, ob eine von ihm intendierte DB-Operation auch ausgeführt wird, da vor oder während der Operation die Verbindung ausfallen kann. Dies gilt zumindest dann, wenn, wie in den meisten Fällen, vom CGI-Programm keine positive Rückmeldung über eine erfolgreiche Bearbeitung gegeben wird.

- **Mangelnde Zugangskontrolle**: Falls der Benutzername nicht explizit abgefragt wird, bevor ein CGI-Programm gestartet wird, so führt dieses alle DB-Operationen als ein und derselbe Benutzer durch. 'Es ist also ohne Zusatzmaßnahmen nicht möglich, benutzerabhängig Informationen im Web bereitzustellen' [Loe97].

- **Schwere Lesbarkeit**: Wie in obigem Beispiel ersichtlich, ist der Quelltext der erstellten Ergebnis-HTML-Seiten schwer lesbar und schlecht zu warten, da normale HTML-Editoren nicht verwendet werden können.

- **Aufwendige Programmerstellung**: Der Programmierer muß sich selbst um den Zugriff auf Benutzereingaben, welche vom WWW-Server als Programmparameter bereitgestellt werden, kümmern.

Die letzten beiden Punkte werden durch die im folgenden beschriebene zweite Generation von DB-Anbindungen entkräftet. Alle anderen Punkte sind allerdings prinzipielle Probleme, die bei der Verwendung von CGI-Programme auftreten und können somit auch nicht von der zweiten Generation gelöst werden, da diese ebenfalls auf dem CGI-Mechanismus basiert.

Den vielen Nachteilen steht ein wichtiger Vorteil von CGI-Programmen gegenüber: **Flexibilität**: ' Da alle Programme selbst entwickelt werden müssen, steht auch die gesamte Funktionalität des DBS im Rahmen der oben genannten Einschränkungen zur Verfügung' [Loe97].

8.4.2 Die zweite Generation

Im folgenden werden die neueren Techniken der zweiten Generation vorgestellt, und es wird ein kurzer Produktüberblick gegeben.

Techniken

Die wesentlichen Neuerungen der zweiten Generation von DB-Anbindungen sind:

- **CGI/Server-Erweiterung**: Bei den meisten Web-Anbindungen der zweiten Generation wird nur noch *ein* allgemeines CGI-Programm aufgerufen. Dieses liegt bei manchen Systemen im Adreßraum des WWW-Servers. Es ist möglich, eine permanente Verbindung zum DB-Server zu halten, wodurch die Wartezeiten verkürzt werden.

- **Parameterübergabe**: 'Neben den bisher als Programmparametern in der URL übergebenen Anfrageinformationen muß in der Regel auch eine Datei spezifiziert werden, deren Inhalt je nach Produkt variiert. Eine häufige Lösung ist der Gebrauch von um Datenbankoperationen erweiterten HTML-Seiten.' [Loe97] Dies hat zur Folge, daß die gewünschten Ergebnisseiten mit üblichen Browsern und HTML-Editoren angesehen und erstellt werden können, was einen großen Vorteil bei der Wartbarkeit bedeutet. Das im folgenden gezeigte HTML-Dokument, welches mit W3-mSQL erstellt wurde, leistet das gleiche wie das weiter oben angeführte Perl-Skript, ist aber viel einfacher zu erstellen.

```
<HTML>
<head><title> The Bookmark Database </title><head>
<BODY>
<H1 align=center> The Bookmark Database </H1>

<!– DB-Verbindung aufbauen –>
<! msql connect>
<HR NOSHADE><P>
```

```
<!– Datenbank auswählen –>
<! msql database bookmarks>

<!– Datenbankanfrage stellen –>
<! msql query ”select url, name from bookmarks where url <> ” order by name” q1>

<!– Resultat formatiert ausgeben –>
<TABLE BORDER=2<TH> URL <TH> Name
<! msql print_rows q1 <tr><td>”, @q1.1 </TD><td>>A HREF= @q1.0, >q1.0
</td></TR>”>

<!– Ressourcen freigeben und Verbindung schließen–>
<! msql free q1>
<! msql close>
</BODY> </HTML>
```

- **Ablage der Seiten**: Bei manchen Systemen können die HTML-Seiten entweder im normalen Dateisystem oder direkt in der Datenbank gespeichert werden. Ersteres hat den Vorteil der einfachen Wartung mithilfe der üblichen Werkzeuge, letzteres bietet erweiterte Suchmöglichkeiten und die Verwaltung von Zugriffsrechten.

- **Transaktionsdauer**: 'Von den angebotenen Produkten bieten nur wenige die Möglichkeit, mehrere DB-Operationen zu einer Transaktion zusammenzufassen' [Loe97].

Produkte

Mittlerweile haben viele namhafte DB-Hersteller spezielle Web-Anbindungen ihrer Datenbanken auf Niveau der zweiten Evolutionsstufe im Angebot, so zum Beispiel Oracle, IBM (DB2), Object Design (ObjectStore), Sybase, die Software AG (Adabas) und O2. Ausführlichere Informationen über konkrete Produkte finden sich in [Loe97] und [Let96].

Die Deutsche Börse - Beispiel einer DB-Anbindung

Im folgenden wird ein konkretes Beispiel einer DB-Anbindung an das WWW anhand eines kommerziellen Projektes [SG97] beschrieben. Die Deutsche Gesellschaft für Wertpapierentwicklung (DGW), eine Tochter der Deutschen Börse bietet ihren Kunden (kleineren und mittleren Banken) ab sofort die Möglichkeit, im Internet Online-Banking für das Wertpapiergeschäft anzubieten. Bisher wurden die Kundenaufträge in den Banken ausschließlich über 3270-Terminals abgewickelt. 'Das Datenmanagement und die Transaktionsverwaltung erfolgten auf einem MVS-Großrechner der Börse, die Datenhaltung in der hierarchischen Datenbank IMS-DL1 und das Transaktionsmanagement über den IMS-Transaktionsmonitor' [SG97]. Ziel war es, in den Banken WWW-Server zu installieren, welche mit der Börsen-Datenbank verbunden sein sollten, so daß ein Bankkunde die Möglichkeit hätte, den WWW-Server anzuwählen und über ihn seine Geschäft online abzuwickeln. Weiterhin sollten auch die Bankangestellten die Möglichkeit besitzen, über ihre PC's oder Workstations auf den Datenbestand zuzugreifen. Die Oberfläche sollte in beiden Fällen ähnlich bedienbar sein. Die Datenbank sollte weiterhin auf dem Host bleiben.

Man entschied sich für 'VisualAge for Smalltalk' (VA ST) als Entwicklungsplattform. Sie besitzt eine Reihe entscheidender Vorteile:

- Ausgereifte, stabile und performante Entwicklungsumgebung.
- Rein objektorientiertes Datenmodell.
- Umfangreiche Klassenbibliothek und Offenlegung aller wesentlichen Programmquellen.
- Wiederverwendbare, vorgefertigte Komponenten.
- Portabilität für OS/2, Windows und verschiedene Unix-Derivate.

Viele der IBM-Connectivity-Produkte sind in VA ST verfügbar - und zwar in der Form von VisualAge-Parts ([IBM1]. Zwei davon sind für das Projekt besonders interessant:

- **Web Connection Parts** [IBM2]: Diese Komponente ermöglicht es, auf VAST-Applikationen eine HTML-Benutzungsschnittstelle aufzusetzen. Diese kann über einen Web-Server angeboten werden, so daß Homebanking-Kunden transparent Funktionen der bereitgestellten Smalltalk-Applikationen nutzen können. Natürlich können dabei nicht alle grafischen Basiseinheiten (Widgets) umgesetzt werden, dies ist bei der Entwicklung der Smalltalk-Anwendung zu berücksichtigen, ebenso wie die Unterschiede in der Benutzerinteraktion. Einem einfachen Ersetzen von Client/Server-durch Web-GUI's sind noch deutliche Grenzen gesetzt.
- **CICS & IMS Connection Parts** [IBM3]: Diese Komponenten ermöglichen es, bestehende Host-Transaktionen unverändert in VA ST anzusprechen. Die Daten, welche die Transaktionen bereitstellen, werden ebenfalls über nicht-visuelle Parts zur Verfügung gestellt. 'CICS & IMS Connection kapselt Basisprotokolle. [..] Die einfache Einbettung bestehender Transaktionen schafft einen erheblichen Produktivitätsgewinn gegenüber einer Direktansprache der Datenbank.' [SG97]

Der WWW-Server ist über CGI mit den Web-Connection-Parts auf dem Applikationsserver verbunden. Die Web-Connection-Parts leiten Datenbankanfragen an die Smalltalk-Anwendung beziehungsweise die CICS & IMS Connection Parts weiter, welche wiederum mit dem Host verbunden sind.

Die Web-Connection-Parts bieten weiterhin Mechanismen an, um das Problem der Zustandlosigkeit des HTTP zu lösen. Sie erlauben darüberhinaus die Integration von Java-Applets.

Das Projekt wurde erfolgreich abgeschlossen. VisualAge in Verbindung mit den angesprochenen Parts 'hat sich als taugliches und stabiles Werkzeug zur produktiven Implementierung komplexer Applikationen erwiesen' [SG97]

8.4.3 Die dritte Generation (Java, JDBC, CORBA)

Während die ersten beiden Generationen noch auf CGI basieren, wird in der dritten Generation Java zur DB-Anbindung eingesetzt. Es besteht die Möglichkeit, durch die Java-Anwendung eine direkte Verbindung zum DB-Server herzustellen, die während der gesamten Browser/Server-Verbindung aufrechterhalten werden kann. Diese Lösung hat gegenüber den CGI-Programmen eine Reihe von **Vorteilen**:

- **Transaktionsdauer**: Da die DB-Verbindung länger aufrechterhalten werden kann, steht der Realisierung von komplexeren Transaktionen nichts mehr im Wege.

- **Komplexe Programme**: 'Der Browser kann im Gegensatz zu den CGI-Lösungen, wo er nur zur Dateneingabe diente, nun als Ausgabeinstrument einer komplexen Anwendung genutzt werden. [..] Bedingungen können schon direkt bei der Eingabe und nicht erst bei der Anfrageausführung überprüft [..] werden. [..] Entwickelte Programme sind plattformunabhängig' [Loe97].

- **Direkte Verbindung**: Durch die direkte DB-Verbindung können die üblichen Protokolle zur Transaktionssicherung (2PC) eingesetzt werden. Somit sind Java-Lösungen auch für diesbezüglich kritische Anwendungen wie Internet-Banking interessant.

- **Aufhebung des Flaschenhalsproblems**: 'Alle DB-Aufrufe gehen ohne Umweg direkt an den DB-Server, die Zwischenstufen WWW-Server und CGI-Programm entfallen' [Loe97].

Demgegenüber stehen aber auch einige **Nachteile** beim Einsatz von Java anstatt CGI:

- **Längere Ladezeiten**: Um die Funktionalität von Java nutzen zu können, muß die Java-Anwendung samt der von ihr benötigten Java-Klassen beim erstmaligen Öffnen der HTML-Seite zunächst vom WWW-Server geladen werden. Dies kann bei den geringen Internet- Bandbreiten stark ins Gewicht fallen.

- **Verlust von HTML**: HTML steht nur noch zur Anwendungseinbettung, nicht mehr zur Eingabe und Ergebnisdarstellung zur Verfügung. Dies muß in Java programmiert werden, was einen höheren Aufwand bedeutet.

- **Sicherheitsproblematik**: 'Arbeitet man nicht in einem abgeschirmten Intranet, besteht bei der Übertragung ohne Verschlüsselung die Gefahr der Manipulation' [Loe97].

- **Einschränkung von Verbindungsmöglichkeiten**: Wegen der strengen Sicherheitsrichtlinien von Java können Applets normalerweise nur Verbindungen zu ihrem eigenen WWW-Server einrichten. Das bedeutet, WWW-Server und DB-Server müssen auf demselben Rechner laufen, welches Engpässe zur Folge haben kann. Dies kann allerdings durch Einrichtung eines Kommunikations-Servers auf dem WWW-Server umgangen werden, der die DB-Aufrufe abfängt und an den zuständigen DB-Server weiterleitet.

Techniken und Produkte

Während bei CGI-Programmen in der Regel mit SQL/OQL-Anfragen gearbeitet wird, gibt es zur Realisierung der direkten DB-Verbindung unter Java mehrere Möglichkeiten:

- **Java Database Connectivity (JDBC)**: JDBC bietet in Anlehnung an das weit verbreitete ODBC eine SQL-Schnittstelle für Java. Einen guten Überblick über die mögliche Funktionalität und verschiedene Treiber-Philosophien bietet [Pie97b].

- **Spezifische Aufrufschnittstelle**: Wie bei C++ können auch für Java von den DBS- Herstellern DBS-spezifische Schnittstellen zur Verfügung gestellt werden. Eine Kurzübersicht über bisher realisierte Schnittstellen findet sich in [Loe97].

- **CORBA**: CORBA ist eine von der OMG spezifizierte Architektur, welche auf verteilten Systemen die Ortstransparenz von Objekten und deren Methoden ermöglichen soll. 'Da in CORBA bereits Schnittstellen zu verschiedenen DBS bestehen, kann nun durch die Realisierung eines Java-CORBA-Clients ein DB-Zugriff vom WWW-Browser aus geschaffen werden' [Loe97]. Der Kommunikationsserver wäre in diesem Fall ein ORB (Object Request Broker), der zwischen Browser und DB-Server vermittelt. Möglichkeiten und Grenzen des Einsatzes von CORBA in DB-basierten Client/Server-Anwendungssystemen werden ausführlich in [Sel97] diskutiert. Die Autoren dieses Artikels kommen zu dem Schluß, CORBA sei in seiner jetzigen Spezifikation zwar gut für auftragsbezogene, aber weniger für datenintensive DB-Anwendungen geeignet. Ursache dafür sei, daß 'die den existierenden CORBA-Implementierungen zugrundeliegenden Verarbeitungsparadigmen ausschließlich *Object Shipping* und kein *Data Shipping* unterstützen'. Eine weitere ausführliche Diskussion der Integration von Java und CORBA findet sich in [Ion96], wobei auf eine konkrete CORBA-Implementierung, nämlich Orbix von Iona Technologies eingegangen wird.

