Ein XML-basiertes Modell für synchrone Gruppenarbeit auf gemeinsamen Informationsräumen

von
Dipl. Math. Morad Ahmad

Dissertation
vorgelegt am Fachbereich Mathematik/Informatik
der Universität Kassel
zur Erlangung des Doktorgrades der Naturwissenschaften
(Dr. rer. nat.)

Kassel 2003

Hiermit versichere ich, daß ich die vorliegende Dissertation selbständig und ohne unerlaubte Hilfe angefertigt und andere als die in der Dissertation angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Kein Teil dieser Arbeit ist in einem anderen Promotions- oder Habilitationsverfahren verwendet worden.

Vom Fachbereich Mathematik/Informatik der Universität Kassel als Dissertation angenommen am 30. Mai 2003.

Erstgutachter: Prof. Dr. L. Wegner, Universität Kassel
Zweigutachter: Prof. Dr.-Ing. B.J. Krämer, FernUniversität in Hagen

Tag der mündlichen Prüfung: 25. Juli 2003

Inhaltsverzeichnis

Danksagung

Die vorliegende Arbeit entstand während meiner Arbeit an der Fachgruppe Praktische Informatik der Universität Kassel.

Mein Dank gilt zu allererst *Prof. Dr. Lutz Wegner* für seine hilfreichen Hinweise und wertvollen Anregungen.

Herzlichen Dank an Frau *Inge Wilmsmeier* für die sprachliche Kontrolle und Hinweise auf Wiederholungen und Inkonsistenzen.

Kapitel 1

Einleitung

Durch die Verbreitung des Internets im geschäftlichen und privaten Leben ist eine weltweite Rechnerkommunikation möglich geworden. Im Laufe dieser Entwicklung haben sich Webbrowser zu einer universellen Schnittestelle für die verschiedensten Anwendungsbereiche gewandelt. Die spätere Einführung von mobilen Geräten und die rapide Entwicklung von Techniken der mobilen Kommunikation ermöglicht den Einsatz dieser Geräte für den Zugriff auf Daten im Internet [107]. Durch UMTS-, und WLAN-Technologien können Mobilgeräte für Anwendungen über die reine Kummunikation hinaus eingesetzt werden. Eine natürliche Folge dieser Entwicklung sind Anwendungen für die Zusammenarbeit mehrerer Benutzer (Groupware). Eine spezielle Ausprägung solcher Anwendungen ist die Zusammenarbeit auf der Basis gemeinsamer Daten.

Einfache tragbare Geräte mit ausreichender Rechenkapazität und Anzeigemöglichkeiten werden mit *zentralen* Server kommunizieren und ihre Dienste nutzen. Verschiedene Hersteller, Geräte mit unterschiedlichen Anzeigemöglichkeiten und Geschwindigkeiten werden Daten aus verschiedenen Quellen zusammenstellen, und visualisieren. Die verschiedenen Geräten werden diese Daten auch unterschiedlich darstellen, so daß die Trennung zwischen Inhalt und Visualisierung notwendig ist.

Sowohl eine gemeinsame Sprache zum Austausch der Daten, als auch eine einfache Zugriffsmöglichkeit und Bedienung, sind erforderlich, um solche Dienste zur Verfügung zu stellen. Eine natürliche Form der Interaktion auf solchen Daten ist *Navigation*. Verschiedene Benutzer werden mit unterschiedlichen Endgeräten aus verschiedenen Orten auf gemeinsame Informationen navigieren. Das gemeinsame Navigieren an ,,benachberten" Stellen wird zum Zeichen gemeinsamer oder ähnlicher Interessen interpretiert. *Konflikte* können entstehen aus dem Versuch Daten gleichzeitig zu ändern oder zu reservieren. Es werden sich also *Gruppen* spontan bilden und aufheben.

Es handelt sich also um eine Form der Computergeschtützten Gruppenarbeit auf gemeinsammen Informationsräumen, die ausgeprägt ist durch den Einsatz verschiede-

ner Endgeräte, die auf eine zentrale Datenquelle zugreifen. Gemeinsame Informationsräume implizieren die Verwendung eines Datenmodells, das sowohl die Struktur der Daten als auch die Operationen auf diese festlegt. Durch die gleichzeitige Zusammenarbeit mehrerer Benutzer müssen Informationen bereitgestellt werden, die den Anwender Gefühl über die Aktivitäten anderer Benutzer vermitteln (Awareness). Die Visualisierung der Daten bei einem Benutzer und die Darstellung von Awareness müssen integriert werden. Ebenso müssen neue Mechanismen für die Kontrolle der Nebenläufigkeit entwickelt werden, die den Einfluß von Awareness berücksichtigen.

Bei der Verfolgung der CSCW-Literatur und verschiedene Softwareprojekte ist es auffallend, daß diese Anwendungsgruppe nicht genügend Aufmerksamkeit genießt. Die meisten Systeme auf gemeinsamen Informationsräumen sind asynchron; Die synchronen Systeme gehen nicht über die Kommunikation hinaus. Neben der Charakterisierung dieser Anwendungsklasse fehlt ein Gesamtmodell, das die verschiedenen Anforderungen dieser Systeme zusammenbringt.

Die vorliegende Arbeit beschäftigt sich mit einer speziellen Form der computergestützten Gruppenarbeit, genauer der *synchronen Gruppenarbeit auf gemeinsamen Informationsräumen*. Der Begriff *synchron* bezieht sich dabei auf das gleichzeitige Zusammenwirken der Teilnehmer im Gegensatz zu einer asynchronen Gruppenarbeit etwa auf der Basis von elektronischen Nachrichten. Eine präzisere Klassifikation folgt in Abschnitt 1.1.1. Während mit synchroner Gruppenarbeit meistens *chat rooms* oder *Video-Konferenzen* assoziert werden, steht die Koordinierung auf gemeinsamen Informationsräumen im Mittelpunkt dieser Arbeit. Natürlich kann diese Form der synchronen Kooperation durch Kommunikationskanäle (Video, Audio, Textströme) unterstützt werden, sie sind aber nicht Teil des vorgeschlagenen Modells[1].

Das Ziel dieser Arbeit ist die Formulierung eines generischen Modells für eine breite Anwendungsklasse von Kooperationsaufgaben und eine prototypische Implementierung dieses Modells. Die Arbeit ist also anwendungsorientiert. Sie soll Anwendungsmöglichkeiten für synchrone Gruppenarbeit auf gemeinsamen Informationsräumen und mögliche Anwendungsszenerien vorstellen. Ausgehend von diesen Szenerien werden die Anforderungen an die Funktionalität eines Modells für synchrone Gruppenarbeit und mögliche Lösungen abgeleitet[2].

Für die hier behandelte Form der synchronen Gruppenarbeit sind drei Aspekte von besonderer Bedeutung:

- das Datenmodell für den gemeinsamen Informationsraum.
- die Kontrolle der Nebenläufigkeit.
- die Signalisierung der Aktionen (Awareness).

[1] Tats¨achlich existieren Mechanismen f¨ur die synchrone Kommunikation schon so lange wie es E-Mail-Dienste gibt. Beispiel hierf¨ur ist das UNIX `write` Kommando aus den 70-er Jahren.

