Verwaltung von E–mails in Datenbanken? — *emailDB*!

Joachim Kröger, Andreas Heuer

Universität Rostock
Fachbereich Informatik
Lehrstuhl Datenbank– und Informationssysteme
18051 Rostock

{jo, heuer}@informatik.uni-rostock.de
http://wwwdb.informatik.uni-rostock.de/

1 Problemanalyse — Motivation

E–mails bilden ein wichtiges Kommunikationsmedium, das aus dem ingenieur–wissenschaftlichen Bereich — speziell der Informatik — heute nicht mehr wegzudenken ist. Mit dem Einzug des Internet in immer mehr Gebiete auch des nicht–wissenschaftlichen, kommerziellen und privaten Lebens steht die E–mail–Funktionalität einer wachsenden Anzahl an Benutzern zur Verfügung und gewinnt als Medium zunehmend an Bedeutung (in [Hug98] wird für die USA im Jahre 1998 der durchschnittliche Anteil elektronischer Post am Gesamtaufkommen der Post mit 30% angegeben). Ausdruck findet dies in der immer häufiger anzutreffenden Angabe einer E–mail–Adresse neben Telefon– und Faxnummern auf Visitenkarten, in Fernseh– und Radiospots sowie Informationsmaterial von Großunternehmen wie Banken, Versicherungen, Fluggesellschaften und Buchverlagen. Aber auch kleinere und mittelständische Unternehmen wie Autohäuser, Zeitungen und Taxiunternehmen und selbst Privatpersonen verfügen bereits immer häufiger über eine eigene E–mail–Adresse, so daß von einer relativ breiten (und zudem dem aktuellen Trend gemäß rasant weiter steigenden) Verfügbarkeit der E–mail–Funktionalität ausgegangen werden kann.

Die Verwaltung von E–mails durch den Nutzer findet zumeist immer noch in speziellen flachen Dateien, den sogenannten Mailboxen oder Foldern, statt, obwohl E–mails gegebenermaßen über eine reiche Strukturierung verfügen. Der Struktur der E–mails wird dabei aber kaum Beachtung geschenkt; sie bleibt bei der Speicherung sogar vollkommen unberücksichtigt, wenn E–mails einfach als flache Texte abgelegt werden. Mailtools erkennen die vorhandene Struktur der E–mails lediglich durch Interpretation zur besseren Darstellbarkeit. Aufgrund dieser Behandlung von E–mails ist zum Beispiel die vorstellbare Anfrage–Funktionalität stark, d.h. auf nur sehr wenige realisierte Möglichkeiten, eingeschränkt. Auch können Sichten als spezielle Anfragen auf E–mail–Foldern oder Bezüge zwischen E–mails nur schwer oder gar nicht erzeugt/dargestellt werden. E–mails können dabei in der Regel nur navigierend über ein Attribut — meist das Datum ihres Eintreffens beim Empfänger — zugegriffen werden.

Hauptkritikpunkt an den derzeit vorhandenen Verwaltungs–Möglichkeiten von E–mails ist jedoch die unzureichende Behandlung von replizierter und verteilter Speicherung. Nutzt ein user E–mail-Funktionalität auf einer dedizierten Mailbox von verschiedenen Systemen aus, beispielsweise den im Unix–Homedirectory liegenden E–mail–Eingangs–Folder vom Büro–Arbeitsplatz oder von zu Hause aus über unter WindowsNT verwendete Mailtools, so hat er jeweils zu entscheiden, ob er E–mails in das aktuelle lokale System übernehmen will oder sie auf dem Server belassen möchte. Auf diese Weise werden die E–mails entweder verteilt oder repliziert gespeichert und Informationen über den Bearbeitungs–Status von E–mails auf den verschiedenen Systemen unabhängig voneinander geführt; zumindest diese Status–Informationen können später zumeist nicht mehr integriert werden.