Integration von CORBA, Java und dem Web anhand eines Beispiel-Projekts

Die Firma SYSTOR AG ließ in einem Testprojekt ([HN97]) untersuchen, ob Java in Kombination mit CORBA die herkömmlichen Technologien im Inter-/Intranet heute schon ablösen oder ergänzen könnten. Zu diesem Zweck realisierte man eine Electronic-Commerce-Lösung mit häufig benötigtem Workflow (Anbieten und Bestellen von Produkten, Entgegennahme von Bestellungen, Ausliefern von Produkten). Als Entwicklungswerkzeuge wurden ausschließlich Produkte der Firma Sunsoft eingesetzt:

- NEO 2.0 (Beta), eine CORBA-Implementierung
- Joe 1.0, ein Java-ORB (Object Request Broker)
- JDK 1.0.2 (Java Development Kit)

Für die Applikation werden mehrere Java-Applets benötigt:

- Das Java-Applet, über das der Kunde seine Bestellungen ausführt.
- Die Java-Applets des Anbieters, die zum Administrieren des Lagers, des Katalogs und der Bestellungen erforderlich sind.

Der Funktionsablauf der Anwendung gestaltet sich folgendermaßen: Der Kunde wählt den Web-Server des Anbieters an. Er lädt das Kunden-Applet mitsamt den Java/Joe-Klassen, die unter anderem den Java-ORB enthalten. Das Kunden-Applet sucht dann im Netz nach seinem Dienstgeber-Objekt und stellt eine Verbindung zu diesem her. Die Kommunikation zwischen dem Java-ORB im Kunden-Applet und dem NEO-ORB, der die Dienste bedient, erfolgt dabei über das IIOP (Internet-Inter-ORB-Protokoll, 'ein von der OMG definiertes Netzwerkprotokoll, daß es Prozessen (Services) erlaubt, mit anderen Diensten, die unter einer CORBA-IIOP-kompatiblen Implementierung laufen, zu kommunizieren. Dabei spielt es keine Rolle, unter welchem Betriebssystem und auf welcher Rechnerarchitektur die verschiedenen Dienste laufen' [HN97] (über IIOP siehe auch [OMG97]).

Die Datenhaltung wird über die Persistenz-Mechnismen der CORBA-Implementierung realisiert. Die konkrete Art der Speicherung (relationale / OO-DB, Flat File) kann vor der Applikation verborgen werden.

Da Joe noch nicht die Funktionaliät des JDK 1.10 (komprimierte Archive) nutzt, gestalten sich die Ladezeiten für das Kunden-Applet noch extrem lang. Ansonsten kommen die Autoren von [HN97] aber zu einem positiven Projekt-Fazit:

'Will man seine zukünftigen Anwendungen auf Web-Technologien migrieren - sei es im Inter- oder im Intranet - und eine echte dreischichtige Client/Server-Architektur betreiben sowie deren Vorteile nutzen, dann kommt man an Java-basierten ORB's, die in Web-Browsern ablaufen, nicht vorbei. Ob man dabei nun Joe von der Firma Sunsoft oder Produkte anderer Hersteller (z.B. Iona, Visigenics) berücksichtigt, spielt sicherlich keine Rolle. Spätestens wenn alle Object Request Broker volle IIOP-Unterstützung anbieten - was in naher Zukunft sicherlich der Fall sein wird - , ist ein echter netzwerkweiter Zugriff auf alle angebotenen Dienste im Netz möglich. *IIOP ist der Schlüssel zum Erfolg für CORBA, Java und das Web.* Dienste im Netz, die untereinander und mit den Applikationen (mehrheitlich Java-Applets) über ein und dasselbe Protokoll (IIOP) kommunizieren, werden das Netz endgültig zum Computer machen.'

8.4.4 Zukünftige Technologien (ActiveX, Java Erweiterungen)

Im folgenden werden Technologien vorgestellt, die in der Zukunft eine wichtige Rolle bezüglich DB-Anbindungen an das WWW spielen könnten.

ActiveX

ActiveX [Joh97] ist eine von Microsoft entwickelte Architektur, die sowohl mit Java als auch mit CORBA konkurrieren soll. Es ist eine Weiterentwicklung der ursprünglichen OLE-Architektur, neu hinzugekommen sind Fähigkeiten der verteilten Ausführung mithilfe des DCOM-Modells. Es wurde von Microsoft als Reaktion auf den starken Erfolg von Java entworfen, welcher die Marktdominanz von Microsoft zunehmend gefährdet. Speziell die seit Erscheinen des JDK 1.1 verfügbaren Java Beans (Java-Komponenten, [Pie97a]) stellen im Bereich der Componentware eine große Herausforderung für Microsoft dar. Die DCOM/ActiveX-Architektur soll eine Alternative zu CORBA/Java darstellen. Welche der beiden Architekturen als Basis für Componentware besser für das WWW geeignet ist, wird in [OB97] untersucht. Java besitzt den Vorteil der Plattformunabhängigkeit, DCOM ist momentan dagegen nur auf Windows-Plattformen verfügbar. Die Software AG, welche mit Microsoft eine Kooperationsvereinbarung eingegangen ist, arbeitet aber an einer Umsetzung auf andere Systeme (Unix, MVS) [Han97].

Das gravierendste Problem von ActiveX, welches einem umfassenden Einsatz im Internet im Wege steht, sind seine Sicherheitslücken. ActiveX-Objekte enthalten binären Code, somit besteht theoretisch die Mögichkeit, beliebig auf lokale Daten des Clients zuzugreifen. Dieses Problem wird auch durch digitale Signaturen nur teilweise entschärft. Java bietet demgegenüber bessere Sicherheitsmechnismen, da der Java-Code erst von einer virtuellen Maschine überprüft werden kann, bevor er zur Ausführung gelangt.

Ein Vorteil von ActiveX gegenüber Java ist die Geschwindigkeit, der binäre Code wird schneller als der Java-Code ausgeführt und wurde außerdem für die Microsoft-Plattformen optimiert.

Mittlerweile existieren viele Bestrebungen, Schnittstellen zwischen Java Beans, ActiveX/DCOM und CORBA zu schaffen. Jede der Technologien besitzt ihre Vorteile, so daß kurzfristig nicht mit einem einzigen festen Standard gerechnet werden kann.

Es ist möglich, mit ActiveX oder Java Beans Komponenten zu erstellen, welche die Datenbankanbindung vornehmen. Eine solche Komponente kann in eine HTML-Seite eingebunden werden.

Java Erweiterungen

Bezüglich der Ladezeiten und auch der Sicherheitsproblematik bestehen bei Java noch Verbesserungsmöglichkeiten. Beides wird in der neuen Java-Version 1.10 besser gelöst. Es gibt ein erweitertes Security-API und Java-Archive, welche komprimierte Module enthalten, die auf Wunsch vom Server geladen werden können. Darauf baut das Java Electronic Commerce Framework (JECF [Jav96a]) auf, welches sich auf Anwendungen im finanziellen Bereich konzentriert. Hierbei werden sogenannte Service-Kassetten eingeführt, die aus

Java-Modulen bestehen und in Java-Archive komprimiert und signiert in einer Datei zusammengefaßt werden. Eine auf dem Client nicht vorhandene Kassette wird im Gegensatz zur bisherigen Technik nicht nur in den Adreßraum des Browsers geladen, sondern zuvor dauerhaft im lokalen Dateisystem abgelegt. Dadurch werden bei mehrmaliger Nutzung lange Wartezeiten vermieden. In einer solchen Kassette kann zum Beispiel DB-Funktionalität bereitgestellt werden. 'Da die Herkunft der Programme über Authentisierung abgesichert ist, könnnen gewisse Sicherheitsrestriktionen aufgehoben werden. Daher sind nun auch Verbindungen zu anderen Rechnern als dem WWW-Server möglich [..]. Das Problem der eingeschränkten Verbindungsmöglichkeiten und der damit verbundenen potentiellen Leistungsengpässe wird somit auch gelöst.' [Loe97] Zum Zeitpunkt der Artikelerstellung (Mai 97) war das Java ECF erst in einer Beta-Version verfügbar. Somit können noch keine verläßlichen Aussagen über seine Praxistauglichkeit gemacht werden.

8.5 Künftige Anforderungen

Traditionelle DBS unterstützen die Anbindung an das WWW bisher nur unzureichend. Im folgenden werden die wichtigsten Forschungsschwerpunkte hinsichtlich der Synthese von WWW und DBS genannt:

- **Neue Datentypen**: Für die multimedialen Daten (Text, Bilder, Filme, Musik und auch Programme) werden neue Datentypen im DBS benötigt.

- **Dateisystem**: Die HTML-Seiten sollten in einem DBS gespeichert werden. Das DBS sollte eine Verzeichnisstruktur bereitstellen, um dem Benutzer eine übersichtliche Navigation durch WWW-Server-Seiten zu ermöglichen. Die bisherige Bedienung der WWW-Server ist unzureichend.

- **Intelligente Suche**: Sind die HTML-Seiten in einem DBS gespeichert, so können erweiterte Suchmöglichkeiten (Volltextsuche, Autoren, Inhaltsstrukturen) angeboten werden.

- **Versionierung**: Vom DBS sollte die Möglichkeit der automatischen Erhaltung von älteren Versionen einer Seite angeboten werden. Dadurch lassen sich viele Anwendungen, wie zum Beispiel das Bereitstellen von älteren Nachrichten, sehr einfach realisieren. Auch die Konsistenz kann bei Verwendung von Versionierung leichter gewahrt werden (es wird erst auf die neue Version umgeschaltet, wenn alle zugehörigen Datenseiten samt Querverweisen aktualisiert wurden).

- **Referentielle Integrität**: Die Wartung der internen Querverweise der HTML-Seiten sollte vom DBS unterstützt werden.

- **Einfaches Editieren**: 'Das Editieren der durch das DBS verwalteten Dokumente muß weiterhin mit den normalen Werkzeugen ohne größeren Aufwand möglich sein' [Loe97].

- **Lange Transaktionen**: Sie sollten vom DBS unterstützt werden. Probleme wie Sperren von Daten und Deadlocks sind zu beachten.

- **Authentisierung**: Anmeldung durch Benutzername und Passwort ist im Internet für kritische Anwendungen (wie Bankgeschäft) nicht ausreichend sicher. Es werden neue Techniken wie z.B. Transaktionsnummern (TAN's) sowie Digitale Signaturen benötigt. Das DBS muß diese neuen Techniken unterstützen.

- **Legacy-Systeme**: Oft werden die Daten eines Unternehmens nicht nur von einem WWW-Server, sondern auch von eigenen traditionellen Applikationen (Legacy-Systeme) benutzt. Replikation sowie Verteilung der Daten auf mehrere Systeme kann oft nur durch Middleware gelöst werden. 'Lösungen hierzu können u.a. aus dem Bereich von Multi-DBS (MDBS) und Föderierten DBS (FDBS) kommen. Wichtig erscheint vor allem die sinnvolle Intergration der Legacy-Systeme. Dabei ist auch eine Verbindung von relationalen und objektorientiereten DBS wichtig, da die Daten meistens noch in relationalen DBS und die Systeme für die Verwaltung der HTML-Seiten meist aber in OO-DBS realisiert sind bzw. werden.

- **Anwendungen**: Software-Entwickler sollten die neuen Möglichkeiten, die sich durch Technologien wie ActiveX oder Java Beans bieten, ausnutzen. Es wird prinzipiell möglich, komplexe Anwendungen aus voneinander unabhängigen Komponenten zusammenzusetzen. Man spricht dabei auch von 'Componentware' (siehe dazu auch [SC97]). 'Wird die Anwendung in einer plattformunabhängigen Sprache entwickelt [..], so lassen sich auf relativ einfache Weise verteilte Anwendugen in einem heterogenen Umfeld realisieren' [Loe97].

8.6 Zusammenfassung

Es wurden einige Techniken der DB-Anbindung an das WWW vorgestellt. 'Eine optimale Lösung existiert bislang nicht, es gibt einen "Trade-Off" zwischen der Mächtigkeit der Ansätze und der Einfachheit der Anwendungserstellung. Wünschenswert ist hier eine Lösung mit der Mächtigkeit von Java und der Einfachheit von HTML. Als Entwicklungstrend ist daher auch die Ergänzung bestehender CGI-basierter Produkte um weitere Funktionalität festzustellen' [Loe97]. Anhand des beispielhaft erläuterten Projekts der Deutschen Börse wird ersichtlich, daß die dritte Generation der DB-Anbindungen (Java) noch am Anfang steht, Industriestandard ist nach wie vor CGI in Verbindung mit diversen Zusatzwerkzeugen.

Bei den DBS selbst bestehen bezüglich der WWW-Unterstützung noch viele Mängel, die teilweise einigen Forschungsaufwand für ihre zufriedenstellende Lösung benötigen werden. Die reine Anbindung der DBS an das WWW dagegen ist schon ausreichend gelöst.

Kapitel 9

JDBC
Datenbankzugriff mit Java

Klaus Wehrle

9.1 Einführung

Das World Wide Web (WWW) ist ein architektonisches Rahmenwerk für den Zugriff auf verknüpfte Dokumente, die auf Millionen von Maschinen überall im Internet verteilt sind. Innerhalb von 5 Jahren entwickelte es sich von einer Möglichkeit der Verteilung wissenschaftlicher Daten zu einer Anwendung, die die meisten Leute für das Internet halten.
Seine enorme Beliebtheit stammt aus der Tatsache, daß es eine grafische Oberfläche hat, die für Anfänger sehr leicht zu bedienen ist, und daß es eine unglaubliche Fülle von Informationen zu jedem erdenklichen Thema gibt. Auf jeder aktuellen Computerplattform gibt es Web-Browser, die es dem Heimanwender erlauben, dieses riesige Informationsangebot auf einfache Weise zu nutzen.
Eine der wichtigsten Eigenschaften des WorldWideWeb gegenüber anderen Medien, wie Zeitung, Radio oder Fernsehen, ist die Möglichkeit der Interaktion für den Benutzer. Wo sich diese Interaktionsmöglichkeiten allerdings auf das Klicken von Hyperlinks beschränken, verliert dieses Medium sehr schnell an Attraktivität und Nutzwert.
Diese Beschränkungen sind zur Zeit auch das größte Problem des WorldWideWeb. Sie sind jedoch in dessen Grundgerüst enthalten, nämlich dem Seitenbeschreibungsformat HTML (Hypertext Markup Language) und dem Seitentransportprotokoll HTTP (Hypertext Transfer Protocol), und können deshalb auch nicht so einfach beseitigt werden. Trotzdem werden mit vielen Tricks und neuen Ideen (z.B. Java) immer mehr *stark interaktive* Anwendungen für das Web entwickelt.
Mit *stark interaktiven* Anwendungen sind solche gemeint, deren Interaktionsmöglichkeiten deutlich über die reiner Hypertextsysteme hinausgehen. Der Benutzer sollte individuelle Eingaben (in welcher Form auch immer) machen können, und so individuelle Ergebnisse erhalten. Dadurch kann sich ein immer wieder veränderndes Szenario abspielen, im Ge-

gensatz zu einem statischen Hypertextsystem, wo ein bestimmter Querverweis immer zum selben Ergebnis führt.
Ein klassisches Beispiel für solch eine Anwendung ist der Online-Katalog der Universitäts-Bibliothek Karlsruhe, bei dem man, anhand von bestimmten Suchkriterien, eine Auswahl von Büchern bekommt, die man dann wahlweise bestellen kann. Hinter diesen WWW Seiten der Unibibliothek steht eine sehr große Datenbank, aus der die Daten geholt werden, und in der die Bestellungen eingetragen werden.
Ohne die Benutzung dieser Datenbank könnte man diese interaktive Dienstleistung für die Benutzer nicht zur Verfügung stellen. Durch den Einsatz von Datenbanken ist es möglich, ganz neue Anwendungsfelder für das World Wide Web zu erschließen, die mit einem reinen Hypertextsystem nicht denkbar wären.
Im Folgenden werden die Probleme der bisherigen Techniken des Zugriffs auf Datenbanken über das WWW diskutiert, und dann eine neue Möglichkeit vorgestellt, nämlich JDBC von Sunsoft, welches den effizienten und einfachen Zugriff auf relationale Datenbanken über das Word Wide Web ermöglichen soll [Lab96,Lab97].