[2] Mit synchroner Gruppenarbeit meinen wir, sofern nichts anders erw¨ahnt wird, synchrone Gruppenarbeit auf gemeinsamen Informationsr¨aumen.

Dabei gibt es einen engen Zusammenhang zwischen Awareness und der Kontrolle der Nebenläufigkeit. Die Lösungen für die Nebenläufigkeitskontrolle und das Awareness-Modell müssen die praktischen Einsatzmöglichkeiten für Gruppenarbeit berücksichtigen. Dies erfordert die Erforschung der Akzeptanz und die Reaktion der Anwender auf die Software (sozialen Aspekt).

Für die Implementierung eines Modells sind natürlich noch andere Aspekte zu beachten. Wichtig ist eine angemessene Modellierung des gesamten Systems und die Angabe einer Schichtenarchitektur, in der bestimmt wird, welche Funktionen jede Schicht bereitstellt und wie die verschiedenen Schichten des Systems zusammenarbeiten.

Systeme zur Unterstützung der Gruppenarbeit sind naturgemäß Mehrbenutzersysteme. Deshalb müssen sie Mechanismen für die Benutzerverwaltung, z. B. das An-/Abmelden der Teilnehmer vorsehen. Die Themen Zugriffskontrolle und die Benutzerverwaltung (*Access Control*) sind jedoch Teil der meisten Groupware-Anwendungen und keine spezifischen Merkmale der synchronen Gruppenarbeit. Die Zugriffskontrolle wird in dieser Arbeit daher am Rande behandelt.

In dieser Arbeit wird von der Annahme ausgegangen, daß das Datenmodell für den gemeinsamen Informationsraum die zentrale Basis des gesamten Systems ist. Obwohl die Modelle und Verfahren für Awareness und die Nebenläufigkeitskontrolle generisch, d. h. vom Datenmodell unabhängig sind, hängt die Effizienz des gesamten Systems von der Implementierung des zugrunde liegenden Datenmodells stark ab.

Für die Kommunikation zwischen den Anwendungen und dem Groupware-Server sollen generische und allgemein unterstützte Standards verwendet werden. Angesichts der universellen Verbreitung des Internets bietet sich eine dokumentenzentrierte Kommunikation, etwa auf Basis von XML, an. Die Abbildung der Datenbankinhalte auf Dokumente des verwendeten Austausch- und Visualisierungsmodells sollte dann schnell und effizient implementiert werden. Dies erfordert vom Datenmodell die Möglichkeit isomorpher Speicherungs- und Darstellungsformen wie unten für Bäume, Tabellen und XML-Dokumente gezeigt.

Es wird ein generisches räumliches Modell für synchrone Gruppenarbeit auf strukturierten Datenräumen entwickelt. Es ist generisch, weil es einen einheitlichen Satz von Operationen und Beziehungen bezüglich Nebenläufigkeit und Wahrnehmung (Awareness) anbietet, unabhängig von der Datenrepräsentation und unabhängig davon, ob die Akteure Menschen oder z. B. autonome Agenten sind. Vorausgesetzt wird lediglich ein strukturierter, im wesentlichen hierarchischer, Datenraum, z. B. eine eNF^2-Tabelle, ein XML-Dokument oder ein DOM-Baum.

1.1 Computergest¨utzte Gruppenarbeit

Computer Supported Collaborative Work (CSCW) beschäftigt sich als Forschungsdisziplin mit dem Erstellen von rechnerbasierten Anwendungen zur Unterstützung der

Gruppenarbeit. Die pragmatische Realisierung solcher Systeme wird als *Groupware* bezeichnet. Eine allgemeine Definition für CSCW findet sich in [161].

Definition 1.1
CSCW bezeichnet das interdisziplinäre Forschungsgebiet, das sich mit Gruppenarbeit und der Unterstützung von Gruppenarbeit durch (Computer-) Technologien befaßt.

CSCW ist also ein interdisziplinäres Forschungsgebiet, an dem verschiedene Disziplinen wie Informatik (Datenbanktechnik, Telekommunikation, GUI-Design, Künstliche Intelligenz), Sozialwissenschaften, Wirtschaftwissenschaften und Psychologie beteiligt sind. Einige Beispiele für Groupware-Anwendungen sind: *Elektronische Post-Systeme*, *Mehrbenutzer-Editoren*, *Computer-Konferenzsysteme*, *Bullatin-Board-Systeme (News)*, *computerunterstützte Entscheidungsysteme*, *collaborative software engineering-Systeme* [188].

Die ersten Arbeiten/Ideen in Richtung CSCW wurden von *Engelbert* unternommen. Bereits 1962 hatte er mit dem „*Augment System*", einen Prototyp entwickelt, der video teleconferencing und shared-screen Collaboration ermöglichte [66]. In der Arbeit von *R¨udebusch* wird erwähnt, daß der Begriff Computer Supported Collaborative Work zum ersten Mal 1984 von *Paul Cashman* und *Irene Greif* verwandt wurde [161, S. 6]. Seit diesem Zeitpunkt war CSCW ein eigenständiges Forschungsthema mit eigenen Konferenzreihen etwa CSCW'86, CSCW'88, CSCW'90 usw. sowie, in Europa, ECSCW'89, ECSCW'91 usw.

1.1.1 Klassifikation von CSCW-Applikationen

CSCW-Applikationen lassen sich nach verschiedenen Kriterien klassifizieren. Hier werden zwei dieser Kriterien betrachtet. Die Raum-Zeit-Matrix von *Johansen*, und die Klassifikation von CSCW-Anwendungen nach den unterstützten Funktionen [174].

Nach der von *Johansen* bekannten *Raum-Zeit-Matrix* [110] werden Applikationen nach der Art der Kommunikation der Benutzer miteinander in vier Kategorien unterteilt:

- gleiche Zeit gleicher Ort.
- gleiche Zeit anderer Ort (Synchrone Gruppenarbeit).
- unterschiedliche Zeit gleicher Ort.
- unterschiedliche Zeit anderer Ort (Asynchrone Gruppenarbeit).

Diese Klassifikation ist jedoch sehr grob, und die Einordnung von vielen CSCW-Anwendungen nach diesem Schema fällt schwer. *Johansen* selbst äußert die Vermutung, daß die zukünftig interessantesten Bereiche die Schnittlinien der Quadranten sind

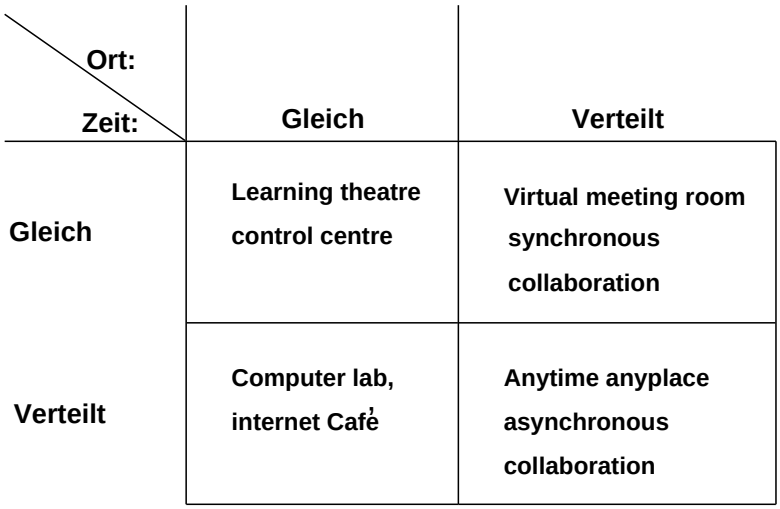


Abbildung 1.1: Klassifi kation von CSCW-Applikationen nach der Raum-Zeit-Matrix (nach [167])

[111]. Ellis behauptet sogar *„a comprehensive groupware system may best fit the needs of all the quadrants"* [64, S. 41].