Diese Art der Behandlung von E–mails scheint aus einer Datenbank–zentrierten Sicht keinesfalls angemessen zu sein. Wünschenswert wäre es auch in einer vorliegenden Client/Server–Architektur vielmehr, alle Informationen über den Bearbeitungs–Status aller E–mails immer möglichst konsistent auf allen Systemen zur Verfügung zu haben und die tatsächlich vorliegende Verteilung oder Replikation von E–mails eindeutig dokumentieren zu können.

Unser Ansatz *emailDB* kombiniert durch die Speicherung von E–mails in einer Datenbank E–mail– und Datenbank–Funktionalitäten. Dadurch wird es ermöglicht, Informationen über den Bearbeitungs–Status von E–mails mit gewissen Einschränkungen auch über Systemgrenzen hinweg konsistent zu halten und darüber hinaus auch zusätzliche Anfrage–Funktionalität und Möglichkeiten zur Sicht–Definition sowie zur Darstellung von Bezügen zwischen E–mails anzubieten. Außerdem läßt sich auf diese Weise die Problematik der Verwaltung von Verteilung und Replikation wesentlich besser lösen, als dies in den meisten der existierenden Systeme derzeit der Fall ist.

Im folgenden Abschnitt 2 wird die Lösungsidee von *emailDB* mit den sich bietenden Möglichkeiten genauer vorgestellt. Anschließend wird in Abschnitt 3 exemplarisch aufgezeigt, inwieweit *emailDB* eine Herausforderung für die Datenbankforschung darstellt und ein vorstellbarer Realisierungsvorschlag skizziert, bevor in Abschnitt 4 eine Zusammenfassung und ein Ausblick auf das weitere Vorgehen zur vollständigen prototypischen Umsetzung von *emailDB* gegeben werden.

2 Lösungsidee

Wir gehen im weiteren von der bereits Eingangs skizzierten beispielhaften Architektur aus: die E–mails unseres Nutzers werden unter Unix empfangen und in einem E–mail–Eingangs–Folder in seinem Unix–Homedirectory gespeichert. Der Nutzer liest, beantwortet und schreibt seine E–mails zunächst unter Unix beispielsweise mittels elm oder den fortgeschritteneren Versionen mutt oder pine. Hierbei gibt es zwar Einschränkungen bezüglich der Anfrage–Funktionalität und der Verwendung von Sichten, Replikate werden aber — sofern die E–mails nicht explizit in verschiedenen Foldern redundant gespeichert werden — nicht erzeugt.

Will der Nutzer nun seine E–mails auch von zu Hause aus, und dies, um längere Verbindungs–(online-) Zeiten zu vermeiden, offline bearbeiten, so muß er die zu bearbeitenden E–mails auf das lokale System, beispielsweise in das unter WindowsNT verfügbare Programm Pegasus, übernehmen. Zu den beschriebenen Einschränkungen tritt nun auch die Problematik der Behandlung von Replikaten bzw. der verteilten Speicherung von E–mails auf. Auch die einheitlich Verwendung von einem E–mail–Programm auf beiden Systemen, beispielsweise Netscape–Mail, kann dieses Problem nicht umgehen, da eine Übertragung bzw. Verschiebung von E–mails stattfindet. Gleichzeitig werden Status–Informationen, wie etwa "diese E–mail wurde gelesen" und "diese E–mail wurde beantwortet" im Allgemeinen nicht zurück geliefert. Lediglich die Information "diese E–mail wurde übertragen" wird ggf. gespeichert.

emailDB ist eine klare Client/Server–Lösung, die darauf basiert, daß alle E–mails des Nutzers zunächst (automatisch bei Eingang im Unix–Homedirectory) in einem Datenbank–Server gespeichert werden. Aufgrund der reichhaltigen Struktur von E–mails bietet sich ein objektorientierter Datenbankentwurf geradezu an. Attribute wie "From", "Date", "Subject", "Message–Id", "To", "Cc", "Bcc", "Priority", "In-Reply–To", "Reply–To" und der eigentliche E–mail–Body ("Text") sind ganz offensichtlich für eine Speicherung hervorragend geeignet. Darüber hinaus sollte ein Attribut "List of Attachments" nicht fehlen. Über den Bearbeitungs–Status Auskunft gebende Informationen, wie "Already–Read", "Already–Replied" und zur Identifikation verschickter E–mails "Sent" können in der Klassenhierarchie eines objektorientierten Entwurfs durch die Definition entsprechender Unterklassen der Klasse "email" implizit dargestellt werden.