9.1.1 Neue Möglichkeiten der Interaktion im Word Wide Web

Mit der Programmiersprache Java hat Sun Microsystems dem WWW eine neue Möglichkeit der Interaktion hinzugefügt. Mit Hilfe von Java kann man nicht nur Text und Bilder darstellen, sondern man kann einer Web-Seite zum Leben verhelfen, d.h. in der Web-Seite kann man bestimmte Bereiche definieren, in denen kleine (oder auch größere) Programme, sog. Applets, ablaufen. Diese Applets werden in Java programmiert, einer objektorientierten Sprache, die Ähnlichkeiten zu C++ aufweist, jedoch aus Sicherheitsgründen fehleranfällige Bestandteile ausgeklammert hat, wie z.B. Pointer, u.ä. Das macht Java zu einer zuverlässigen, gut strukturierten und sicheren Programmiersprache.
Java ist eine interpretierte Sprache, d.h. der Quellcode wird in einen sog. Byte-Code compiliert, der maschinenunabhängig ist und auf der jeweiligen Plattform durch die Java Virtual Machine (JVM) in plattformabhängige Befehle umgesetzt und ausgeführt wird. Dadurch wird nicht nur die Unabhängigkeit von dem jeweiligen Rechner erreicht (d.h. ein Java Programm läuft auf allen Rechnern, die über eine JVM verfügen), sondern die Interpretation des maschinenunabhängigen Codes erlaubt es auch, zur Laufzeit verschiedenste Sicherheitskriterien zu überwachen.
Ein Java Applet darf beispielsweise nicht auf lokale Daten zugreifen. Denn es ist meist nicht bekannt (oder überprüfbar), auf welche Daten zugegriffen wird. Außerdem ist es einem Java Applet nicht erlaubt, native Programme aufzurufen, d.h. Programme, die nicht zu hundert Prozent in Java verfaßt sind. Denn man weiß auch bei diesen Programmen nicht, ob es sich um *'schwarze Schafe'* handelt (es kann nur Java Byte-Code überprüft werden).
Mit Java ist es auch möglich, mit anderen Rechner in Interaktion zu treten, ohne dabei das HTTP Protokoll benutzen zu müssen (s. Abschn. 2 - Probleme mit HTTP). Es können somit die oben angesprochenen stark interaktiven Anwendungen entwickelt werden.
Mit Java und speziell dem Abstract Windowing Toolkit (AWT) erhält man auch die Mög-

lichkeit benutzerfreundliche grafische Oberflächen für ein Applet zu entwerfen. Damit kann man beispielsweise ein Front End zu einer Datenbank entwerfen, welches mit Eingabefeldern, Auswahlknöpfen, u.v.m. die Eingabemöglichkeiten einer herkömmlichen Datenbank mit GUI, wie z.B. MS-Access, erlaubt.
Doch wie oben angesprochen, benötigen diese stark interaktiven Anwendungen oft auch die Möglichkeit des Datenbankzugriffs. Ab der Version 1.1 des Java Development Kits (JDK) hat man nun eine Schnittstelle, mit deren Hilfe man auf Datenbanken zugreifen kann. Zuvor werden jedoch noch zwei der bisher am meisten benutzten DB-Schnittstellen vorgestellt und deren Probleme diskutiert. Die Schnittstelle JDBC - Java DataBase Connectivity wird anschließend ab Abschnitt 3 vorgestellt.

9.2 Bisherige DB-Zugriffsmöglichkeiten im WWW

In der Zeit vor JDBC und Java war es sehr umständlich, über eine Web-Seite Zugriff auf eine Datenbank zu erhalten.
Wir betrachten noch einmal das Beispiel der Universitätsbibliothek Karlsruhe. Dort muß bei jeder Bestellung die Benutzernummer und das Paßwort erneut eingegeben werden, auch wenn dies am Benutzerterminal in der Bibliothek nicht nötig ist. Der Grund liegt darin, daß der Webserver keine Informationen oder Daten über bereits getätigte Zugriffe (Transaktionen), oder geöffnete Datenbestände speichert. D.h. es können nur vorher festgelegte Transaktionen durchgeführt werden. Vor allem fehlt dem Benutzer die Möglichkeit, den Ablauf einer Transaktion zu steuern.
Größere Transaktionen sind nur mit einigen Tricks möglich (z.B. implizite Verwendung von Transaktionsnummern). Der Grund der ganzen Misere liegt im Protokoll HTTP (Hypertext Transfer Protokoll) begründet, welches den Transport der Webseiten zwischen Client (Browser) und Server vornimmt.
Wenn der Client eine Web-Seite anfordert, so baut er eine TCP-Verbindung zum entsprechenden Webserver auf und übergibt diesem den Namen der HTML-Datei. Der Server sendet daraufhin die Seite an den Client, und die TCP-Verbindung wird sofort wieder geschlossen. Der Server hat daraufhin keinerlei Informationen mehr zur Verbindung gespeichert. Man spricht auch von einem **Stateless-Protocol**.
Besteht eine Seite aus mehreren Teilen, z.B. Bildern, Audio-Dateien, ... , so wird für jeden dieser Teile eine eigene TCP-Verbindung aufgebaut. Der Server merkt nicht einmal, daß alle diese Teile zum selben Empfänger gehen. Auch kann er keine Verbindung zum Client mehr herstellen, da er nicht gespeichert hat, wohin er welche Daten geschickt hat.
Um nun doch auf Datenbanken zugreifen zu können, benutzt man einige Tricks, die die Beschränkungen des HTTP-Protokolls umgehen. Im folgenden werden zwei Varianten vorgestellt: CGI-Skripte (Common Gateway Interface) und proprietäre Server.

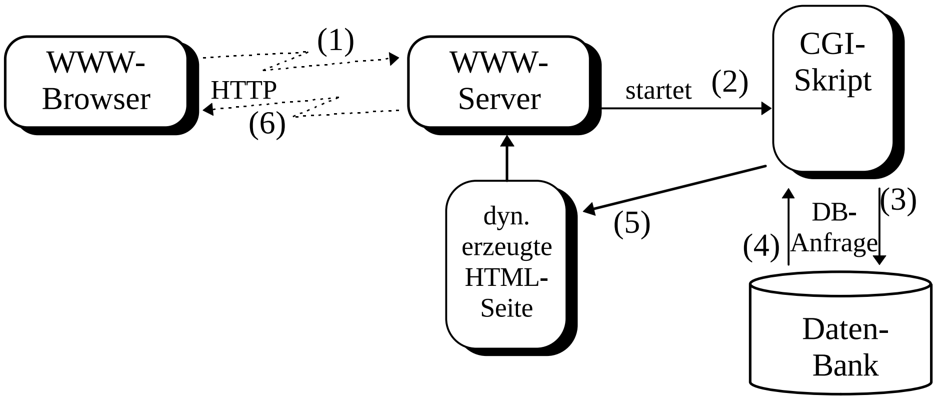


Abbildung 9.1: CGI - Common Gateway Interface

9.2.1 CGI-Skripte

Eine Information, die man aus dem World Wide Web ziehen möchte, liegt eigentlich nur in Form einer (statischen) HTML-Datei vor. Da man aber beim Zugriff auf Datenbanken meist nicht die ganze Datenbank in Form einer HTML-Seite sehen will, oder für jede mögliche Anfrage eine HTML-Seite vorsehen kann, muß man versuchen, spontan auf die entsprechende Anfrage zu reagieren und eine Seite speziell mit den Ergebnissen dieser Anfrage dynamisch zu erstellen.
Dies ist die Aufgabe eines Common Gateway Interface (CGI). CGI definiert einen Standard, den solche Gateways zwischen Webserver und Datenbank erfüllen müssen. Das Gateway kann aus einem Skript oder Programm bestehen, welches in C/C++, Perl, etc. geschrieben ist.

Den Ablauf eines Datenbankzugriffs über ein Common Gateway Interface zeigt Abbildung 1. Der Ablauf ist durch die Numerierung gekennzeichnet.

1. Der Client (z.B. Netscape) stellt eine Anfrage, die über das HTTP-Protokoll an den Webserver gesendet wird.

2. Der Webserver erkennt, daß es sich um eine Anfrage an ein bestimmtes CGI handelt, und startet das Gateway-Programm. Außerdem übergibt er dem Gateway die Daten, mit denen der Benutzer seine Anfrage spezifiziert hat. (Dies ist auch noch ein großes Problem, da dies in HTML ursprünglich nicht vorgesehen war. Es ist aber ab Version 2.0 möglich, Daten in speziellen Eingabefelder einzugeben, und zum Server zu übertragen).

3. Das Gateway erkennt die Anfrage, und setzt sie in eine Datenbank-spezifische Form um, z.B. in eine SQL-Anfrage. Die Anfrage wird an die Datenbank geschickt.

4. Nach der Bearbeitung der Anfrage sendet die Datenbank die Ergebnisdaten an das Gateway zurück.

5. Das Gateway erzeugt aus den empfangenen Daten eine HTML-Seite, und übergibt diese an den Webserver.

6. Der Webserver sendet die HTML-Seite mit dem Ergebnis der Datenbankanfrage an den Client zurück.

Es besteht auch die Möglichkeit, daß die Ergebnisdaten auf andere Weise zum Benutzer zurückkommen, z.B. über eMail, oder die Daten werden einfach nur in der Datenbank gespeichert.
CGI war lange Zeit die einzige Programmierschnittstelle zur Erweiterung der Funktionalität eines Webservers. Die meisten (interaktiven) Anwendungen, die heute im Einsatz sind, basieren auf Common Gateway Interfaces. Jedoch ist diese Art und Weise des Datenbankzugriffs sehr umständlich und erfordert vor allem sehr viele Tricks und einen hohen Programmieraufwand.
Eine zweite Möglichkeit, über das World Wide Web mit einer Datenbank in Verbindung zu treten, ist, einen proprietären Webserver zu benutzen.

9.2.2 Proprietäre Webserver

Aus der Sicht des DBMS ist ein Common Gateway Interface (CGI) ein Datenbank-Client, der die WorldWideWeb-Technologie versteht, d.h. die Anpassung findet auf der Client-Seite statt. Andererseits kann man sich die Anpassung aber auch auf Server-Seite vorstellen, d.h. einen Webserver, der eine (oder mehrere) Datenbankschnittstelle(n) unterstützt. In so einer Architektur ist die Fähigkeit zur Datenbankanbindung in den Webserver integriert (siehe Abbildung 2). Einige Beispiele für proprietäre Webserver sind z.B. der Microsoft Information Server oder der Oracle Webserver.
Jedoch sind diese Webserver sehr an spezielle Datenbanken angepaßt, und nicht allgemein einsetzbar.
Die in Abschnitt 2 betrachteten Datenbankschnittstellen haben also mehr oder weniger versucht, unter teilweise erheblichen Einschränkungen der Funktionalität (bei CGI) oder unter Anpassung an spezielle Datenbanksysteme (bei proprietären Webservern), die Beschränkungen des HTTP-Protokolls zu umgehen. Da denkt man sich vielleicht, wieso soll man denn dieses Protokoll überhaupt benutzen, wenn es nur Probleme bringt. Und genau dieser Gedanke führt zu JDBC.
Die Hoffnung der Internet-Welt ruht nun auf der Datenbankschnittstelle JDBC. Sie soll nach dem großen Erfolg der Programmiersprache Java, deren Vielfältigkeit und die Anzahl der Java- Applikationen noch vergrößern.

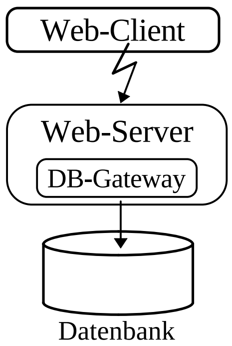


Abbildung 9.2: proprietäre Webserver

9.3 Java Database Connectivity (JDBC)

JDBC ist eine Klassenbibliothek, die es ermöglicht, aus einem Java-Applet oder einer Java- Anwendung heraus, auf relationale Datenbanken zuzugreifen. Sie besteht aus einer Menge von Klassen und Methoden, die in Java geschrieben sind, und die es einem Entwickler ermöglichen, ein reines Applet bzw. Anwendung mit Datenbankzugriff zu schreiben. Man muß dabei nicht auf die Sicherheit und die Vorteile von Java verzichten, welche man verlieren würde, wenn man Datenbankschnittstellen benutzt, die in nativen Programmiersprachen, wie C oder C++, geschrieben sind.
Im Folgenden sind nur einige der Vorteile von JDBC aufgeführt:

- Es ist sehr einfach, SQL-Anfragen an eine Datenbank zu senden, und die Ergebnisse auch wieder zu empfangen. Es genügen beispielsweise nur 6 Zeilen Code, um alle Datensätze einer Tabelle auszugeben (s.Abbildung 5).
- Die Programmierschnittstelle ist völlig unabhängig von der verwendeten Datenbank. Es ist nicht nötig, ein extra Programm zu schreiben, um auf eine Oracle Datenbank zuzugreifen, und ein anderes, weil man eine Sybase Datenbank hat. Man muß nur ein Programm schreiben, und kann damit auf beliebige relationale Datenbanken zugreifen, die SQL92 kompatibel sind, und für die man einen JDBC-Treiber zur Verfügung hat (s. Abschnitt 4).
- JDBC hat auch, da es komplett in Java geschrieben ist, den Vorteil, daß ein Programm auf allen denkbaren Plattformen sicher und zuverlässig läuft. Dies ermöglicht beispielsweise in einem Intranet allen Benutzern, egal ob sie nun Windows-NT oder UNIX benutzen, mit dem *selben* Programm auf verschiedene Datenbanken zuzugreifen, ohne großen Aufwand betreiben zu müssen.
- JDBC ermöglicht auch, daß man im Internet über ein Applet auf eine Datenbank zugreift, ohne genau zu wissen, ob es nun eine Oracle oder eine Sybase Datenbank ist, und ohne spezielle Treiber zu installieren, um nur kurz Zugriff auf diese Datenbank zu erhalten.