Für die Qualifizierung nach Unterstützungsfunktionen werden die folgenden Begriffe eingeführt [174]:

Kommunikation ist der Austausch von Informationen zwischen den Benutzern. Dabei kann man zwischen synchroner und asynchroner Kommunikation unterscheiden. Die Kommunikation kann unterschiedliche Ziele haben.

Koordination bezeichnet die Abstimmung der Tätigkeiten der Benutzer untereinander. Koordination kann Ziel einer Kommunikation sein.

Kooperation bezeichnet die Zusammenarbeit verschiedener Benutzer für die Lösung einer gemeinsamen Aufgabe.

Anhand des Unterstützungsgrades dieser Funktionen werden CSCW-Applikationen in vier verschiedenen Klassen unterteilt *Kommunikation, gemeinsame Informationsräume, Workf low Computing* und *Workgroup Computing*.

Kommunikation: Die Unterstützung asynchroner Kommunikation zwischen räumlich verteilten Benutzern ist die älteste und verbreiteste CSCW-Anwendung. Diese sind bekanntlich die E-Mail-Systeme. Sie waren grundsätzlich für den Austausch von Text konzipiert, doch heutzutage sind diese Systeme weitgehend erweitert worden, so daß auch der Austausch von Multimedia Objekten möglich ist. Basierend auf diesen Systemen können die sogenannten *Voice-Mail* und *Video-Mail* aufbauen, indem entsprechend standardisierte Kodierungs- und Dekodierungstechniken bei allen Benutzerapplikationen zur Verarbeitung der elektronischen Post eingesetzt werden [174].

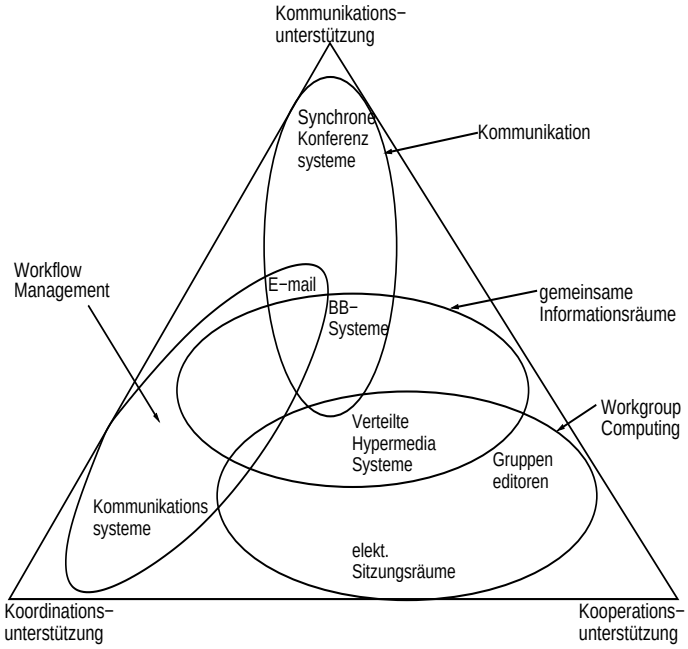


Abbildung 1.2: Klassifi kation von CSCW-Applikationen nach Funktionalit¨at (nach [174, S. 27])

Seit längerer Zeit existieren auch synchrone Kommunikationssysteme, die heutzutage über den Austausch von Texten hinausgehen. *Teufel* unterteilt diese Systeme in drei Kategorien [174]: Textuelle Kommunikation (Sehsinn), Audio-Kommunikation (Gehörsinn), und Video-Kommunikation (Gehör- und Sehsinn)[3]. Beispiele für synchrone Kommunikationssysteme sind textbasierte Konferenzsysteme (`write`, `talk`), audio-, videobasierte Konferenzsysteme, und Desktopkonferenzsysteme, wie etwa das weitverbreitete *Netmeeting*.

Gemeinsame Informationsräume: Jedes CSCW-System, das den Benutzer bei der Bearbeitung von gemeinsamen Daten unterstützt, verwaltet seine Daten in einem sogenannten gemeinsamen Informationsraum. Dabei kann man zwischen Systemen, die einen lesenden Zugriff (Hypertext-Systeme), und Systemen, die lesenden und schreibenden Zugriff auf den gemeinsamen Informationsraum erlauben, unterscheiden. Die Bearbeitung dieser gemeinsamen Daten durch die Benutzer ist implizit eine Kommunikation der Benutzer untereinander. Beispiele für CSCW-Anwendungen, die auf gemeinsamen Informationsräumen aufbauen, sind Bulletin Board-Systeme und News, sowie verteilte Hypertext-Systeme. Heutzutage ist das World Wide Web das bekannteste Beispiel eines verteilten Hypertext-Systems.

[3] Eine Kritik an dieser Klassifi zierung ist die Gleichstellung von Textsystemen und Videosystemen bez¨uglich des ,,Sehsinns". W¨ahrend sich das Sehen bei textbasierten Systemen auf das Lesen beschr¨ankt, sind bei videobasierten Systemen Animationen und damit auch mehr Emotionen von Seite des Benutzers zu beobachten.

Workflow Management Systeme: Unter dem Begriff *Workflow* versteht man eine endliche Folge von Aktivitäten, wobei die Folge durch Ereignisse ausgelöst und beendet wird. Ein Workflow Management-System ist ein aus mehreren Werkzeugen bestehendes System, welches die Aufgaben des Workflow Managements durch die Ausführung von Software unterstützt. Workflow Management Systeme sind besonders geeignet für die Unterstützung von betrieblichen Abläufen, die einen hohen Strukturierungsgrad und eine geringe Komplexität aufweisen [174].

Workgroup Computing: Workgroup Computing sind CSCW-Anwendungen, die mehreren, räumlich verteilten Personen das Arbeiten an gemeinsamen Aufgaben ermöglichen. Dazu gehören Planungssysteme, Gruppeneditoren, Entscheidungsunterstützungssysteme und Sitzungsunterstützungssysteme.

1.2 Eingrenzung der Arbeit

Synchrone Gruppenarbeit auf gemeinsamen Informationsräumen kann durch die folgenden drei Punkte charkterisiert werden:

- Benutzer führen Operationen auf einem gemeinsamen Informationsraum aus.
- Die Aktivitäten der Gruppenteilnehmer auf diesem Raum erfolgen synchron.
- Modifikationen eines Benutzers werden sowohl bei ihm, als auch bei anderen Gruppenmitgliedern dargestellt.