Eine darüber hinausgehende Speicherung weiterer Attribute aus dem E–mail–Header ist natürlich auch ohne weiteres möglich und wird hier aufgrund von Platz–Restriktionen nicht ausführlich betrachtet; einige Attribute (z.B. "Return-receipt-to" oder "X–mailer") können evtl. gezielt ganz weggelassen werden, da sie für den Empfänger einer E–mail nicht unbedingt von besonderem Interesse sind; für andere, nicht standardmäßig verwendete Attribute, bietet sich eine Speicherung als generische Attribute an. Der "Content–type of Attachment x" kann implizit dadurch gespeichert werden, daß für verschiedene Attachments Spezialisierungen des Objekt–Typs "Attachment" verwendet werden. *emailDB* kann ohne weiteres auf einem objektorientierten oder objektrelationalen Datenbankmanagementsystem (DBMS) und mit einigen Änderungen auch auf einem relationalen DBMS umgesetzt werden (s. Unterabschnitt 3.2).

Der Zugriff auf E–mail–Funktionalität erfolgt in *emailDB* über ein Interface, beispielsweise einen Internet–Browser. Der Vorteil liegt hier in der breiten Verfügbarkeit auf allen Plattformen. Nicht der Zugriff ist aber interessant, sondern was während des Zugriffes auf Datenbankseite geschieht. Unabhängig davon, von welcher Plattform zugegriffen wird, werden zunächst unmittelbar nach Verbindungsaufbau alle ggf. auf dem Client veränderten Status–Informationen aus früheren Sitzungen zum Server übertragen. Dann werden auf dem Server vorliegende veränderte Status–Informationen zu allen noch auf dem lokalen System vorliegenden E–mails zum Client übertragen. Hilfreich sind hier das Attribut "Message–Id" und das oben noch nicht eingeführte Attribut "Replicated–On–Client x", einer Liste von Client–Namen (oder genauer: Client–Identifikatoren). Erst im Anschluß an diesen Informationsaustausch ist die Weitergabe von E–mails an den Client vorgesehen.

Durch die vorgenommene Synchronisation soll sichergestellt werden, daß die dem Nutzer angebotenen Daten so konsistent wie nur irgend möglich sind; eine Garantie für Konsistenz kann letztlich aber aufgrund möglicherweise fehlender Synchronisation am Ende einer Sitzung nicht gegeben werden.

emailDB vermeidet eine verteilte (nicht–replizierte) Speicherung, weil die Client–Systeme beispielsweise aufgrund vorhandener Systemgrenzen und fehlender Synchronisation mangels konstant verfügbarer

Verbindungen nicht per default jederzeit erreichbar sind. Dies wird dadurch erreicht, daß alle E–mails auf dem Server belassen und auf den Clients lediglich Replikate erzeugt werden. Der klare Vorteil dieser Lösung liegt darin, daß alle E–mails immer und von jedem beliebigen Client aus verfügbar sind. Einzige Ausnahmen sind hier die von einem Client verschickten E–mails, wenn sie nicht über den Server verschickt werden (um den Datenbestand hier konsistent zu halten reicht es aber prinzipiell schon aus, wenn von jeder E–mail eine Kopie an den eigenen E–mail–Account geschickt wird) und E–mails, die nicht über den Server empfangen wurden (also etwa über einen weiteren E–mail–Account eingegangene E–mails).