9.3.1 Einsatzmöglichkeiten von JDBC

Bevor nun die JDBC-Datebankschnittstelle im Einzelnen vorgestellt wird, sollte man die typischen Einsatzvarianten von Java-Programmen und die daraus resultierenden Möglichkeiten des Einsatzes von JDBC untersuchen.

Java Applets - JDBC im Internet

Am meisten werden Java-Programme derzeit im Internet verwendet und zwar in Form von Applets, die auf einem Web-Browser ausgeführt werden. Diese werden über das Internet als Teil einer entsprechenden Web-Seite geladen. Es liegt daher nahe, nun auch JDBC zu benutzen, um über eine Web-Seite, genauer einem darin befindlichen Applet, auf eine

Datenbank, die sich irgendwo in der Welt befindet, zuzugreifen.

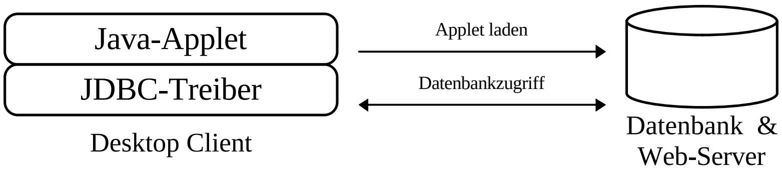


Abbildung 9.3: JDBC im Internet

Man kann sich vorstellen, daß beispielsweise ein großes Versandhaus seinen Katalog online zur Verfügung stellt, und daß man sich über die entsprechenden Web-Seiten die einzelnen Produkte anschauen kann und diese eventuell in eine Bestelliste einträgt. Zum Schluß läßt man sich dann noch einmal eine Liste aller Waren anzeigen, und bestellt sie per Knopfdruck. Diese Anwendung wäre ohne eine gute Datenbankschnittstelle nicht zu realisieren. Bei der Verwendung von CGIs wäre es zum Beispiel sehr umständlich eine Bestellung mittendrin abzubrechen, d.h. eine größere Transaktion zurückzusetzen.
Bei der Verwendung von Applets im Internet gibt es jedoch einige Sicherheitseinschränkungen, die auch für die Benutzung einer Datenbank Konsequenzen haben:

- Normale Applets, sog. *untrusted applets*, dürfen nicht auf das lokale Filesystem zugreifen, d.h. sie dürfen weder Daten lesen, noch Daten abspeichern. Außerdem dürfen sie keine Netzwerkverbindung zu einem anderen Rechner, als demjenigen, von dem das Applet geladen wurde, aufbauen. Das heißt, daß sich die Datenbank, auf welche man zugreifen will, auf dem selben Rechner wie das Applet befinden muß.

- Die Performance beim Zugriff auf eine Datenbank kann erheblich schwanken, da sich die Client-Rechner überall im Internet befinden können. Einmal ist der Benutzer über ein Hochgeschwindigkeitsnetz und ein anderes Mal über eine Modemleitung mit der Datenbank verbunden.

Es ist auch möglich, einem Applet dennoch zu erlauben die obigen Sicherheitsbestimmungen außer Kraft zu setzen. Dies kann man tun, falls man sich sicher ist, daß die verwendeten Ressourcen sicher sind. Man nennt dies dann ein *trusted applet*.

Java-Applications - JDBC im Intranet

Mit Java kann man auch ganz normale Anwendungen schreiben, die dann auf der Java Virtual Machine interpretiert werden, jedoch ist diese Ausführungsmethode etwa 10 mal langsamer als ein vergleichbares C++ Programm. Man kann sie auch aus Geschwindigkeitsgründen mit einem speziellen Compiler in den jeweiligen Maschinencode übersetzen. Diese Java- Anwendungen sind dann genau so schnell, wie ein entsprechendes C++ Programm.

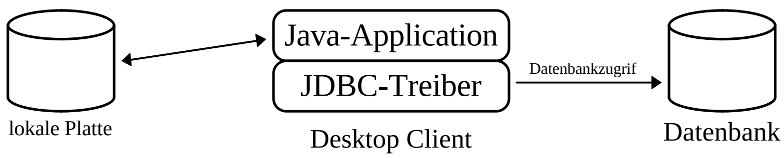


Abbildung 9.4: JDBC im Intranet

Diese Java Applikationen müssen sich auch nicht den strengen Sicherheitsvorschriften der Java-Applets beugen, da man bei ihnen davon ausgeht, daß diese Programme sicher sind. Java-Applications haben also uneingeschränkten Zugriff auf die lokale Platte, können zu beliebigen Rechnern eine Netzwerkverbindung öffnen und können auch beliebige Routinen anderer Programmiersprachen aufrufen.
Java-Applications sind vor allem für den Gebrauch in einem Intranet gedacht. Sie haben die Funktionalität wie andere Programme auch, und vielleicht sogar noch einige Vorzüge, wie beispielsweise Portabilität, gute Netzwerkunterstützung, ...
Und auch im Datenbankzugriff haben die Java-Applications einen entscheidenden Vorteil, falls sie JDBC benutzen. Man muß jetzt nur noch *ein* Programm schreiben, ohne zu berücksichtigen, welche Datenbank man verwendet, und kann dieses auch noch auf allen Plattformen verwenden.
Dies ist vielleicht die Lösung für Probleme, die viele Firmen haben. Ihre Rechneranlagen bestehen aus vielen verschiedenen Plattformen und Datenbanksystemen. Diese zu verbinden und Daten untereinander zugängig zu machen, verursachte bisher viele Probleme und einen Haufen Arbeit. Mit JDBC ist es nun möglich, alle Datenbanken auf allen unterstützen Plattformen anzusprechen und die Daten gemeinsam zu verarbeiten. Es können mehrere Datenbanken gleichzeitig bearbeitet werden, und auf diesen Datenbanken können auch gleichzeitig größere Transaktion ablaufen, die beliebig zurückgesetzt werden können.

9.3.2 Wie funktioniert JDBC ?

JDBC ist eigentlich sehr simpel, da es nur eine sehr einfache Schnittstelle für den Zugriff auf Datenbanken, eine sog. Low-Level API (Application Programmers Interface), zur Verfügung stellt. Die Entwickler von JDBC gehen davon aus, daß dieses Low-Level-API eine gute Basis für speziellere High-Level-APIs ist, beispielsweise eine graphische Benutzeroberfläche (GUI), bei der die Felder mit speziellen Anfragen verknüpft sind, o.ä. .
Den Zugriff auf eine SQL-Datenbank kann man in 3 Schritte einteilen:

- Aufbauen einer Verbindung mit der Datenbank
- Stellen einer SQL-Anfrage
- (tupelweises) Empfangen der Ergebnisse

Abbildung 5 zeigt ein Programmbeispiel mit einer einfachen Datenbankanfrage.
Wie in Abbildung 5 zu sehen, baut JDBC zuerst eine Verbindung mit der Datenbank auf. Dabei wird nicht das HTTP-Protokoll verwendet, welches vorher immer so große

```
Connection con = DriverManager.getConnection
                 ('jdbc:odbc:129.13.11.23:/TestDB','USER','PASSWORD');
Statement stm = con.CreateStatement();
ResultSet rs = stm.executeQuery('SELECT A,B,C FROM Table');
while (rs.next()) {
                    int x = getInt('A');
                    String s = getString('B');
                    float f = getFloat('C'); }
```

Abbildung 9.5: JDBC-Anfrage

Probleme bereitet hat. Sondern es wird, abhängig von der entsprechenden Datenbank und des verwendeten Schichtenmodells (s. Abschnitt 4) ein spezielles Protokoll verwendet, welches erlaubt, größere Transaktionen zu starten, und auch wieder abzubrechen.
Es können auch mehrere Verbindungen zu einer Datenbank, oder auch zu verschiedenen Datenbanken aufgebaut werden.
Wie man auch erkennen kann, gibt es bei JDBC die bisherigen Probleme des DB-Zugriffs im World Wide Web nicht mehr.
Man kann sich aber fragen, wieso man gerade JDBC verwenden soll, es gibt doch schon eine allgemeine Datenbankschnittstelle, nämlich ODBC (Open DataBase Connectivity) von Microsoft.
Die Frage ist berechtigt, jedoch gibt es einige Argumente, die gegen die Verwendung von ODBC im World Wide Web sprechen:

- ODBC ist nicht für die Verwendung mit Java entwickelt worden. Es ist in C geschrieben. Man kann es natürlich in Java verwenden, muß dann aber auf einige Sicherheitsmerkmale von Java verzichten (siehe 3.1.1. JDBC im Internet).

- Man kann nicht das ganze ODBC in Java verwenden, da in Java einige der Funktionalitäten von C/C++ ausgeklammert sind.

- ODBC ist viel komplizierter als JDBC. Selbst für einfache Anfragen muß man viel Progammierarbeit aufwenden, um zum Ziel zu gelangen.

- Wenn man ODBC benutzt, so muß man auf jeder Maschine den ODBC Driver-Manager installieren, und zudem sind ODBC-Treiber sehr groß (4-8 MB). Verwendet man jedoch JDBC kann man überall den selben Treiber installieren, da Java plattformunabhängig ist. Und außerdem ist JDBC standardmäßig in Java enthalten. Es entfällt also dessen Installation.

Da ODBC in der Datenbankwelt weit verbreitet ist, und es fuer die meisten Datenbanken einen ODBC-Treiber gibt, hat SunSoft eine JDBC-ODBC-Bridge entwickeln lassen, die es erlaubt, mit Java auf eine ODBC-Treiber gestützte Datenbank zuzugreifen, ohne dabei

auf den vollen Umfang von JDBC und die Vorteile von Java verzichten zu müssen. Dabei werden die Aufrufe von JDBC in Aufrufe von ODBC umgesetzt.
Mit dieser JDBC-ODBC-Bridge ist es möglich JDBC-Anwendungen für die meisten Datenbanken zu schreiben, obwohl es noch keine JDBC-Treiber gibt. Es ist aber nur eine vorläufige Lösung, bei der man dann später, ohne große Probleme, auf einen echten JDBC-Treiber umstellen kann.
Im folgenden werden die verschiedenen Treiber, und deren Einsatzmöglichkeiten erläutert.

9.4 JDBC-Treiber

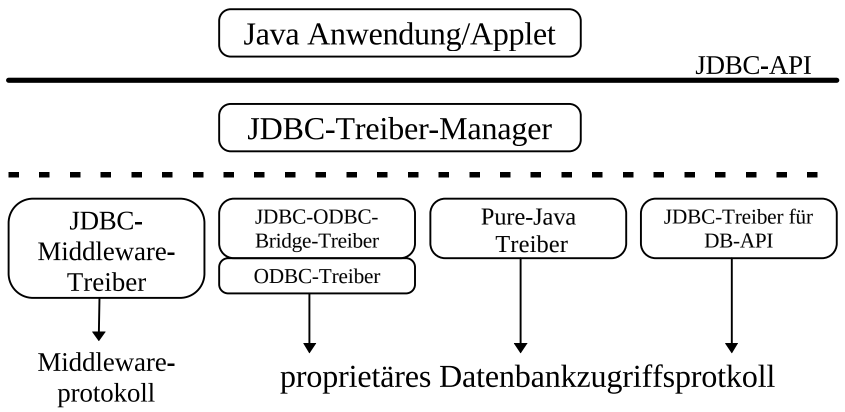
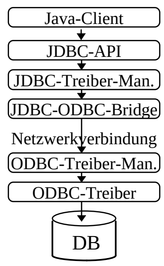


Abbildung 9.6: JDBC-Treiber-Architektur

Die JDBC-Datenbankschnittstelle unterstützt vier mögliche Arten von Treibern, über die man mit der Datenbank in Verbindung treten kann.
Möchte man auf eine Datenbank zugreifen, so teilt man dies dem Driver-Manager mit. Dieser versucht nun für die angegebene Datenbank einen passenden Treiber zu finden. Hat er einen gefunden, so tritt der Treibermanager in den Hintergrund, und der entsprechende Treiber baut eine Verbindung mit der Datenbank auf.
Man unterscheidet folgende vier Arten von Treibern (s. Abb. 4.1):

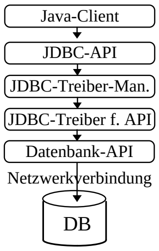
9.4.1 JDBC-ODBC-Bridge plus ODBC-Treiber

Diese Treiber bilden die JDBC-Statements auf die ODBC-Ebene ab. Es ist so möglich, über entsprechende ODBC-Treiber auf Datenbanken zuzugreifen. Diese Lösung ist meist eine pragmatische und nur zeitlich begrenzte Alternative, falls entsprechende JDBC-Treiber für diese Datenbank noch nicht existieren.
Da der ODBC-Treiber und der ODBC-Treibermanager in C geschrieben



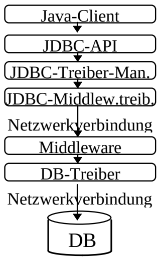
sind, kann diese Variante nicht für Applets benutzt werden (Stichwort *untrusted applets*). Außerdem ist diese Variante langsamer, als die anderen, da hier die Daten erst noch vom JDBC-Format in Java-Datentypen umgewandelt werden müssen.

9.4.2 JDBC-Treiber für DB-API auf Client-Seite



Bei dieser Variante wird vom Datenbankhersteller ein JDBC-Treiber für den Datenbanktreiber auf der Client-Seite zur Verfügung gestellt. Dieser Treiber setzt die JDBC-Befehle in Statements für das DB-API um, welches dann die Kommunikation mit der Datenbank, über ein proprietäres Protokoll, vornimmt.
Da die Datenbankschnittstelle auf der Client-Seite nicht in Java geschrieben ist, kommt es auch hier zu Sicherheitsproblemen bei der Ausführung von Applets (Stichwort *untrusted applets*), d.h. diese Variante ist nicht für den Einsatz im Internet geeignet.