Trotz langjähriger Forschung und Existenz verschiedener Groupware-Anwendungen ist diese Klasse vernachlässigt worden. Der Begriff gemeinsamer Informationsraum wurde immer in Zusammenhang mit **asynchroner Gruppenarbeit** gebracht: „*Such workspaces support primarily asynchronous modes of communication. This mode is normally the most important one for cooperation between researchers since in such an environment cooperation consists often in parallel, loosely coupled activities of the individual group members.*" [23, S. 2]. Auf der anderen Seite wurde immer synchrone Gruppenarbeit auf **audio/video conferencing** oder **chat sessions** reduziert: „*Synchronous types of cooperation such as audio/video conferencing or chat sessions are usually of less importance but should also be supported to some extent.*" [23, S. 2].

Zwar kann man gemeinsame Funktionalitäten von allen CSCW-Applikationen feststellen: „*Building groupware proved a frustrating experience. Implementing even the simplest systems was a lengthy process, as much time was spent inventing similar ideas over and over again.*" [160, S. 2]. Jedoch unterscheiden sich diese bei deren Funktion und Implementierung enorm bei den verschiedenen Anwendungsklassen. „*It is difficult to design a one-size-fits-all groupware system that works well under all potential usage situations*" [127, S. 107].

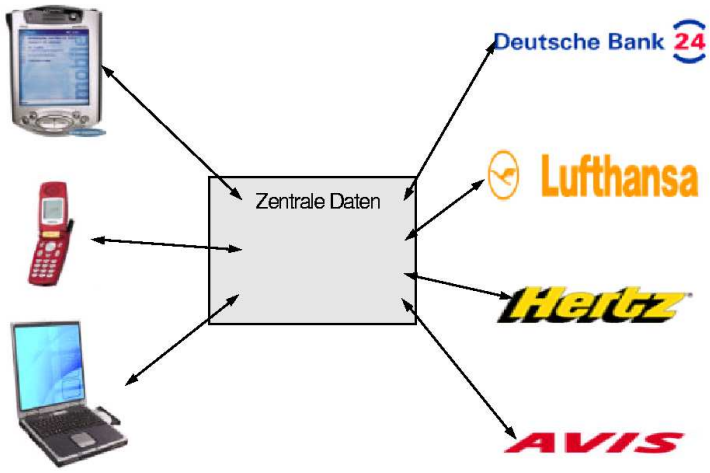


Abbildung 1.3: Gruppenarbeit mit verschiedenen Anzeigem¨oglichkeiten

Ziel dieser Arbeit ist, die Funktionalitäten, die Anwendungen für die synchrone gruppenarbeit benötigen, zuerst zu bestimmen und daraus genaue Anforderungen an ein System für die Implementierung solcher Anwendungen zu stellen. Basierend auf diesen Anforderungen wird dann ein generisches Modell entworfen und in Form eines Prototypen implementiert. Mit dem Wort generisch wird ausgedrückt, daß die Kommunikation zwischen dem Groupware-Server und dem Client auf einem anwendungsunabhängigen Protokoll basiert [159]. Dieses soll von Anwendungen auf verschiedenen Geräten (PC's, Web-Clients, PDA's, Mobiltelefone) interpretierbar sein und nach den Anzeigefähigkeiten des Clients unterschiedliche und individuelle Visualisierungen der Daten ermöglichen [97]. Die Semantik der Benutzeroberfläche sowie die Navigation auf den Daten ist einheitlich, aber ebenfalls anpaßbar und kann daher auf verschiedene Weisen auf diesem Protokoll aufsetzen.

1.2.1 Anwendungsszenerien

Moderne Formen dieser Gruppenarbeit werden beliebige, nicht miteinander assozierte Anwender, autonome Agenten und häufig wechselnde Server zusammenbringen [224]. Die Verbindungen werden oft spontan und kurzlebig sein. Anwender und Agenten werden von Ort zu Ort wandern, z.T. die Anwesenheit anderer Benutzer suchend, z.T. deren Anwesenheit bewußt meidend. Typische Aktivitäten werden Einkäufe, Einholen von Auskünften, Verhandlungen, Erfahrungsaustausch, Reparaturtips, Krisenmanagement, gemeinsames Stellen von Anträgen, offene Abstimmungen, Terminplanung usw.

sein. Vieles spricht dafür, daß die traditionelle Vertriebsformen für Software, also z. B. der Versand einer CD, die Installation einer speziellen Applikation auf dem Rechner des Anwenders usw. für diese Art der Interaktion ungeeignet sind, speziell wenn es auf Anwenderseite nur einen „thin client", also z. B. eine Netzwerkbox, gibt.

Um die Anwendungsmöglichkeiten von synchroner Gruppenarbeit im allgemeinen und die Wichtigkeit des generischen Aspektes im speziellen zu zeigen, werden einige Anwendungszenerien betrachtet. Diese könnten in naher Zukunft typische Anwendungsbeispiele dieser Art der synchronen Gruppenarbeit werden.

Service Center

Viele der heutigen Geschäftsabwicklungen zwischen Anbietern und Verbrauchern werden schon über das Internet abgewickelt [167]. Dabei fehlt in den meisten Fällen ein Papierdokument o.ä., auf das der Kunde bei Problemfällen oder Änderungswünschen zurückgreifen kann. In diesen Fällen greifen die Kunden immer noch auf die traditionelle Telefonverbindung und beschreiben ihr Problem einem Bearbeiter. In manchen Fällen ist sogar ein weiterer Beobachter erforderlich, der z. B. von dem Bearbeiter herangezogen wird. Jeder der drei Teilnehmer könnte in solchen Fällen unterschiedliche Daten bzw. Dokumente besitzen.

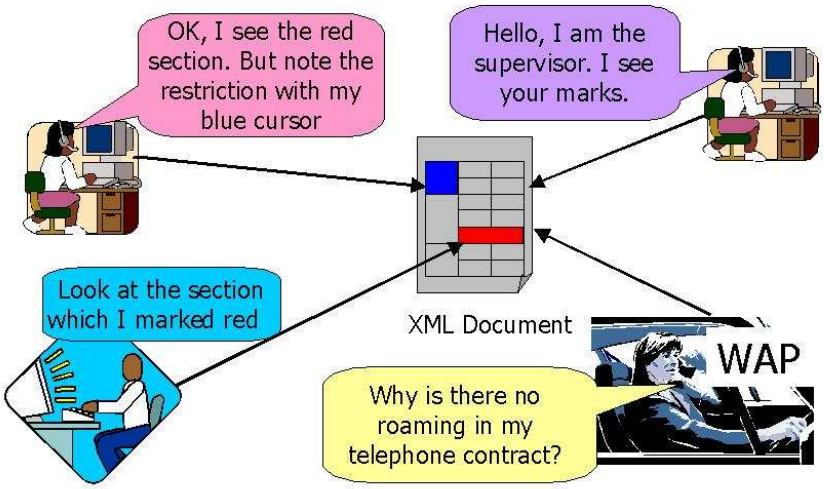


Abbildung 1.4: Anwendungszenario: Service Center

Der Einsatz von Software für die synchrone Gruppenarbeit kann bei solchen Fällen große Hilfe leisten. Diese ermöglicht allen Beteiligten den Zugriff auf die bei der Firma oder vertrauenswürdigen Drittparteien abgelegten Daten. Die Benutzer können zusätzlich eine Audioverbindung zueinander haben und gemeinsam das Problem diskutieren und lösen. In diesem Fall kann der Bearbeiter bestimmte Teile des Dokuments hervorheben, die vielleicht der Kunde sonst in dem Vertrag übersehen hat. Auch der Kunde könnte durch einen *Cursor*[4] zu bestimmten Teilen des Dokuments *navigieren* und über diese mit dem Bearbeiter oder Vorgesetzten diskutieren.