Der Nutzer wird bei der Verwaltung von Replikaten durch *emailDB* unterstützt. Zum Teil (soweit nur ausgezeichnete Clients betroffen sind, auf denen der Nutzer regelmäßig verkehrt), kann die Replikatverwaltung sogar vollständig automatisiert werden.

3 *emailDB* als Herausforderung für die Datenbankforschung?

In Unterabschnitt 3.1 wird exemplarisch aufgezeigt, daß *emailDB* keine reine Anwendung ist, sondern vielmehr mit der Thematik einer Reihe von aktuellen Datenbank–Forschungsproblemen eng verknüpft ist. Anschließend wird in Unterabschnitt 3.2 ein vorstellbarer Realisierungsvorschlag skizziert.

3.1 Herausforderungen

Als Vorbild für die Betrachtungen zu Konsistenzproblemen und der Verwaltung von Replikaten dient die Behandlung mobiler Aspekte, wie sie beispielsweise im Projekt MoVi [HL98] vorgenommen wird. Dort kann Konsistenz ebenfalls — etwa aufgrund von möglicherweise auftretenden Verbindungsunterbrechungen — nicht generell zugesichert werden. Dies gilt insbesondere für evtl. vorliegende Replikate [Ber98]. In MoVi wird daher versucht, Replikationsverfahren für mobile Anwendungen mittels verschiedener Parameter gezielt zu konfigurieren [HL99]. Diese Grundproblematik mobiler Systeme findet sich auch in *emailDB* wieder; hier ist der mobile Nutzer von E–mail–Funktionalität der Ausgangspunkt für die entstehende Notwendigkeit, Replikate zu verwalten. Den Verbindungsunterbrechungen in MoVi entspricht eine zeitweise fehlende Synchronisation durch offline–Zeiten in *emailDB*; hier kann es (zumindest zeitweise) zu Inkonsistenzen kommen, die geeignet zu beheben sind.

Bei Verwendung eines objektorientierten DBMS ist es möglich, die Anfragefunktionalität von ODMG–OQL [C$^+$97] umzusetzen, wie im Rahmen des CROQUE–Projektes [KH99b] prototypisch nachgewiesen wurde. Damit kann diese Anfragefunktionalität auch in *emailDB* erreicht und dem Nutzer für Anfragen über seinem gesamten E–mail–Bestand zur Verfügung gestellt werden. Dadurch sind auch Querbezüge zwischen E–mails recherchierbar. *emailDB* wird dem Nutzer neben einer Formular–orientierten Oberfläche auch die Möglichkeit bieten, direkt mittels OQL–Anfragen auf seine E–mails zuzugreifen. Die Schaffung einer QBE–ähnlichen Anfrageschnittstelle ist darüber hinaus ebenfalls denkbar.

In CROQUE wurde mit der Optimierung objektorientierter Anfragen [GKG$^+$97] ein weiterer Aspekt untersucht, der sich — wenn auch in einer etwas anderen Form — auch in *emailDB* wiederfindet: lokal auf einem Client gestellte Anfragen können dahingehend optimiert werden, daß in gewissen Fällen auf einen Verbindungsaufbau zum Server ganz verzichtet werden kann. Dies ist immer dann möglich, wenn die beim letzten Verbindungsaufbau hergestellte Konsistenz (die sich oftmals als Anfrage zur Bestimmung der zu übertragenden Daten ausdrücken läßt) die Möglichkeit indiziert, die aktuell vom Benutzer gestellte Anfrage lokal zu beantworten. Weiterführend ist es ohne Verbindung zum Server denkbar, die Antworten auf Nutzeranfragen — etwa mittels eines Nutzerprofils — heuristisch abzuschätzen. Der Nutzer erhält dann ein nach der Wahrscheinlichkeit der Korrektheit geranktes Ergebnis mit dem Hinweis auf möglicherweise zwischenzeitlich durch ihn selbst (etwa auf einem weiteren Client) geänderte Daten. Da der Nutzer sein Nutzungsprofil selbst einstellen kann und seine E–mails exklusiv bearbeitet, kann dabei eine hohe Genauigkeit erreicht werden.