9.4.3 JDBC-Middleware-Treiber

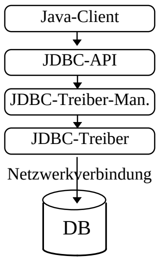


Bei dieser Treiber-Variante benutzt man ein sog. 3-Schichten-Modell. Es wird zwischen Client (Java-Anwendung) und Server (Datenbank) eine dritte Schicht (Middleware) eingebaut, die es ermöglichen soll, die Java-Anwendung unabhängig von der entsprechenden Datenbank zu machen. Dies ist auch ein häufig benutztes Modell, um auf mehrere Datenbanken zugreifen zu können. Die Middlewareschicht fungiert somit als ein sog. *Request Broker*, welcher die Datenbankanfragen (mehrerer) Clients sammelt und dann auf die verschieden Datenbanken (mit deren spezifischen Treibern) verteilt.
Beim Client wird somit nur ein Treiber installiert, der mit der Middleware über ein Datenbank-unabhängiges Protokoll kommuniziert. Die Middleware gibt dann die JDBC-Befehle an die entsprechenden Treiber weiter, die dann mit der Datenbank kommunizieren. Diese Architektur weist zusätzlich noch folgende Vorteile auf:

- Beim Austausch oder der Modifikation eines Datenbanktreibers findet die Aktualisierung nur auf der Middleware statt, und nicht auf allen Clients.
- Durch die bessere Skalierbarkeit kann der Anwender auch in großen Umgebungen geordnet auf verschiedene Datenbanken zugreifen.

Diese Variante eignet sich auch für Datenbankzugriffe aus dem World Wide Web, bei denen der Benutzer keine Kenntnis von der verwendeten Datenbank hat. Ein Beispiel für eine Anwendung eines Request Brokers kann in [Klu97] nachgelesen werden.

9.4.4 Pure-Java-Treiber



Dieser Treiber ist wie der Middleware-Treiber vollständig in Java entworfen und kommuniziert direkt mit der Datenbank über das, von der entsprechenden Datenbank abhängige, Netzwerkprotokoll. Dieser Treiber erlaubt direkten Zugriff von der Client-Anwendung auf die Datenbank und eignet sich vor allem für Intranetlösungen.
Da dieser Treiber zu 100% in Java implementiert ist, eignet er sich auch für die Verwendung im Internet, falls bekannt ist, auf welchen Typ von Datenbank man zugreift. Man kann das Java-Applet so organisieren, daß der Treiber beim Laden des Applets automatisch mit auf den Client-Rechner geladen wird. Somit kann die Unabhängigkeit von der verwendeten Datenbank erreicht werden.

9.5 Struktur von JDBC

Die JDBC-API besteht aus vier wichtigen Klassen (und einigen Unterklassen), mit denen man den Zugriff auf relationale Datenbanken erhält. Diese vier Klassen und deren Abhängigkeiten (bzgl. Aufruf) verdeutlicht Abbildung 7.

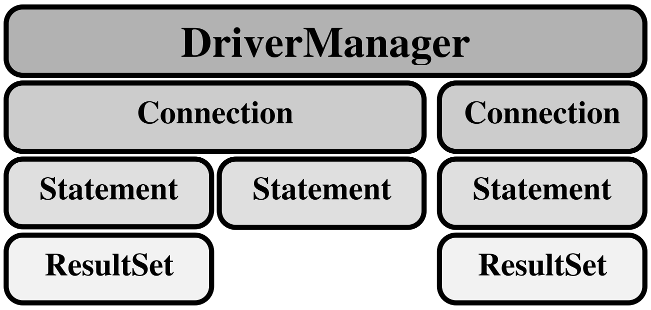


Abbildung 9.7: JDBC-Aufruf-Hierarchie

9.5.1 Der DriverManager

Der DriverManager ist der erste Anlaufpunkt für einen Zugriff auf eine Datenbank. Er verwaltet die Treiber, und sucht bei einer aufzubauenden Verbindung einen passenden Treiber aus. Dabei muß dem DriverManager mitgeteilt werden, welche Datenquelle angesprochen werden soll. Diese Information wird dem DriverManager, wie im WWW üblich, in Form eines speziellen Uniform Ressource Locators (URL) übergeben.

JDBC-Uniform Ressource Locator (URL)

Das Prinzip des URL ist aus dem World Wide Web hinreichend bekannt. Dieser spezifiziert nicht nur den Aufenthaltsort einer Information, sondern auch die Art und Weise (d.h. das

Protokoll), mit der man Zugriff auf diese Information erhält.

Der JDBC-URL ist folgendermaßen aufgebaut: `JDBC:<subprotocol>:<subname>`

- **JDBC** ist ein Schlüsselwort, welches kennzeichnet, daß es sich hier um eine JDBC-URL handelt, d.h. den Zugriffspfad zu einer Datenbank.
- **subprotocol** bezeichnet den Mechanismus mit dem der Treiber mit der Datenbank in Kontakt tritt. D.h. hier wird das verwendete (Sub-)Protokoll zwischen Datenbank und Treiber angegeben. Ein Beispiel für ein (Sub-)Protokoll ist ODBC. Bei einem Verbindungswunsch fragt der DriverManager alle verfügbaren Treiber, welche dieses spezielle Subprotokoll verstehen, und ob sie eine Verbindung mit der entsprechenden Datenquelle aufbauen können. Der erste Treiber, der die Verbindung aufgebaut hat, erhält den Zuschlag, d.h. über ihn wird der Datenbankzugriff durchgeführt.
- **subname** spezifiziert die genaue Datenquelle, d.h. beispielsweise den Namen und das Verzeichnis der Datenquelle. Befindet sich die Datenbank auf einem anderen Rechner, so muß man hier auch den Hostname (und event. den Port) dieses Rechners angeben, z.B.: `jdbc:db-test://db.sweet.home.de/oracle/db1`

Man kann zum subname auch noch zusätzlich Parameter übergeben. Diese sind jedoch abhängig vom verwendeten (Sub-)Protokoll.

Aufbau einer Verbindung

Möchte man nun eine Verbindung mit einer Datenbank aufbauen, so teilt man dies dem DriverManager durch die Methode *DriverManager.getConnection(URL, USER, PASS)* mit. Als Parameter übergibt man den URL der Datenquelle, eine Zugangskennung für die Datenbank (Username) und ein zugehöriges Paßwort.
Somit kann nun der DriverManager versuchen, eine Verbindung zur Datenbank herzustellen. Ist dies erfolgreich, und hat die angegebene Benutzerkennung Zugriffsrechte auf die Datenbank, so gibt die Methode *getConnection* ein Objekt vom Typ *Connection* zurück. Dieses Objekt repräsentiert nun die aufgebaute Verbindung zur Datenquelle. Über dieses Objekt laufen nun auch im Folgenden die Zugriffe auf die Daten.
Wie in Abbildung 7 erkennbar ist, kann man mehrere Verbindungen zu einer Datenbank aufbauen. In der Regel wird für jede Anwendung, welche Zugriff auf eine Datenquelle wünscht, genau eine Verbindung aufgebaut. Diese Anwendung kann natürlich noch zusätzliche Verbindungen zu anderen Datenquellen besitzen. Möchte man nun eine Anfrage an eine bestimmte Datenquelle stellen, so erfolgt dies über Objekte vom Typ *Statement* (oder Untertypen). Diese werden im Folgenden vorgestellt.

9.5.2 Statements

Statement-Objekte werden benutzt, um SQL-Anfragen an eine Datenbank zu stellen. Dabei unterscheidet man drei Arten von *Statement*-Objekten: **Statement**, **PreparedState-**

ment und **CallableStatement**, wobei PreparedStatement ein Untertyp von Statement und CallableStatement ein Untertyp von PreparedStatement ist.

Statement

Objekte vom Typ *Statement* werden benutzt, um einfache SQL-Anfragen zu stellen. Mit der Methode *connection.createStatement()* erzeugt man zuerst ein Objekt vom Typ *Statement*, und kann dann mit den verschiedenen Methoden dieser Klasse die Anfragen an die Datenquelle senden. Als Ergebnis erhält dann ein Objekt, welches die Resultate der Anfrage enthält (s.5.3 ResultSet).
Hat man nun ein Statement Objekt, so gibt es generell drei Möglichkeiten, die Anfrage zu stellen. Welche der drei Methoden man benutzen muß, ist abhängig davon, was die Anfrage als Ergebnis produziert.

- **ExecuteQuery(sql-string):** Im Normalfall hat man eine SELECT ... FROM ... WHERE Klausel, welche verschiedene Relationen miteinander verbindet, und als Ergebnis eine einzige Relation zurückgibt. Genau dann, bei einer Ergebnisrelation, benutzt man *executeQuery* und übergibt die SELECT-Klausel in einem String. Als Rückgabewert der Methode erhält man dann ein Objekt vom Typ *ResultSet* (s. 5.3), durch welches man auf die Ergebnisrelation zugreifen kann.

- **ExecuteUpdate(sql-string):** Mit dieser Methode werden Abfragen auf der Datenbank durchgeführt, die keine Ergebnisrelation erzeugen. Dies sind beispielsweise UPDATE, DELETE oder INSERT-Operationen. Wie bei *executeQuery(sql-string)* wird auch hier die SQL-Anfrage in Form eines String übergeben.
  Da diese Methode kein Ergebnis erzeugt, wird hier auch kein ResultSet zurückgegeben. Um jedoch einen Rückschluß auf die vollzogenen Änderungen der Anfrage zu erhalten, wird von dieser Methode die Anzahl der veränderten Tupel in Form eines Integer-Wertes zurückgegeben.

- **Execute(sql-string):** Diese Methode wird in solchen Fällen benutzt, in denen mehr als eine Ergebnisrelation erzeugt wird oder mehr als eine Relation geändert, erzeugt oder teilweise gelöscht wird. Um nun auf die verschiedenen Ergebnisrelationen und die verschiedenen Angaben über veränderte Zeilen zugreifen zu können, gibt es die Methoden *getResultSet* und *getUpdateCount* (ausführliche Behandlung von execute in [Lab97].

Objekte vom Typ *Statement* sind am besten für ad hoc SQL-Statements geeignet, die nur einmal ausgeführt werden. Wenn ein DBMS eine Anfrage vom Typ *Statement* erhält, so wird jedesmal die Syntax auf Korrektheit überprüft, vom Optimierer eine möglichst optimale Anfrage erzeugt, und ein Ablaufplan zur Ausführung wird generiert. Nach der Ausführung der Abfrage wird der Ausführungsplan und die optimierte Anfrage wieder gelöscht.
Diese Schritte werden auch ausgeführt, falls die gleiche Anfrage mehrere Male hintereinander gestellt wird. Nun gibt es jedoch Anfragen, die man öfters stellen möchte. Es wäre

dann ziemlich aufwendig, jedesmal die selben Schritte redundant auszuführen. Deshalb gibt es in JDBC die Klasse *PreparedStatement*, die eine Unterklasse von *Statement* ist.

PreparedStatement

Bei einem *PreparedStatement*-Objekt bereitet das Anwendungsprogramm, wie der Name schon sagt, ein SQL-Statement mit der Methode *connection.prepareStatement(sql-string)* vor. Die *preparedStatement*-Methode nimmt als Parameter einen String entgegen, der die gewünschte SQL-Anfrage enthält, und an das entsprechende DBMS weitergibt.
Das DBMS führt nun eine Syntaxüberprüfung und eine Optimierungsphase durch. Danach wird der optimierte Ablaufplan für die Ausführung der Anfrage erzeugt. Im Gegensatz zur Klasse *Statement* wird jedoch hier die Anfrage nicht sofort ausgeführt, sondern es wird lediglich ein Handle auf den optimierten Ausführungsplan zurückgegeben, welches im *PreparedStatement*-Objekt gespeichert wird.
Wie bei der Klasse *Statement* kann man nun eine Abfrage mit den Methoden *executeQuery*, *executeUpdate* und *execute* ausführen. Es ist jedoch zu beachten, daß diese Methoden keine Parameter haben, d.h. es wird nur das vorbereitete (optimierte) SQL-Statement ausgeführt.
Durch diese Art von Statements kann man sich viel Overhead bei der Ausführung sparen, falls vorher bekannt ist, daß die Anfrage oft gestellt wird. Diese Art von Anfrage eignet sich auch sehr gut für die Verwendung im Internet, weil beim Aufruf eines PreparedStatement nur noch ein Handle auf die Abfrage über das Netz geschickt wird, und nicht ein kompletter Abfrage- String. Es ist jedoch zu beachten, daß der optimierte Ausführungsplan eines *Prepared Statement* nur für die Dauer einer Verbindung (Connection) im DBMS gespeichert wird.
Weil die Anfrage aber nur einmal an die Datenbank gesendet wird, muß man die Parameter der Anfrage explizit vor jeder Ausführung angeben. Dies wird erreicht, indem man im SQL-String die Stellen, an denen Parameter stehen, mit einem Fragezeichen versieht. Dadurch weiß das DBMS, daß diese Parameter jeweils vor der Ausführung gesetzt werden. Um einen solchen Wert zu setzten gibt es die *setxxx(Index, Wert)* Methoden (hier steht xxx für Byte, Float, Long, etc.). Index gibt die Nummer des Parameters an, den man setzen will. Wert gibt den spezifischen Wert an, mit dem man den Parameter belegen will. Wird der Wert vor einer erneuten Anfrage nicht explizit gesetzt, so wird der Wert der letzten Anfrage übernommen. Im Beispiel 5.2. ist die Verwendung eines *PreparedStatement*-Objektes dargestellt.

CallableStatements

Viele DBMS erlauben den Zugriff auf sog. *Stored Procedures*. Bei diesen DBMS sind die Abfragen schon intern gespeichert und optimiert. Man muß deshalb auch nicht den Anfrage- String übergeben.
Um eine *stored procedure* aufrufen zu können, muß ein Objekt vom Typ *CallableStatement* erzeugt werden. Dies erledigt man mit der Methode *connection.prepareCall({ call Proce-*

```
Connection con = DriverManager.getConnection(URL,USER,PWD);
Statement preStm = con.prepareStatement
                    ('SELECT A,B,C FROM Table WHERE A = ?  AND B = ?');
preStm.setString(1,'WertA');
preStm.setFloat(2,0.1234);
ResultSet rs = stm.executeQuery();
```

Abbildung 9.8: Prepared Statement

durName[?, ?, ?] }) , wobei ProcedurName den Namen der stored procedure im DBMS angibt, und die Anzahl der Fragezeichen der Anzahl der Parameter entspricht.
Die Übergabe der Parameter und das Erhalten der Ergebnisse verhält sich genau so, wie bei den PreparedStatements (s. 5.2.2.).