Abbildung 1.4 zeigt das folgende zukünftige Call Center Szenario: Die Fahrerin ist auf dem Weg ins Ausland und ruft ihren Ehemann an. Sie erklärt ihm, daß sie kein Roaming in ihrem Mobiltelefon hat. Der Mann greift über seinen PC auf die Vertragsdaten bei dem Unternehmen zu und stellt eine Verbindung zu einem Bearbeiter her. Jeder der Benutzer kann jetzt durch seinen Cursor, der von beiden gesehen wird, auf die wichtigen Bereiche des Dokuments zeigen und diese hervorheben. Andere Benutzer könnten an der Diskussion teilnehmen, wobei jeder einzelne Teilnehmer zwar auf dasselbe Dokument zugreift, eventuell aber eine andere visuelle Darstellung davon hat. Vorstellbar ist, daß die Fahrerin über eine WAB-Schnittstelle ebenfalls an der Diskussion teilnimmt[5].

Gruppenspiel

Eine der möglichen Anwendungen der neuen UMTS-Technologie könnte die Koordination von Aufgaben zwischen verschiedenen Benutzern sein [223]. Eine solche Koordination könnte an unterschiedlichen Orten spontan entstehen. Beteiligt sind daran in der Regel nur wenige Benutzer (2 bis 4 Personen). Hier ist es vorstellbar, daß die Unternehmen selbst, oder vertrauliche Drittparteien, ihre Dienste über zentrale, für die Öffentlichkeit zugängliche Daten bereitstellen. Diese Daten sind in einem generischen Format gespeichert und können von verschiedenen Geräten über ein wohldefiniertes Protokoll bearbeitet werden. Für die Ausführung von Operationen auf den gemeinsamen Daten werden intuitive Navigationsoperationen bereitgestellt, die von jedem Client ausgeführt werden können.

Dieses Szenario ist eine Implementierung eines Gruppenspiels, das auf der CeBIT 2001 in Hannover als Kooperationsprojekt mit der SIEMENS AG und D2 Vodafone entstanden ist [76, 166]. Das Spiel dient lediglich der Darstellung der Einsatzmöglichkeiten solcher Anwendungen. In dem Spiel können drei Spieler mittels mobiler UMTS-Geräte eine Party organisieren. Sie können zwischen einer bayerischen Folklore-Party, einer Techno-Party oder einer spanischen Fiesta wählen. Zu der gewählten Party muß jeder Spieler das passende Zubehör in einer bestimmten Zeit finden und dann zu dem Treffpunkt zurückgehen.

[4] Aus traditionellen Gr¨unden verwenden wir den Begriff *Finger* (siehe Abschnitt 4.1.1).

[5] Roaming vorausgesetzt!

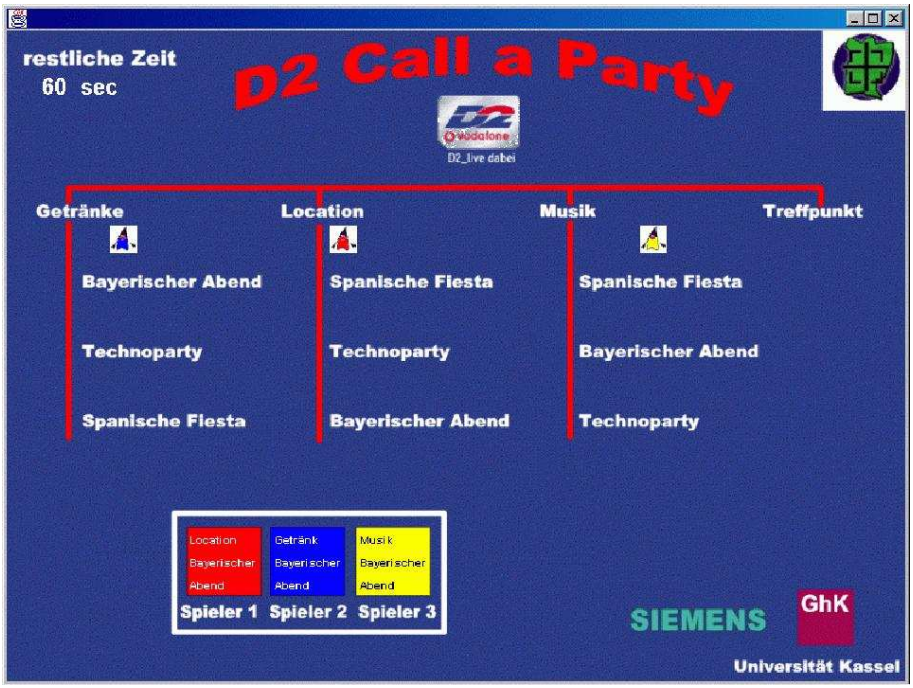


Abbildung 1.5: Anwendungszenario: D2 Party (aus [76] und [166])

Natürlich ist dies ein sehr einfaches Anwendungszenario, aber es verdeutlicht die Einsatzmöglichkeiten solcher Anwendungen und benutzt die wichtigsten Merkmale der synchronen Gruppenarbeit. Abbildung 1.5 zeigt die Serveranzeige, wie sie auf der CeBIT 2001 in Hannover zu sehen war.

Lernsoftware

Bei Lernsoftware existieren zwei Ansätze für die Unterstützung des Lernprozesses: Bei dem ersten wird versucht, durch die Software den Lehrer zu „ersetzen", so daß die Schüler selbstständig den Stoff abarbeiten können. Bei dem anderen Ansatz unterstützt die Lernsoftware die Kommunikation und den Austausch von Informationen zwischen den Schülern [108].

Im allgemein ist Computer-basiertes Lernen eine Anwendungsmöglichkeit von Groupware. Neben der Unterstützung von Kommunikation und dem Austausch von Informationen, können gemeinsame Objekträume einen gemeinsamen Lehrinhalt bilden, den mehrere Schüler gemeinsam abarbeiten können. Awareness-Informationen können hier die gleichzeitige Bearbeitung von diesem Lehrmaterial verstärken, und dadurch mehr Kommunikation zwischen den einzelnen Schülern, aber auch Lehrern, hervorbringen. In [122] stellen die Autoren ein Szenario vor, in dem Schüler und Lehrer gemeinsam durch Navigation auf gemeinsamen Objekträumen und Awareness-Unterstützung Lehrmaterial bearbeiten können[6].