Heterogene Anfragebearbeitung, also die Bearbeitung von Anfragen auf unterschiedlichen Plattformen, wird derzeit beispielsweise im GETESS–Projekt behandelt [DHKP99]. Eine solche Situation tritt auch in *emailDB* auf, da auf Client und Server verschiedene DBMS mit unterschiedlicher Anfragesprachen–Syntax und Sprach–Mächtigkeit zum Einsatz kommen können; *emailDB* wird auf Grundlage verschiedener DBMS implementiert (s. auch Abschnitt 3.2). GETESS beschäftigt sich darüber hinaus mit semistrukturierten Daten [KH99a]. E–mails weisen, wie in den Abschnitten 1 und 2 hervorgehoben wurde, eine reiche Strukturierung auf; insgesamt sind sie jedoch — aufgrund der Möglichkeit, unterschiedlichste Attachments zu beinhalten — auch in die Klasse der semistrukturierten Daten einzuordnen.

3.2 Realisierungsvorschlag

Um den in Abschnitt 2 skizzierten objektorientierten Datenbankentwurf adäquat umsetzen zu können, werden zunächst nur objektorientierte und objektrelationale DBMS in Betracht gezogen und auf eine Umsetzung auf relationalen DBMS (wie etwa Ingres) verzichtet, obwohl dies ebenfalls möglich scheint.

Haupt–Vorteile von objektrelationalen DBMS wie Informix und DB2 liegen darin, geschachtelte Strukturen verwenden und neue Datentypen (etwa Text oder strukturierter Text) definieren zu können. Allgemein gilt jedoch die Programmierschnittstelle, die derzeit noch aus einer Kombination von Programmiersprache und Cursor–Konzept besteht, als Nachteil.

Das objektorientierte DBMS O_2 bietet ebenfalls die Möglichkeit, geschachtelte Strukturen zu verwenden, erlaubt aber darüber hinaus an der Programmierschnittstelle die Verwendung von Java und C++. Erweiterbare Datentypen werden über eine intern in der Klassenhierarchie durchgeführte Definition erreicht. Als Haupt–Nachteile sind hier Probleme mit der Stabilität, dem Transaktionskonzept und der Rechtevergabe zu nennen. Außerdem sind heterogene Strukturen nicht angemessen darstellbar [Web98].

Als erste Wahl für die Umsetzung von *emailDB* werden zunächst die objektrelationalen DBMS favorisiert. Aufgrund der Programmierschnittstelle wird aber auch O_2 für eine Testimplementierung herangezogen werden.

Die Realisierung von *emailDB* sieht vor, daß die im Unix–Homedirectory eingehenden E–mails geparst und im verwendeten DBMS gespeichert werden. Vom Client über einen Browser geschickte Datenanforderungen werden in Datenbankzugriffe übersetzt und die Anfrage–Ergebnisse an den Client zurück übertragen. Auf Client–Seite besteht dann die Möglichkeit, die E–mails entweder in Foldern abzulegen (falls kein unterstütztes DBMS vorhanden ist) oder in einer Datenbank abzulegen.

Sowohl die Datenanforderungen, als auch die Übertragung der Anfrage–Ergebnisse geschieht dabei mittels eines definierten Protokolls. Das am weitesten verbreitete Mail–Zugriffs–Protokoll POP3 [MR96] wurde ursprünglich für den offline–Mailmodus entwickelt, bei dem die E–mails in der Regel auf den Client übertragen und auf dem Server gelöscht werden. E–mails werden hierbei in jedem Falle verteilt — evtl. auch "nur" verteilt repliziert — gespeichert, das Protokoll sieht aber keine Re–Integration dieser E–mails bzw. der zugeordneten Status–Informationen ohne Zuhilfenahme von Remote Filesystem–Protokollen für gewisse Aktionen vor [Gra95]. POP3 kommt damit für eine Implementierung von *emailDB* nicht in Frage.