9.5.3 ResultSets

Bisher wurde nur betrachtet, wie man eine Anfrage stellt. Doch im allgemeinen will man auch die Ergebnisse einer Anfrage erhalten. Dafür gibt es die Klasse *ResultSet*.
Ein Objekt der Klasse *ResultSet* erhält man (vgl. 5.2) als Rückgabewert der Methode *executeQuery()*. In diesem Fall gibt es genau eine Relation, in der die Ergebnisse abgelegt sind. Ein *ResultSet* besteht aus Datenzeilen (Tupeln), die in uniformen Spalten organisiert sind. Es ist aber jeweils nur ein Tupel der Relation sichtbar. Um die nächste Zeile zu erhalten, muß man die Methode *ResultSet.next()* aufrufen, die dann den Cursor (Zeiger auf die aktuelle Zeile) um eine Zeile weiter setzt. Falls es keine weitere Zeile mehr gibt, so wird *False* zurückgegeben.
Somit ist es möglich, die gesamte Relation tupelweise durchzugehen und deren Ergebnisse zu verarbeiten. Es gibt in JDBC leider keine andere Möglichkeit als die *next()* Methode, um zu bestimmten Zeilen zu gelangen.
Um nun auf die einzelnen Attribute der aktuellen Zeile zugreifen zu können, gibt es in JDBC die Methoden *getxxx()*, mit denen man den Wert einer Spalte einliest und in den Java-Datentyp xxx (z.B. Int, Time, etc.) konvertiert. Um nun die jeweilige Spalte zu spezifizieren, gibt es zwei Möglichkeiten:

1. Entweder, man gibt den Spaltennamen an, wie beispielsweise *resultSet.getFloat('A')*, oder

2. falls dieser nicht eindeutig ist, kann man auch den Positionsindex (1 für erste Spalte, 2 für zweite ...) der Spalte angeben, wie z.B. *resultSet.getFloat(i)*, das den Wert der i-ten Spalte der aktuellen Zeile einliest und in den Java-Datentyp *Float* konvertiert.

Dabei ist zu beachten, daß man den richtigen Java-Datentyp angibt. Es sollte beispielsweise der SQL-Datentyp *VARCHAR* in einen Java-Datentyp *String* umgewandelt werden, und ein *TIME* in den Java-Typ *Time*. Tabelle 5.3. zeigt die Default-Java-Datentypen, in die der jeweilige SQL-Datentyp konvertiert werden sollte. Diese Zuordnung ist aber keinesfalls zwingend, so kann beispielsweise jeder SQL-Datentyp in einen Java-String konvertiert werden. Die Entscheidung, in welchen Datentyp man konvertiert, hängt vom jeweiligen Einzelfall ab. Die Möglichkeiten der Zuordnung kann man in [Lab97-Kap.5.1.3] genau nachlesen.

SQL	**Java**	**SQL**	**Java**
CHAR	String	REAL	float
VARCHAR	String	FLOAT	double
NUMERIC	BigDecimal	DOUBLE	double
DECIMAL	BigDecimal	BINARY	byte[ ]
BIT	boolean	VARBINARY	byte[ ]
TINYINT	byte	DATE	Date
SMALLIN	short	TIME	Time
INTEGER	int	TIMESTAMP	Timestamp
BIGINT	long		

9.6 Zukunft von JDBC

JDBC zeigt vielversprechende Ansätze zur Beseitigung der Mängel der bisherigen Datenbankschnittstellen für das World Wide Web. Es erlaubt beliebig viele Verbindungen zu beliebigen relationalen Datenbanken, (große) Transaktionen und hat eine flexible Treiberarchitektur. Außerdem verfügt es mit Java über eine sehr gute und weitestgehend akzeptierte Programmiersprache.

Es liegen jedoch nur wenige praktische Erfahrungen mit dem JDBC-API vor. Die Praxis muß zeigen, ob Performance und Stabilität den Erfordernissen, besonders von heterogenen Umgebungen, gewachsen sind.

Vor allem hängt die Zukunft von JDBC auch von der Unterstützung der Datenbank-Hersteller ab. Falls diese für ihre Produkte in nächster Zeit JDBC-Treiber anbieten, wird sich JDBC immer mehr verbreiten.

Durch die große Ähnlichkeit zu ODBC wird es den meisten Datenbankherstellern nicht schwer fallen, auch eine JDBC-Schnittstelle zur Verfügung zu stellen. Auch den Anwendungsentwickler wird, durch das sehr einfache JDBC-API, der Einstieg in JDBC nicht schwer fallen.

Zusammenfassend lässt sich sagen, daß es mit JDBC nun möglich ist, die eingangs geforderten *stark interaktiven* Anwendungen für das World Wide Web, auf einfache Weise, zu entwickeln.

Kapitel 10

Umweltinformationssysteme im World Wide Web

Jürgen Enge

Kurzfassung *Gegenstand dieses Textes sind Umweltinformationssysteme. Dabei werden zwei Systeme exemplarisch vorgestellt, um die vorangestellte Analyse der Nutzung näher zu konkretisieren.*

10.1 Einleitung

Um möglichst vielen Nutzern den Zugriff auf Umweltinformationssysteme zu gewähren, werden immer mehr dieser Systeme in das World Wide Web eingebunden. Damit ist der Zugang von fast allen Rechnerplattformen und Fenstersystemen gewährleistet und es ist nicht mehr nötig, für jedes System eine eigene Client-Software zu verwenden.

Welche Umweltsystemen gibt es im World Wide Web und werden diese überhaupt genutzt? Mit diesen Fragen beschäftigt sich diese Arbeit. Es wird eine Statistik über die Nutzung des Informationsangebotes von Umweltinformationssystemen aufgestellt und es werden zwei existierende Systeme vorgestellt. Bei den beiden Umweltinformationssystemen handelt es sich um das UDK, einen europäischen Umweltdatenkatalog, und um GIS, ein Geographisches Informationssystem. ([LBM+96, MNE+96, KNK+96, Hab96])

10.2 Statistiken über die Nutzung und Nutzer von Umweltinformationssystemen

10.2.1 Motivation

Es hat sich gezeigt, dass der Zugang und die Benutzbarkeit von Geographischen Informationssystemen zu einem großen Problem bei einer steigenden Anzahl von Nutzern geworden ist. Aus diesem Grunde wurde eine Umfrage bei 78 Wissenschaftlern, Dozenten und Doktoranden aus akademischen Einrichtungen durchgeführt. Diese Personen, welche 28 Fachrichtungen widerspiegeln, sollten ihre Bedürfnisse bezüglich dem Zugriff auf Geographische Informationssysteme ausdrücken. Diese Daten stammen aus einem Paper mit dem Titel "Developing Internet-based user interfaces for improving spatial data access and usability", welches an der Universität von Manchester veröffentlicht wurde [Li, Bree, Moss, Petch]

10.2.2 Tatsächliche und potentielle Nutzung der Daten

Die Nutzung von geographischen Informationssystemen ist eine Aktivität, die viel Wissen voraussetzt, da die Analysekonzepte und -techniken sich von anderen Methoden, welche normalerweise von den Nutzern verwendet werden (Fotheringham und Rogerson, 1993), signifikant unterscheiden. Die Benutzer müssen sich mit den Befehlsstrukturen und Nomenklaturen von Geographischen Informationssystemen genauso wie mit den Systemen, auf denen sie sich befinden - häufig high-end Workstations - auseinandersetzen. Daher scheint die tatsächliche Nutzung von solchen Systemen relativ gering zu sein. Das Verhältnis zwischen der tatsächlichen und potentiellen Nutzung liegt hauptsächlich bei ca. 1:6. Nur bei staatlichen Stellen und bei Erhebungen liegt sie bei über 1:2.

10.2.3 Die technischen Fähigkeiten der Nutzer

Bei der zugrundliegenden Umfrage kamen lediglich 52% der Fragebögen komplett ausgefüllt zurück. Daraus wurden die Nutzer in vier Kategorien eingeteilt:

- Basic: Der Benutzer hat nur absolut notwendiges Basiswissen (17.15%)
- Intermediate: Der Benutzer hat mittelmäßiges Wissen (48.6%)
- Experienced: Der Benutzer ist erfahren (28.6%)
- Professionel: (5.7%)

Es zeigt sich, daß alle Benutzergruppen hauptsächlich WWW zur Recherche verw enden. In der Gruppe "Intermediate" wird jedoch auch noch News verwendet. Die anderen Dienste wie gopher und wais sind vernachlässigbar.

10.2.4 Benutzerschnittstellen

Die Benutzerschnittstelle ist eines der wichtigsten Merkmale von geographischen Informationssystemen. In der Umfrage wurden die Benutzer gebeten, die einzelnen Schnittstellentypen zu bewerten. Ausgehend von der Gesamtsumme der Punkte ergibt sich folgende Reihenfolge

1. Interaktive Dialogboxen
2. Baum-Menüs
3. Interaktive Karten
4. Mehrfachauswahlen
5. Natürlichsprachliche Schnittstellen

Betrachtet man jedoch nicht die Gesamtsumme der Punkte, sondern den Durchschnitt, so liegen die Interaktiven Karten vor den Baum-Menüs.

10.2.5 Gründe für die geringe Nutzung von geographischen Informationssystemen

Die Umfrage hat gezeigt, daß viele Befragte gar nicht von der Existenz dieser Systeme wußten, obwohl sie Nutzen aus ihnen ziehen könnten. Faktoren wie schlechte technische Unterstützung und wenig Hilfe erklären teilweise die geringe Nutzung dieser Systeme.

Es hat sich auch gezeigt, daß die einfache Nutzung der Systems, vor allem für Benutzer mit wenig technischem Wissen, noch wichtiger als die Funktionalität ist. Es wurde auch schon früher gezeigt, dass der unerfahrene Nutzer Probleme damit hat, vorhandenes Wissen in das Daten-Handling von Informationssystemen einfliessen zu lassen. Die Ergebnisse zeigen also, dass benutzerfreundliche und flexible Schnittstellen notwendig sind, um die Nutzbarkeit von geographischen Daten im Netz zu erhöhen.

10.3 WWW-UDK

10.3.1 Einführung

Der Umweltdatenkatalog UDK ist eine Meta-Informationssystem für Umweltdaten für staatliche Einrichtungen. Der ursprüngliche UDK bestand aus einer Datenbank und einer zugehörigen Applikation. Im FZI wurde die UDK-Datenbank im WWW verfügbar gemacht. WWW-UDK unterstützt verschiedene Abfragemodi für UDK-Objekte und Adressen, einen Umwelt-Thesaurus, Online-Zugang zu zugrundeliegenden Daten wie Datenbanken und Umweltberichten, mehrsprachige Abfrage- und Ergebnisformulare und ein Online-Hilfesystem. Im Moment wird WWW-UDK von vier Umweltministerien und staatlichen

Stellen für Umweltschutz in Österreich und Deutschland sowohl im Internet als auch im Intranet verwendet. WWW-UDK kann leicht in verteilte Architekturen, die auf CORBA, WWW und Java basieren, integriert werden.

Ende des Jahres 1994 war der erste prototypische Zugriff über das WWW auf die Daten des UDK möglich.

Es gibt drei Versionen des WWW-UDK, die für die öffentlichkeit zugänglich sind:

- WWW-UDK von Österreich (http://udk.bmu.gv.at)
- WWW-UDK des Landes Baden-Württemberg
- WWW-UDK des Landes Niedersachsen

WWW-UDK ist das Ergebnis eines 1994 am Forschungszentrum Informatik (FZI, Forschungsgruppe Datenbanksysteme) gestarten Forschungsauftrages des Umweltministeriums des Landes Baden-Württemberg. Ende des Jahres 1994 wurde der erste prototypische Online-Datenbankzugriff mit den Techniken und Werkzeugen des WWW auf die Datenbestände des Umweltdatenkatalogs (UDK), der auf einer Initiative des Landes Niedersachse basiert, entwickelt und realisiert. Im Laufe des Jahres 1995 wurde die Lösung im Auftrag des Umweltministeriums Baden-Württemberg erweitert sowie im Auftrag des Bundesministeriums für Umwelt, österreich, an spezifische Anforderungen, wie etwa mehrsprachige Masken, angepasst. Anfang 1996 fand dann die Umstellung auf das modifizierte Datenmodell von UDK 3.0 statt.

10.3.2 Inhalte

Der Umweltdatenkatalog enthält hauptsächlich Datenbezugsquellen. Bei der Suche nach Informationen kann man so z.B. eine Telefonnummer eines Amtes, das die gesuchten Daten besitzt, erhalten oder darüber informiert werden, welche staatliche Institution überhaupt für die Daten zuständig ist.

10.3.3 WWW-Schnittstelle

Die WWW-Schnittstelle erleichtert den Zugriff auf Umweltinformationen wesentlich und macht die Information dadurch vielen Anwendern zugänglich.

Dabei werden mehrere Arten von Diensten unterschieden:
Elementare Systemdienste bestehend aus

- Hilfsdienste, wie Verwaltungsdienste und Darstellungsdienste, welche die HTML-Seiten und Grafiken bereitstellen
- Datendienste, die den Datenbankzugriff implementieren. Dabei unterscheidet man

- Basisdienste, wie zum Beispiel der Kartendienst (GIS) oder Schlüssellisten mit den zugehörigen Datenbanken
- Auskunftsdienste, wie der UDK

Anwenderdienste bauen darauf auf. Dies sind die Dienste, die für den Anwender sichtbar sind. Sie steuern die Abfolge von einem oder mehreren Systemdiensten. Zu ihnen gehören der UDK-Dienst, die Kartendienste etc. Alle diese Anwenderdienste können durch den WWW-Server angesprochen werden.

Durch diese Unterscheidung wird die modulare Architektur der Anwenderdienste sichtbar.

Die Auskunftsdienste orientieren sich am Begriffsnetz des Technosphäre- und Luftinformationssystems (TULIS) und an den Datenstrukturen des UDK. Jeder Anwenderdienst erhält eine HTML-Startseite, über die er aufgerufen werden kann. (siehe Abbildung 1: Startseite für einen Dienst.)

10.3.4 Struktur und Implementierung

Das UDK-Objektmodell

Der UDK basiert auf einem Drei-Wege Objektmodell, in dem zwischen Umweltobjekten, Umweltdatenobjekten und UDK (Meta) Objekten unterschieden wird. Dabei sind unter den Umweltobjekten die Objekte der realen Welt zu verstehen. Das schließt sowohl natürliche Dinge, wie Seen und Biotope, als auch von Menschenhand geschaffene Dinge, wie Strassen und Fabriken ein. Jedes dieser Umweltobjekte wird durch eine Menge von Umweltdatenobjekten beschrieben. Diese Beschreibung kann aus Vermessungsdaten, Bildern oder anderen Bewertungsdaten bestehen.

Ein Umweltdatenobjekt muss eines der folgenden beiden Kriterien erfüllen:

1. Environmental Context: Das Datenobjekt enthält Informationen, die auf andere (reale) Umweltdaten verweisen. Hier sind also Referenzen abgespeichert
2. Data Capture: Hier befinden sich Informationen in Form von Text, Klang oder Bildern. Die Daten können auf folgende Art und Weise angelegt sein:
   - beschreibend
   - empirisch
   - durch Sammlung und Zusammenführung von anderen Daten
   - durch Simulation oder andere Modellierungstechniken
   - durch bewertende Daten

Jedes dieser Umweltdatenobjekte ist nun mit genau einem UDK Metadatenobjekt verknüpft, das das Datenformat und den Inhalt beschreibt.