1.3 Problembeschreibung

In diesem Abschnitt werden die wichtigsten Aspekte für die Entwicklung eines generischen Modells für synchrone Gruppenarbeit behandelt. Daraus werden die genauen Teilaufgaben bezüglich dieser Aspekte formuliert. In Kapitel 2 gehen wir auf diese Aspekte im Detail ein und betrachten den momentanen Stand in der CSCW- Literatur. In Kapitel 3 wird das in dieser Arbeit entwickelte Modell und die Lösungen der hier formulierten Teilaufgaben vorgestellt. Anschließend wird in Kapitel 4 und Kapitel 5 die Realisierung des Prototyps erläutert.

Die Implementierung von Anwendungen für die synchrone Gruppenarbeit ist eine sehr komplexe Aufgabe. Viele Anwendungen haben jedoch ähnliche Strukturen, die in einer generischen Form als Funktionenbibliothek für den Anwendungsentwickler zur Verfügung gestellt werden können. Ähnlich einem Datenbankmanagementsystems (DBMS) soll ein CSCW-System eine Schnittstelle nach außen anbieten, die Grundfunktionen von Groupware-Anwendungen bereitstellt. Zusammenfassend legen wir folgende drei Punkte fest:

- Synchrone Groupware-Anwendungen haben gemeinsame Funktionalitäten.
- Diese können durch eine gemeinsame Technologie erbracht werden.
- Die Technologie muß generisch sein [187].

Hauptaufgabe: *Welche Anforderungen soll ein System für die synchrone Gruppenarbeit erfüllen und welche Funktionen soll es bereitstellen. Die folgenden Punkte werden zur Entwicklung des Systems diskutiert* [64, 65]:

- *System-Architektur*
- *Datenmodell, Datenbank + Operationen*
- *Nebenläufigkeitskontrolle*
- *Zugriffskontrolle*
- *Awareness*

System-Architektur

Für CSCW-Systeme existiert kein generelles Schichtenmodell, wie es bei Datenbanksystemen mit dem ANSI-SPARC-Modell [173] der Fall ist. Doch wurde dies von verschiedenen CSCW-Forschern gefordert. Eine wichtige Arbeit zu diesem Thema ist die von *Ellis* und *Wainer* [65] aus dem Jahr 1994. In ihr charakterisieren die Autoren CSCW-Applikationen durch drei Komponenten:

[6] Siehe z. B. `http://www.abitur-online.nrw.de`.

- Die Beschreibung der Objekte und der Operationen auf diesen Objekten.
- Die Beschreibung des dynamischen Teils des Systems, die Organisation der Aktivitäten der Benutzer.
- Die Beschreibung der Schnittstelle zwischen dem System und den Benutzern.

Ellis bekräftigt, daß dieses Modell das System aus Benutzersicht beschreibt, in dem Sinne, daß systeminterne Konzepte nicht betrachtet werden, wie z.B. *replizierte* oder *zentrale* Datenhaltung: „*We must stress that these three models describe the system from its users' point of view*" [65, S. 79]. *Teege* diskutiert dieses Konzept und versucht es zu erweitern [187, S. 23]. Er unterscheidet grundsätzlich zwischen Organisationsmodell, Notation und Speicherung des Organisationsmodells (z. B. in einer Datenbank), Zugriff auf das Organisationsmodell, auf dem Organisationsmodell aufbauende Dienste und der Benutzeroberfläche [187].

Die erste Aufgabe für die Entwicklung eines Systems für die synchrone Gruppenarbeit ist also die Festlegung einer Schichtenarchitektur und Zuordnung der Funktionen zu den verschiedenen Schichten. Die Schnittstellen zwischen den verschiedenen Schichten müssen formuliert werden. Für den Anwendungsentwickler muß eine eindeutige Schnittstelle zur Verfügung gestellt werden, mit deren Funktionen die Erstellung von synchroner Groupware erleichtert wird.

Aufgabe 1: Die Angabe einer Schichtenarchitektur und die Zuordnung der Funktionalitäten des Systems zu den entsprechenden Schichten.

Datenmodell

Bei der synchronen Gruppenarbeit können mehrere Benutzer gleichzeitig den gemeinsamen Informationsraum modifizieren, während andere Benutzer diese Veränderungen sehen. Die Abstraktion und die Speicherung der Daten sowie die Zugriffe darauf wird durch ein sogenanntes Datenmodell behandelt. In der Arbeit von *Greif* [91] wird festgestellt, daß für die Verwaltung von gemeinsam genutzten Daten in Groupware-Systemen der Einsatz von sog. *speziellen Datenbanken* notwendig ist. Diese sind an die Besonderheiten und Anforderungen von Groupware-Systemen angepaßt [7].

Da das System generisch sein soll, müssen die Daten bei verschiedenen Benutzern mit unterschiedlichen Anzeigegeräten auf verschiedene Weisen dargestellt werden können. Dies erfordert ein Datenmodell, das einerseits verschiedene Darstellungsmöglichkeiten der Daten zuläßt, andererseits die Abbildung des Objektraumes auf vorhandene und verbreitete Datenmodelle im Datenbankbereich (Relational, Objektorientiert, Objektrelational) ermöglicht.

[7] In dem Buch von *Teufel* [174] werden diese Anforderungen detailiert diskutiert (Seite 174).

Die unterschiedliche Sichtweise der Darstellung der Daten aus der Anwendersicht (Dokumente) und der Datenbanksicht (Tabellen, Relationen) stellt dabei ein wesentliches Problem dar [17]. Das Datenmodell kann in diesem Sinne als Vermittler zwischen diesen verschiedenen Sichtweisen der Datenrepräsentation dienen. Auf der einen Seite kann es Daten aus verschiedenen Datenbanksystemen abbilden, auf der anderen Seite verschiedene Darstellungsmöglichkeiten auf der Benutzersicht ermöglichen.

Aufgabe 2: Die Anforderungen von synchronen Groupware-Applikationen an das Datenmodell formulieren und daraus das Datenmodell bestimmen.

Kontrolle der Nebenläufigkeit

Wie bei jedem Mehrbenutzersystem müssen Mechanismen gefunden werden, die konkurrierende Zugriffe auf die Daten regeln und die Konsistenz der Daten gewährleisten. Die Nebenläufigkeitskontrolle ist ein Thema, das vor allem bei Datenbanksystemen lange erforscht wurde. Die Unterschiede dieser Problematik bei der synchronen Gruppenarbeit zu der bei Datenbanksystemen sind aber offensichtlich. Der Hauptunterschied liegt darin, daß die Benutzer bei der synchronen Gruppenarbeit die Auswirkungen der Operationen anderer Benutzer sehen, während bei Datenbanksystemen die Isolierung von parallel laufenden Transaktionen als Voraussetzung für die Korrektheit gilt (ACID-Transaktionen). Bei der synchronen Gruppenarbeit sehen die Benutzer u.U. zu bestimmten Zeitabschnitten ein nicht aktuelles Bild der Daten. Die Ausführung von Operationen kann also in manchen Fällen aus einem nicht aktuellen Zustand der Visualisierung heraus erfolgen (dirty read). Weitere Besonderheiten sind: ein höherer Grad an Parallelität und die Integration des Benutzers als Teil von Transaktionen [88].

Aufgabe 3: Ein Kriterium für die Korrektheit angeben. Das Verfahren für die Nebenläufigkeitskontrolle soll aus diesem Kriterium abgeleitet werden.