Da *emailDB* auf dem Grundgedanken basiert, E–mails zu verschiedenen Zeiten von verschiedenen Systemen aus zuzugreifen, ist das Verbleiben aller E–mails auf dem Server absolut notwendig. Darüber hinaus sollen neben dem offline–Mailmodus sowohl online– als auch disconnected–Zugriffe erlaubt sein. Disconnected bedeutet dabei, daß Replikate ausgewählter E–mails auf dem Client abgelegt werden und nach einer offline–Arbeitsphase des Nutzers auf dem Client zu Beginn des nächsten Kontakts zum Server eine Synchronisation stattfindet (vgl. Abschnitt 2). Da POP3 diesen Zielen nicht genügt, wurde das Protokoll IMAP4 [Cri96] entwickelt, welches auch im Rahmen von *emailDB* Verwendung findet.

Durch die Abbildung des IMAP4–Protokolls in Datenbank–Operationen wird das DBMS quasi zu einem IMAP4–Mail–Server ausgebaut. Die Abbildung wird für verschiedene DBMS implementiert, u.a. um einen Qualitäts–Vergleich zu erlauben. Jede dieser Implementationen kann dazu genutzt werden, auf dem Client eine entsprechende Datenbank zur Speicherung replizierter E–mails zu verwalten. Damit werden die DBMS bis auf die fehlende Nutzerschnittstelle auch gleichzeitig zu IMAP4–Clients.

In der ersten Implementierungsphase von *emailDB* wird zunächst voraussichtlich ein Internet–Browser als Nutzerschnittstelle dienen. Der Vorteil dieser Lösung liegt in der breiten Verfügbarkeit, die *emailDB* plattformübergreifend verwendbar macht und dem Nutzer eine einheitliche Oberfläche anbietet, die auch "von unterwegs" nutzbar ist. In späteren Phasen ist auch die direkte Ankopplung an Public–Domain Implementierungen geplant, die durch Einhaltung des IMAP4–Protokolls erleichtert wird.

4 Zusammenfassung und Ausblick

E–mails sind bereits heute — zumindest in gewissen Bereichen, wie etwa der Informatik — Massendaten. Für weitere Anwendungsbereiche ist dies in Folge des aufgeführten Trends der steigenden Nutzerzahlen zumindest zu erwarten. E–mails sind darüber hinaus wohl strukturiert.

Das Problem der Verwaltung strukturierter Massendaten bildet bekanntermaßen die klassische Ausgangssituation für die Verwendung von Datenbanken [HS97]. Somit bietet es sich für die Verwaltung von E–mails an, eine Datenbanklösung in Betracht zu ziehen. Auch die weiter genannten Probleme bei der Verwaltung von E–mails, wie fehlende Anfragemöglichkeiten und die Frage nach der Behandlung von Replikaten, sind im Datenbankbereich erforscht, so daß sich eine umfassende Datenbanklösung, wie sie in Form von *emailDB* angedacht ist, geradezu anbietet.

Die erste prototypische Implementierung von *emailDB* auf verschiedenen DBMS wird im Rahmen eines vorlesungsbegleitenden Praktikums zu "Komplexen Software–Systemen" am Fachbereich Informatik der Universität Rostock stattfinden. Diese Implementierungen werden sich auf frei verfügbare Quellcode–Bibliotheken zu IMAP4 stützen und dabei evtl. als "kleine Lösung" auch eine SQL–Datenbank berücksichtigen, um eine Client–Lösung zu schaffen, die ohne eines der genannten, umfangreicheren objektrelationalen bzw. objektorientierte DBMS auskommt, da diese nicht ohne weiteres als auf einem Client verfügbar angenommen werden können.

Die notwendige Synchronisation, die wesentliche Teile des Problems der Replikat–Verwaltung löst, wird durch Log–Files auf Client– und Server–Seite unterstützt. Untersuchungen zu dieser Thematik werden einen wesentlichen Teil der unmittelbar folgenden Forschungsarbeiten umfassen, da die IMAP4–Spezifikation [Cri96] gerade an dieser Stelle nicht ausreichend fortgeschritten ist [Gra96].