Objekt-Klassen und Ableitungen

Man kann im Moment sieben verschiedene Klassen unterscheiden:

1. Projektdaten (Bauprojekte, Studien über Umwelteinwirkungen, etc.)
2. empirische Daten (Meßserien, Labordaten, etc.)
3. Daten über Anlagen (Fabriken, Gebäude, etc.)
4. Karten
5. Expertisen und Reports
6. Produktdaten
7. Modelldaten (Simulationen etc.)

Für jede dieser Klassen existiert eine UDK-Klasse, die das beschreibende UDK-Objekt enthält. Ziel ist es nun, diese horizontale Struktur in eine Vertikale einzubetten. Die Hierarchie soll dann auf folgende Art und Weise strukturiert werden:

- Die Wurzel der Hierarchie (Ebene 0) besteht aus der generischen Klasse UDK-Objekt mit vier obligatorischen Attributen: die eindeutige Objektidentifizierungsnummer (OID), dem Objektnamen, dem Datum der letzten Änderung und der für das Objekt zuständigen Einrichtung (oder Person).
- Ebene 1 enthält die relativ kleine Anzahl von Klassen, die Gemeinsamkeiten zwischen allen UDK-Teilnehmern repräsentieren. Dies sind im Moment die sieben oben genannten Klassen.
- Die folgenden Ebenen können die Klassen enthalten, die die teilnehmenden Länder und Institutionen benötigen, um speziellen Anforderungen entsprechen zu können.

Klassenattribute werden in dieser Hierarchie in objektorientierter Manier weiter vererbt. Dies schließt die Möglichkeit ein, Attribute, die optional waren, zu notwendig hochzustufen.

Semantische Beziehungen zwischen UDK-Objekten

Orthogonal zu der gerade beschriebenen Klassenhierarchie bietet der UDK noch die Möglichkeit, konkrete UDK-Objekte in hypertextähnlicher Form zu verknüpfen. Die entstehende Datenstruktur ist ein gerichteter Graph, dessen Knoten UDK-Objekte sind und dessen Kanten semantische Verbindungen zwischen ihnen repräsentieren. Die Semantik der Kanten kann verschieden sein.

Der wichtigste dieser Graphen ist der Primärbaum oder Primärkatalog. Jedes UDK-Objekt entspricht genau einem Knoten dieser Baumstruktur, das heißt, es existiert ein

Isomorphismus zwischen den Primärbaumknoten und den UDK-Objekten. Die Semantik der Knoten des Primärbaumes variiert zwischen "ist ver antwortlich für" und "ist eine Vereinigung von".

Abhängig von der Notwendigkeit kann es auch Sekundärkataloge geben, die dann eine andere semantische Repräsentation besitzen. Anders als beim Primärkatalog muß der Sekundärkatalog keine Baumstruktur besitzen. Man kann also ein UDK-Objekt mit einer beliebigen Anzahl von Sekundärkatalogen referenzieren.

Eine typische Anwendung für einen Sekundärkatalog ist die Repräsentation von zusätzlichen Zusammenfassungen, die nicht im Primärkatalog zu finden sind. Diese Art von Links sind sinnvoll, um Benutzer zuerst zu Zusammenfassungen von interessanten Daten und erst dann zu den Details zu führen. Eine andere Anwendung ist die Möglichkeit, mit Hilfe von Sekundärkatalogen persönliche Assoziationen abzubilden. Für diese freien Strukturen gibt es keinerlei Restriktionen. Genau wie die Freiheit bei Hyperlinks im World Wide Web ist jede Art von gerichtetem Graphen erlaubt, selbst zyklische Graphen.

Visualisierung

Die Visualisierung von Datensätzen ist ein wichtiger Punkt in allen Informationssystemen. Die Ansprüche an die Visualisierungstools steigen mit der Komplexität der Daten. Im UDK ist es absolut notwendig, daß sowohl tabularische, als auch komplexe geometrische Daten behandelt werden können.

Alle Tools, die im UDK verwendet werden, befinden sich auf Seite des WWW-Servers und sind public domain. Auf der Client Seite wird nur ein normaler WWW-Client benötigt.

Um Karten aus dem GIS darzustellen und um komplexe GIS-Funktionalität anzubieten wird das public domain Werkzeug GRASS verwendet. GRASS ist in C geschrieben und auf verschiedenen Unix-Plattformen lauffähig.

Die Anzeige tabularischer Daten benötigt ein Werkzeug, das in der Lage ist, Geschäftsgrafiken zu erzeugen. Im Moment wird GNUPLOT dazu verwendet Linien-, Balken- und Kuchengrafiken zu generieren, obwohl es dafür nicht sonderlich geeignet ist. Es gibt jedoch kein public domain Werkzeug, das es besser könnte.

Zusätzlich wird noch ein Werkzeug benötigt, das in der Lage ist, die Ausgabe von verschiedenen Rasterdatenquellen, wie Karten, Diagramme und Sinnbilder zu verbinden und zwischen verschiedenen Rastergrafikformaten, wie GIF, PPM und TIFF zu konvertieren. Dazu wird Tom Boutell's gd-library für Operationen mit GIF-Bildern und das FLY-tool von Martin Gleeson, welches ein Kommandozeilenwerkzeug für die gd-library ist, verwendet.

Implementierung

Die Programme für den Datenbankzugriff wurden unter UN*X in der Programmiersprache C mit embedded SQL implementiert. Die UDK-Daten liegen in dem relationalen Datenbanksystem ORACLE vor.

Das Word Wide Web ermöglicht den Zugriff auf verteilte Informationsquellen. Die reine WWW-Technologie läßt es aber nicht in ausreichender Art und Weise zu, verteilte Informationsquellen auf einer HTML-Seite zu kombinieren. Daher wurde eine Mittelschicht eingeführt, welche für die Verbindung zwischen verschiedenen Datenquellen zuständig ist. Ein Beispiel für solch eine Verbindung ist, wenn Daten aus dem UDK in Östereich mit Daten aus Deutschland zusammen auf einer Seite dargestellt werden sollen.

Zur Realisierung dieser Schicht wurde die Common Object Request Broker Architektur (CORBA) verwendet.

CORBA

Die Common Object Request Broker Architektur ist eine Zwischenschicht, die die Verknüpfung von verteilten Ojekt-Providern (Servern) und die Benutzung dieser Server durch Clients unterstützt. CORBA wurde von der Object Management Group (OMG), welche inzwischen mehr als 570 Mitglieder zählt, definiert. In dieser Architektur, welche hardwareunabhängig ist, ist der sogenannte Object Request Broker (ORB) verantwortlich für die Verteilung der Objektaufrufe der Klienten und den Antworten der Server auf diese Aufrufe. Wichtige Teile der CORBA-Definition sind ein Kern-Objektmodell, örtliche Transparenz (Clients und Server können sich auf beliebigen Maschinen befinden) und eine Programmiersprachenunabhängigkeit, die durch das Anbieten einer spezifischen Schnittstellendefinitionssprache (IDL) und einem "dynamic invocation interface" (DII) von Objekten.

IDL wird auf programmiersprachenspezifische Eigenschaften abgebildet, z.B. C und C++.

10.3.5 WWW-basierter Zugriff auf den UDK

In Abbildung 10.1 wird die HTML-Seitenabfolge des WWW-basierten UDK-Zugriffs dargestellt. Bei der Suche nach Objekten ist es grundsätzlich möglich in allen Katalogen zu suchen, was das Auffinden von Objekten bei unbekannter Katalogzuordung erleichtert.

Die Unterschiede zwischen den Sucharten auf Objekten liegt in der Komplexität der Suchanfrage und in den Attributen, auf denen gesucht wird. Das gesamte UDK-Zugriffsprogramm ist in mehrere Anwendungsdienste zerlegt. Jede dynamische HTML-Seite aus Abbildung 3 entspricht dabei der Ausgabe eines Dienstes. Dieser modulare Aufbau hat den Vorteil, daß die einzelnen Module auch in anderen Programmumgebungen aufgerufen werden können.

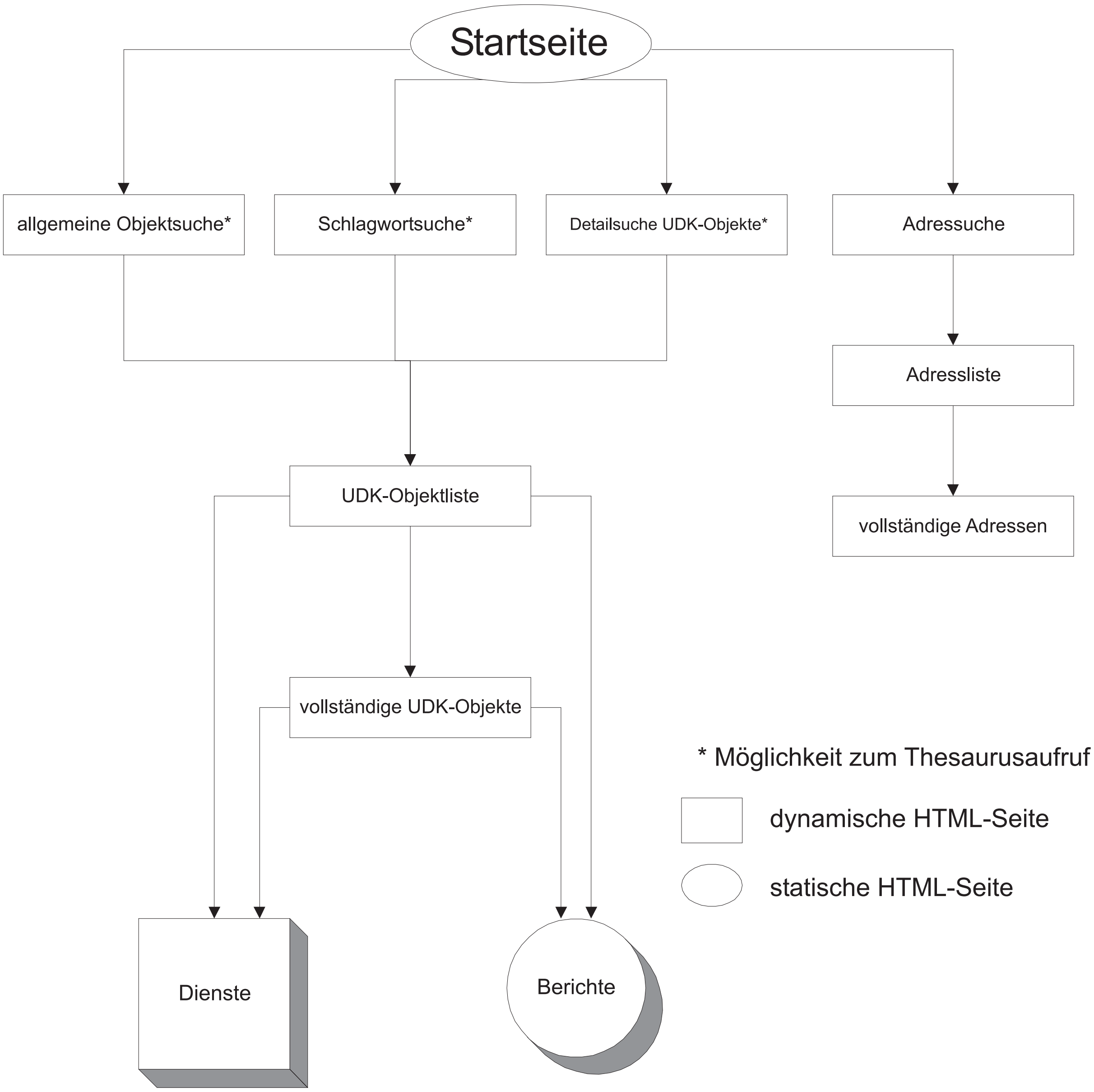


Abbildung 10.1: Die Abfolge der HTML-Seiten

So ist insbesondere auch der verteilte Aufruf von Diensten, z. B. unter CORBA, möglich.

Es fällt dabei auf, daß neben der Startseite nur noch Berichte aus statischen HTML-Seiten bestehen. Der Grund dafür besteht darin, daß die Seiten aus den Ergebnissen von Datenbankabfragen zusammengesetzt werden.

10.3.6 Beispiel Schlagwortsuche

Eine Möglichkeit des Suchens im WWW-UDK ist die Schlagwortsuche. Dabei wird in ein Formular das Schlagwort eingetragen und kann dann direkt zur Suche verwendet oder an einen Thesaurus gesendet werden. (siehe Abbildung 1: Einstiegsseite Schlagwortsuche)

Als Antwort auf die Schlagwortsuche erhält man eine Liste mit gefundenen Objekten, die nach UDK-Klassen (hier empirische Daten und Karten) sortiert sind. (siehe Abbildung 2: Objekttabelle)

Wählt man nun ein Objekt aus, so erhält man Informationen über dasselbige, in der eine Beschreibung der Daten und eine Adresse, unter der man Informationen erhalten kann, enthalten ist. (siehe Abbildung 3: Objekt und Abbildung 4: Adresse)

In der aktuellen Version sind im UDK kaum reale Daten zu finden, sondern hauptsächlich Referenzen auf Daten.

10.4 GIS/WWW

Geographisches Informationssystem im World Wide Web

GIS/WWW ist Teil eines Informationssystemes das dem Umweltschutz in West Virginia dient. Die Website wird von der "West Virginia Division of Environmental Protection" unterhalten. In diesem System sind verschiedene Arten von Umweltinformationen, wie Schürfgebiete, Luftqualität, Öl und Gasförderung, Wasserquellen etc. zusammengefaßt.

10.4.1 GRASS im WWW

Grass ist hauptsächlich ein Bitmap-orientiertes System, das ursprünglich vom U.S. Army-Construction Engineering Research Laboratory (USA-CERL) entwickelt wurde. GRASS Datensätze, die aus Lage, Vektor und Rasterdaten bestehen sind nach Orten angeordnet. Jeder Ort kann mehrere Kartensätz besitzen. GRASS besitzt verschiedene Vorteile: Offenen Dateiformate, frei verfügbarere Sourcecode und keine lizenzrechtlichen Einschränkungen. Mit GRASS Lite existiert ein grafisches Interface zu GRASS, das auf Tcl/Tk basiert und unter X-Windows lauffähig ist. Die CGI-Schnittstelle zu GRASS besteht nun aus drei Dokumenten, wobei der Benutzer im ersten Dokument (ein HTML-Form) seine DISPLAY-Variable angeben muss (freischalten des Display für den GRASS-Server muss auch noch erledigt werden). Im zweiten Dokument kann dann der gewünschte GRASS-Datensatz ausgewählt werden. Im letzten Dokument wird dann die GRASS-GUI auf dem

Server gestartet und die Bildschirmausgabe auf das Display des Benutzers umgelenkt.