Zugriffskontrolle

Bei gemeinsamen Datenbeständen müssen Mechanismen entwickelt werden, die Besitzverhältnisse und Benutzerrechte vergeben und Zugriffe der Benutzer auf die Daten entsprechend dieser Rechte kontrollieren [91]. Die Besonderheiten der Zugriffsrechte bei der synchronen Gruppenarbeit müssen auch hier gesondert betrachtet werden [169, 153]. Insbesondere ist bei der synchronen Gruppenarbeit mehr Dynamik bei der Erstellung von Objekten und eine feinere Aufteilung der einzelnen Objekte in kleineren Einheiten zu beobachten. *Ellis* schreibt: „*Groupware systems typically have an expanded set of rights*" [65] und weist darauf hin, daß bei Groupware-Applikationen außer den typischen Lese-, Schreiberechten noch weitere Zugriffsrechte gibt. Da mehrere Personen z. B. gleichzeitig an der Erstellung eines Dokuments beteiligt sein können, sind die Besitzverhältnisse sehr komplex.

Obwohl zu dem System ständig neue Benutzer hinzugefügt werden können, und mehrere Benutzer sich an der Arbeit beteiligen oder die Anwendung verlassen, bleibt die Zugriffskontrolle überwiegend statisch geprägt und unterscheidet sich daher bei der synchronen Gruppenarbeit nicht viel von der Zugriffskontrolle bei anderen Groupware-Applikationen. In der CSCW-Literatur wird das Thema Zugriffsberechtigung im allgemeinen wenig behandelt, jedoch betrachten es *Greif* [91], und *Ellis* [64] als ein zentrales Thema bei der Behandlung von gemeinsamen Informationsräumen.

Aufgabe 4: Ein Konzept für die Zugriffskontrolle muß angegeben werden. Dies soll die Anforderungen der synchronen Gruppenarbeit, insbesondere die Dynamik, die aus der Verfügbarkeit im Web entsteht, berücksichtigen.

Awareness

Ein grundlegendes Merkmal von synchronen CSCW-Applikationen ist die Möglichkeit, zu sehen, welche anderen Benutzer das System benutzen, was sie machen und wo sie gerade agieren. Veränderungen, die sowohl die Objekte des gemeinsamen Informationsraumes, als auch die Benutzerinformation betreffen, werden den anderen Benutzern mitgeteilt. Für die Realisierung von Mechanismen, die diese Informationen filtern und auf der Benutzeroberfläche in einer leicht verständlichen Weise visualisieren, ist ein sogenanntes *Awareness Modell* zu entwickeln. Das Awareness-Modell muß eine passende *Metapher* finden, um diese Informationen auf verständliche Weise darzustellen. Dabei soll das Awareness-Modell feststellen können, welche Information für einen bestimmten Benutzer *wichtig* ist, und dem Benutzer nur diese Information mitteilen. Das Benachrichtigen der Benutzer über alle Änderungen im gemeinsamen Informationsraum wäre ungeeignet [152].

Aufgabe 5: Ein Awareness-Modell muß entwickelt werden. Dieses soll generische Awareness-Informationen liefern, die von jeder Anwendung auf ihre Art visualisiert werden kann.

Kapitel 2

Grundlagen

In diesem Kapitel wird auf die verschiedenen Aufgaben, die für die Implementierung des Groupware-Systems von Interesse sind, im Detail eingegangen. Dabei werden vorhandene Datenmodelle, Awareness-Modelle, Ansätze zur Lösung der Nebenläufigkeit sowie verschiedene Web-Technologien betrachtet und diskutiert.

2.1 Datenmodelle

Die erste Aufgabe bei der Implementierung eines Softwaresystems ist die Modellierung. Speziell geht es um die Abbildung der Gegenstände des zu lösenden Problems auf die Objekte einer Programmiersprache oder eines Datenbanksystems. Dies ist nicht anders bei der Implementierung einer Anwendung für die synchrone Gruppenarbeit. So schreibt *Mariani*: „*Database technology has a vital part to play in collaborative systems by supporting information sharing within a user community.*" [134, S. 376]. Bei der Implementierung eines synchronen Gruppeneditors z. B. müssen große Dokumente in kleinere logische Einheiten aufgeteilt werden, damit mehrere Benutzer gleichzeitig Teile des Dokuments bearbeiten können. Die Operationen des Editors können vom Anwendungsentwickler durch Funktionen des Datenmodells implementiert werden.

Ein System für die synchrone Gruppenarbeit muß daher ein Datenmodell zur Verfügung stellen, auf das sich sowohl die Struktur der Objekte als auch die Operationssemantik der Anwendung abbilden läßt. Erst durch das Festlegen eines Datenmodells und einer *endlichen Menge* aller möglichen Operationen auf die Daten kann ein generisches System für Awareness und die Kontrolle der Nebenläufigkeit implementiert werden.

Dabei meinen wir mit generischen Awareness-Nachrichten die Möglichkeit, auf jede Operation des Datenmodells eine *korrespondierende Awareness-Nachricht* generieren zu können, die von jeder Anwendung auf eigene Art visualisiert werden kann.

Ein **Datenmodell** bestimmt, wie sich Objekte und deren Verhalten, Beziehungen zwischen den Objekten sowie Abläufe von Prozessen der (realen) zu modellierenden Welt in Datenobjekte und Algorithmen der (virtuellen) Rechnerwelt abbilden lassen. In der Literatur (z. B. [40, 141, 130, 102]) hat sich für die Definition von Datenmodellen eine Aufteilung in mehrere Komponenten herauskristallisiert[1]:

- Im **strukturellen Teil** wird beschrieben, wie die Datenobjekte und die Beziehungen zwischen diesen modelliert werden können,
- der **operationale Teil** beschreibt die Manipulation der Datenobjekte, und
- zusätzlich werden die **Integritätsbedingungen** zur Beschreibung der Konsistenz betrachtet.

Die Datenmodelle[2], die sich heute für Datenbanken sowohl im wissenschaftlichen als auch im kommerziellen Bereich durchgesetzt haben, sind das relationale Datenmodell und dessen Nachfolger.

2.1.1 Das relationale Datenmodell

Das relationale Datenmodell [49] hat das hierarchische Datenmodell [198] und das Netzwerk-Datenmodell [186] abgelöst, obwohl diese beiden „Urahnen“ immer noch kommerzielle Bedeutung haben (*legacy systems*: IDS, DMS, IMS). Das Relationale Datenmodell basiert auf der sogenannten Relationalen Algebra. Daten werden in Relationen (Tabellen) aufgeteilt. Eine Relation besteht aus Tupeln, die wiederum atomare, unstrukturierte Objekte enthalten. Jede Relation enthält einen eindeutigen Schlüssel (*Primary Key*). Beziehungen zwischen den Relationen werden über sog. Fremdschlüssel hergestellt (*Foreign Key*). Um die Redundanz der Daten zu minimieren und dadurch auch die Bewahrung der Konsistenz der Daten zu sichern, wird ein Datenbankschema durch die sog. *Normalisierung* eventuell in mehrere Tabellen aufgespalten. Der Zugriff auf die Daten wird vom Datenbanksystem über verschiedene Schnittstellen zur Verfügung gestellt. Die Bearbeitung der Datenbank basiert auf SQL-Abfragen [105, 106].