Als Erweiterung der hier nur skizzierten Lösung von *emailDB* ist bereits angedacht, Attribute, die den Absender näher spezifizieren, wie "Organization", "Location", "Phone", "Address" und "X-face" zu extrahieren und in einer eigenen Klasse "Sender" zu speichern. Den E–mails könnte durch Zuordnung eines "Sender"–Objekts der Absender ohne Informations–Verlust eindeutig zugeordnet werden, während gleichzeitig eine Verfeinerung des objektorientierten Entwurfs erreicht würde. Das zu speichernde Datenvolumen würde sich durch diese Maßnahme allerdings nur bei Vorliegen von sehr vielen sehr kurzen E–mails weniger Absender deutlich verringern.

Literatur

[Ber98] Berger, B.: Auswahl und Weiterentwicklung eines Replikationsverfahrens für den mobilen Datenbankzugriff im Projekt MoVi. Studienarbeit, Universität Rostock, Fachbereich Informatik, November 1998.

[C+97] Cattell, R. et al. (Hrsg.): *The Object Database Standard: ODMG 2.0.* Morgan-Kaufmann, San Francisco, CA, 1997.

[Cri96] Crispin, M.: *RFC2060: Internet Message Access Protocol – Version 4rev1*, Dezember 1996. http://www.imap.org/docs/rfc2060.html.

[DHKP99] Düsterhöft, A.; Heuer, A.; Klettke, M.; Priebe, D.: GETESS: Der Text-orientierte Anfrage- und Suchdienst im Internet. In: *11. Workshop "Grundlagen von Datenbanken"*, Mai 1999.

[GKG+97] Grust, T.; Kröger, J.; Gluche, D.; Heuer, A.; Scholl, M.: Query Evaluation in CROQUE — Calculus and Algebra Coincide. In: *Proc. of the 15th British National Conference on Databases (BNCOD 15), London, UK, LNCS 1271, Springer*, S. 84–100, Juli 1997.

[Gra95] Gray, T.: *Message Access Paradigms and Protocols*, September 1995. http://www.imap.org/imap.vs.pop.html.

[Gra96] Gray, T.: *Status of Disconnected Operation in IMAP*, Juli 1996. http://imap.org/docs/disc-status.html.

[HL98] Heuer, A.; Lubinski, A.: Data Reduction - an Adaptation Technique for Mobile Environments. In: *Proc. of the Interactive Applications of mobile Computing Conference (IMC'98)*, 1998.

[HL99] Heuer, A.; Lubinski, A.: Configured Replication for Mobile Applications. 1999. Submitted for publication.

[HS97] Heuer, A.; Saake, G.: *Datenbanken - Konzepte und Sprachen.* International Thomson Publishing, Bonn, 1. korrigierter Nachdruck, 1997.

[Hug98] Hughes, L. (Hrsg.): *Internet E–mail: protocols, standards, and implementation.* Artech House, Norwood, MA, 1998.

[KH99a] Klettke, M.; Heuer, A.: Informationsspeicherung in GETESS oder Die Strukturierung der Semistrukturiertheit. In: *11. Workshop "Grundlagen von Datenbanken"*, Mai 1999.

[KH99b] Kröger, J.; Heuer, A.: Optimierung objektorientierter Anfragen: Das Projekt CROQUE. Technischer Bericht, Fachbereich Informatik, Universität Rostock, März 1999. Eingereicht zur Veröffentlichung.

[MR96] Myers, J.; Rose, M.: *RFC1939: Post Office Protocol – Version 3*, Mai 1996. http://www.imap.org/docs/rfc1939.html.

[Web98] Weber, G.: Integration von Literaturdatenbanken über das WWW. Diplomarbeit, Universität Rostock, Fachbereich Informatik, Juni 1998.