Der Nachteil dieser Art der Datenausgabe besteht darin, daß mit jedem Nutzer ein Serverprozeß gestartet wird, der mit einem grafischen X-Window Interface zum einen eine Hohe Netzlast erzeugt und dabei noch sehr viele Resourcen des Servers beansprucht.

10.4.2 TAGIS

Das "Technical Application and Graphical Information System" (TAGIS) bietet eine Vielzahl von geographischen und Umweltdaten.

Folgende Daten werden in TAGIS angeboten:

TRI (Toxic Release Inventory Interface): Schadstoffausstoß

TAGIS/WVDEP entwickelte verschiedene Online-Schnittstellen zu TRI. HTML Schnittstellen erlauben die Datensuche in TRI nach Firmennamen, Chemikalienname, dem Stadtnamen und nach industriellen Abwässern in Flüsse von West Virginia. Als Antwort erhält man den Namen der Chemikalie, die Menge des Ausstoßes und das Jahr.

Beispiele

Um in TRI den Schadstoffausstoß einer bestimmten Firma einzusehen, wählt man einfach die gesuchte Firma aus der Liste mit den in der Datenbank enthaltenen Firmen aus (siehe Abbildung 6: Firmenauswahl).

In der nun erscheinenden Ansicht sieht man in tabularischer Form den Gesamtschadstoffausstoß der Firma über mehrere Jahre und den Ausstoß nach Chemikalien aufgeschlüsselt (siehe Abbildung 7: Schadstoffausstoß von 3M).

Ein weiterer Einstiegspunkt in TRI ist eine als clickable Image dargestellte Landkarte, die nach Flußgebieten aufgeteilt ist (siehe Abbildung 8: Clickable Image von West Virginia).

Nach Auswahl eines Flußgebietes wird eine Karte dieses Gebietes angezeigt, in der Firmen eingezeichnet sind, die Schadstoffe in dieses Gebiet ausstoßen (siehe Abbildung 9: Anzeige einer Landkarte des gewählten Bereiches mit Firmen, die Abwässer produzieren).

Ferner gibt es auch noch die Möglichkeit, sich eine generierte Karte anzeigen zu lassen, die in vielfacher Art und Weise manipuliert werden kann (siehe Abbildung 10: generierte Karte mit Schadstoffausstoß).

Interaktive Landkarten

Eine weitere Möglichkeit, sich TAGIS-Daten anzeigen zu lassen, sind die interaktiven Landkarten. Nach einer Auswahl eines Kartentyps aus den vier Möglichkeiten

- gesamter Staat
- Stromgebiete
- Länder
- Quadrangles (Einteilung in viereckige Abschnitte)

Die resultierende Karte wird dann auf dem Server aus verschiedenen Datenquellen generiert und erlaubt verschiedene Manipulationen wie Skalierung, die Anzeige von Flüssen, Stromgebieten und Landesgrenzen etc (siehe Abbildung 11: Karte).

Online-Suchsystem nach Namen

Im Suchsystem nach Namen kann nach folgenden Objekten gesucht werden

- Städte
- Schulen
- Kirchen
- Flüsse
- Flughäfen
- Friedhöfe
- Krankenhäuser
- Dämme

Hat man die Abfrage nach einem Name abgeschickt, so erhält man als Ergebnis eine Liste mit gefundenen Gebieten, aus denen man eines auswählen kann. Nach der Auswahl erhält man eine Landkarte, in der man, ähnlich wie in der vorigen Karte, auch verschiedene Objekte wie Autobahnen, Straßen, Flüsse, Landesgrenzen etc. ein- und ausblenden kann (siehe Abbildung 12:Landkarte).

Java-Interface zu den TAGIS-Karten

Ein Highlight der TAGIS-Landkarten stellt meiner Ansicht nach das Java-Interface zu den Karten dar. Hier ist es möglich interaktiv an den Karten zu manipulieren. Die Funktionalität entspricht weitgehend den schon zuvor vorgestellten Karten, das Java-Interface hat aber den enormen Vorteil, daß direkt auf Mausbewegungen und Tasten reagiert werden kann. Es ist auch die Selektion von beliebigen viereckigen und runden Gebieten möglich, was bei clickable Images und anderen WWW-Formularen technisch nicht möglich ist (siehe Abbildung 13: Java Interface.

Literaturverzeichnis

[ABI97] S. Abiteboul. Querying Semi-Structured Data, Database Theory, ICDT '97, Delphi, Proceedings, Springer Verlag

[AC93] Bernard Aboba, Vinton Cerf. *The Birth of the Arpanet* in *The Online User's Encyclopedia*, Addison Wesley.1993.

[ADM96] S.Abiteboul, J.McHugh, J.Widom, J. Wiener. The Lorel Query Language for Semistructured Data, Stanford University, http://www-db.stanford.de

[AQM96] S.Abiteboul, D.Quass, J.McHugh, J.Widom, and J.Wiener. The lorel query language for semistructured data, 1996, ftp://db.stanford.edu/pub/papers/lorel96.ps

[AS97] AltaVista-Suchmaschine. http://www.altavista.com. Internet, 1997.

[BCC+94] E. W. Brown, J. P. Callan, W. B. Croft, and J. E. B. Moss. Supporting full-text information retrieval with a persistent object store. In *EDBT '94*, 1994.

[Bla88] D. C. Blair. An extended relational document retrieval model. *Inf. Process. Mgmnt.*, 24(3):349–371, 1988.

[BN96] Sven Behrens, Ralf Nikolai. *Interaktive und dynamische WWW-Anwendungen mit Java.* Technical Report No. 3/96, Forschungszentrum Informatik (FZI) Karlsruhe, März 1996

[CDN+95] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, K. J. Worrell. *A Hierarchical Internet Object Cache.* Technical Report 95-611, Computer Science Department, University of California, Los Angeles, March 1995.

[CGI95] *The CGI Specification, Version 1.1* University of Illinois, http://hoohoo.ncsa.uiuc.edu/cgi/interface.html, 1996

[CHH+94] B. Chabra, D. Hardy, A. Hundhausen, D. Merkel, J. Noble, M. F. Schwartz *Integrating Complex Data Access Methods into the Mosaic/WWW Environment.* In: Proc. of the 2nd Int. World Wide Web Conference, Chicago, Illinois, pp. 909-919, October 1994

[CK+97] Vinton Cerf, Robert E. Kahn et al. *The Past and Future History of the Internet.* "Communications of the ACM". Vol. 40, No. 2, February 1997. *Und unter:* http://www.isoc.org/internet-history.

[ES97] Excite-Suchmaschine. http://www.exite.com. Internet, 1997.

[FKK96] Freier, A.O., Karlton, P., Kocher, P.C. *The SSL Protocol, Version 3.0 - Internet Draft.* Netscape Communications, http://home.netscape.com/eng/ssl3/ssl-toc.html, Mai 1996

[Gos96] Gosling, J., Joy, B., Steele, G. *The Java Language Specification.* Addison Wesley; http://java.sun.com:80/doc/ language_specification.html, 1996

[Hab96] Corinna Habeck: *Erweiterung und Modularisierung des Datendienstes Umweltdatenkatalog (UDK) in einem WWW-basierten Umweltinformationssystem.* Studienarbeit am Forschungszentrum Informatik, Universität Karlsruhe, 1996.

[Hal88] F. G. Halasz. Reflections on notecards: Seven issues for the next generation of hypermedia systems. Comm. of the ACM, 1988.

[Han97] Hansen,W.R. *Componentware mit DCOM/ActiveX* OBJEKTspektrum 3/97, Sigs Publications, 1997

[Har] The Harvest Project Group *Technical Discussion of the Harvest System.* http://harvest.transarc.com/afs/transarc.com/public/trg/Harvest

[HC96] Hamilton, G., Cattell, R. *JDBC - A Java SQL API, Version 1.10.* ftp://splash.javasoft.com/pub/jdbc-spec-0100.ps, Oktober 1996

[HC97] Hotbot-Crawler. http://www.hotbot.com. Internet, 1997.

[HN97] Haffter, P., Nagler, C. *CORBA, Java und das Web.* OBJEKTspektrum 2/97, Sigs Publications, 1997

[IBM1] IBM: *Visual Age und Construction from Parts.* http://www.software.ibm.com/software/ad/ vastub.html

[IBM2] IBM: *Web Connection.* http://www.software.ibm.com/software/ad/ visage/vawebwp.html

[IBM3] IBM: *CICS & IMS Connection.*

[Ion96] IONA Technologies Ltd. *Orbix, distributed object technology, White Paper.* http://www.iona.com/Orbix/Java/index.html, Februar 1996

[Jav96a] Sun Microsystems *White Paper: The Java Electronic Commerce Framework.* http://java.sun.com/products/commerce/doc.white_paper.html, 1996

[Jav96b] Sun Microsystems *Java Beans: A Component Architecture for Java* http://splash.javasoft.com/beans/WhitePaper.html, 1996

[Joh97] Johann, M. *Aus alt mach neu: Active X - Evolution einer Komponenten-Technologie* Software-Entwicklung 4/97, AWi-Verlag 1997

[Kah96] Robert E. Kahn. *The Role of Government in the Evolution of the Internet* as published in the internet.1996.

[KKS92] M.Kifer, W.Kim, and Y. Sagiv. Quering object-oriented databases. In SIGMOD, 1992

[Klu97] Rainer Klute: *Dateneintopf.* iX Multiuser Multitasking Magazin, (6):126-132, Juni 1997. `http://www.heise.de/iX/`, `http://www.ibm.de./go/ide/products/ims/imshome.html`

[KNK+96] R. Kramer, R. Nikolai, A. Koschel, C. Rolker, A. Keitel, R. Legat und K. Zwirn: *WWW-UDK: A Web-based Environmental Metainformation System* In Proc. 10th Intl. Symposium Umweltinformatik'96, Hannover, Germany, 1996.

[Kri97] Dave Kristula. *The History of the Internet.*1997.

[Lab96] Sun Mircosystems Laboratories. it The JDBC Database Access API. `http://splash.javasoft.com/jdbc/jdbc/databases.html`, 1996.

[Lab97] Sun Microsystems Laboratories. *The JDBC Guide - Getting Started.* `http://splash.javasoft.com/jdbc/`, 1997.

[LBM+96] Chun Sheng Li, David Bree, Adrian Moss, and James Petch: *Developing Internet-based user interfaces for improving spatial data access and usability.* Third Intl. Conf./Workshop Integrating GIS and Environmental Modeling, Santa Fe, New Mexico, USA, January 1996. http://ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM/sf_papers.html.

[Let96] Letovsky, S.I.: *Web-Database Gateways.* `http://gdbdoc.gdb.org/letovsky/genera/dbgw.html`, 1996

[Loe97] Loeser, Henrik *Datenbankanbindung an das WWW.* In 'Datenbanksysteme in Büro, Technik und Wissenschaft', GI-Fachtagung, Springer, Berlin, 1997

[MAG96] J.McHugh, S.Abiteboul, R.Goldman, D. Quass, J.Widom. Lore: A Database Management System for Semistructured Data, Stanford University, http://www-db.stanford.edu/lore

[Mih96] G. A. Mihaila. Websql- an sql-like query language for the world wide web. Diplomarbeit, Universität von Toronto, 1996.

[MNE+96] J.D. McCauley, K.C.S. Navulur, B.A. Engel, and R. Srinivasan: *Serving GIS Data Through the World Wide Web* Third Intl. Conf./Workshop Integrating GIS and Environmental Modeling, Santa Fe, New Mexico, USA, January 1996. http://ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM/sf_papers.html.

[MSG97] Udi Manber, Mike Smith, and Burra Gopal. WebGlimpse — Combining Browsing and Searching. In USENIX Association, editor, *To appear in the USENIX Annual Technical Conference, Jan 6-10*, Anaheim, CA, USA, 1997. USENIX.

[MW94] Udi Manber and Sun Wu. GLIMPSE: A tool to search through entire file systems. In USENIX Association, editor, *Proceedings of the Winter 1994 USENIX Conference: January 17–21, 1994, San Francisco, California, USA*, pages 23–32, Berkeley, CA, USA, Winter 1994. USENIX.

[New97] Alexander Newman, et.al. *Java - Special Edition - Referenz.* (34)-(36) 773-833, Que Verlag 1997

[OB97] O'Brien, T., Heise, D. *Java Beans oder Active X - was ist die geeignete Komponentenarchitektur für das WWW ?* OBJEKTspektrum 3/97, Sigs Publications, 1997

[OMG97] Object Management Group: *IIOP Specification.* `http://www.omg.org/corbask.htm`, 1997

[PGW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Data Engineering*, Taipei, Taiwan, 1995

[Pie97a] Piemont, C. *Components von der Insel - Java Beans als ActiveX-Konkurrenz ?* Software-Entwicklung 4/97, AWi-Verlag, 1997

[Pie97b] Piemont, C. *Ein Kännchen SQL.* Software-Entwicklung 5/97, AWi-Verlag, 1997

[QC94] John S. Quarterman, Smoot Carl-Mittchel *What is the Internet, Anyway?* in Matrix News, 4(8).1994

[RS95] Rescorla, E., Schiffman, A. *The Secure Hypertext Transfer Protocol - Internet Draft* Enterprise Integration Technologies, http://www.eit.com/projects/s-http/draft-ietf-wts-shttp-00.txt, Juli 1995

[Sal89] Gerard Salton. *Automatic Text Processing*, chapter 8–10, pages 227–365. Addison-Wesley, 1989.

[SC97] *Schwerpunktthema Componentware* OBJEKTspektrum 3/97, Sigs Publications 1997

[Sel97] Sellentin, J., Mitschang, B. *Möglichkeiten und Grenzen des Einsatzes von CORBA in DB-basierten Client-/Server-Anwendungssystemen.* In 'Datenbanksysteme in Büro, Technik und Wissenschaft', GI-Fachtagung, Springer, Berlin, 1997

[SG97] Schulz, N., Grund, M. *Vom Host ins Web - die deutsche Börse geht ans Netz.* OBJEKTspektrum 3/97, Sigs Publications, 1997

[TZ97] Jörg Turner, Stefan Zorn. *Heiße Tasse - Datenbankzugriff mit Java.* iX Multiuser Multitasking Magazin, (3):120-124, März 1997. `http://www.heise.de/iX/`

[VM96] Visa Mastercard. *Secure Electronic Transaction (SET) Specification - Book 3* http://www.visa.com/cgi-bin/vee/sf/set/setprot.html?2+0, August 1996

[WH97a] W3QL-Homepage. http://www.cs.technion.ac.il/~w3qs. Internet, 1997.

[WH97b] WebSQL-Homepage. http://www.cs.toronto.edu/~websql. Internet, 1997.

[W3C95] The W3C. *A little History of the World Wide Web.* 1995 `http://www.w3.org/WWW/`