Einzelne Relationen haben durch die Tabellen mit Zeilen und Spalten eine gut verständliche und leicht implementierbare Visualisierung. Durch Normalisierung geht allerdings der strukturelle Zusammenhang verloren. Durch join gebildete Sichten enthalten dann wieder viel Redundanz, sind unübersichtlich und teuer in der Generierung.

[1] Diese Arbeit bezieht sich haupts¨achlich auf den strukturellen und den operationalen Teil der Datenmodelle.

[2] Mit Datenmodellen sind hier immer Datenbankmodelle gemeint; Datenmodelle gibt es z. B. auch f¨ur Programmiersprachen, wobei dort eher von Programmierparadigmen und Typsystemen gesprochen wird.

Die Implementierung von grafischen Schnittstellen, die eine einfache und intuitive Navigation und Bearbeitung der Daten von unerfahrenen Benutzern ermöglichen, ist im allgemein eine schwierige Aufgabe. Für die Visualisierung zusammenhängender Informationen müssen die Daten aus verschiedenen Tabellen zusammengestellt werden. Schreiboperationen auf ein Dokument könnten die Modifikation mehrerer Tabellen verursachen. Bei der synchronen Gruppenarbeit werden diese Probleme noch deutlicher. Durch Awareness muß die Visualisierung des Dokuments in kurzen Zeitabständen aktualisiert werden, um eine konsistente Visualisierung zu bewahren. Schreiboperationen, die parallel von verschiedenen Benutzern ausgeführt werden, können in Konflikt stehen. Je nach verwendetem Verfahren für die Kontrolle der Nebenläufigkeit (Pessimistisches, Optimistisches Verfahren) könnte dies die Sperrung von mehreren Objekten derselben Tabelle erzwingen und dadurch andere Teilnehmer an der Arbeit hindern, oder bei Eintritt von Konflikten bei optimistischen Verfahren zu lästigen visuellen Rollbacks führen.

Das relationale Datenmodell soll gegenüber den post-relationalen Datenmodellen als einfaches Datenmodell abgegrenzt werden. Es wird deshalb als „einfach“ bezeichnet, weil es in seiner ursprünglichen Form nur einfache Datentypen (elementare oder atomare, alpha-numerische Datentypen wie Zahlen, Daten, Währungen, Zeichenketten, usw.) und fest vorgegebene Strukturierungen (Relationen sind Mengen von Tupeln) erlaubt. Im Gegensatz dazu erlauben komplexe Datenmodelle die Strukturierung von komplexen Objekten der realen Welt und deren Beziehungen in einer „natürlichen“ Art und Weise. Als Nachfolger des relationalen Datenmodells, d. h. als post-relationale Datenmodelle, werden hier das NF^2-Datenmodell und dessen Erweiterung, das eNF^2-Datenmodell[3], sowie objekt-orientierte Datenmodelle und die Mischform aus relationalem Datenmodell und objekt-orientierten Konzepten, das objekt-relationale Datenmodell angesehen.

Die Hervorhebung der komplexen Datenobjekte beruht auf der Beobachtung, daß in fast allen Anwendungen die modellierten Entitäten strukturiert sind [44]. Bei nichtkommerziellen Anwendungen, etwa in Wissenschaft und Technik, treten komplexe Strukturierungen sogar besonders häufig auf. Mit dem Aufkommen des Internets entstanden andere Arten von Informationen, die man auch in Datenbanken verwalten möchte. Die Speicherung von solchen semi-strukturierten Dokumenten (HTML, XML usw.) in relationalen Datenbanken erweist sich als äußerst schwierig und ineffizient [69]. Die einfachen Mittel des relationalen Datenmodells sind eher geeignet, kaufmännische und administrative Anwendungen adäquat zu modellieren und in RDBMSs effizient zu implementieren.

[3] Da das eNF^2-Datenmodell, ESCHER sowie TclDB Teile unserer Forschung sind, werden diese erst in Kapitel 4 behandelt, im Gegensatz zu den anderen Datenmodellen und XML.

2.1.2 Objektorientierte Modelle

Das Aufkommen der objekt-orientierten Programmiersprachen mit Merkmalen wie Vererbung und Kapselung bildete die Basis für die objekt-orientierten Datenmodelle[4] in der Datenbankwelt. Sie stellten zunächst keine evolutionäre Weiterentwicklung des relationalen Datenmodells dar, sondern waren ein revolutionärer Neuanfang und erlebten in der Wissenschaft eine geradezu stürmische Entwicklung.

Es zeigte sich jedoch, daß die als Erweiterungen objekt-orientierter Programmiersprachen realisierten Systeme den Anforderungen an eine Datenbank nicht gerecht wurden, da sie grundlegende Konzepte nicht verwirklichten (z. B. fehlten meist Transaktionskonzepte, nicht-prozedurale Anfragesprachen, Meta-Daten-Verwaltung, Sichten, u. a.). Sie werden daher, etwas abfällig, oft auch als *„persistent storage manager“* bezeichnet [117, S. 5]. Die Forschung ist heute auf Datenbanksysteme gerichtet, die die Konzepte von Datenbanken (Transaktionen, Nebenläufigkeit, physische Datenunabhängigkeit, Dateiorganisation und Zugriffspfade, Optimierbarkeit usw.) mit den Konzepten der Objekt-Orientierung (Kapselung, Vererbung, Objektidentität, strukturierte und verschachtelte Objekte) verschmelzen, und nicht das eine um das andere erweitern. Dadurch soll eine bessere Integration der beiden Ansätze erreicht werden.

Ein grundlegendes Konzept aller objekt-orientierten Datenmodelle ist die Unterstützung komplexer Objekte (auch „strukturelle Objekt-Orientierung“ genannt [57]). Betrachtet man die strukturelle Komponente objekt-orientierter Datenmodelle genauer, so stellt man fest, daß sie im Wesentlichen nicht über die Erweiterung des relationalen Datenmodells durch das eNF^2-Datenmodell hinausgehen (siehe Abschnitt 3.3.1). Im Gegenteil wird insbesondere die Konstruktion von Kollektionen unzureichend unterstützt, indem nur Listen von Komponentenobjekten zugelassen werden [120, S. 243 ff.]. Diese Unzulänglichkeit resultiert eindeutig aus dem Ursprung in den objekt-orientierten Programmiersprachen, die selbst keine beliebigen Kollektionen erlauben.

Von der Object Database Management Group (ODMG) wurde ein Objektmodell für Objektdatenbanksysteme als auf dem Kernobjektmodell aufbauendes Profil (*ODBMS profile*) entwickelt. Dieses floß in den Object Database Standard ODMG-93 in der Version 1.2 ein [46] und findet sich auch in der aktuellsten Version ODMG-2.0 [47]. Es erlaubt die Konstruktion komplexer Typen durch Konstruktoren (dort *generators* genannt) für Kollektionen (*set*, *bag*, *list* und *array*) und Strukturen (*tuple*). Letztere, die Strukturen, können jedoch nur als sog. „literale Typen“ (*literal types*) definiert werden und besitzen somit keine Objektidentität.

Die objekt-orientierte Programmiersprache Java, die nicht nur wegen ihrer Plattformunabhängigkeit immer mehr an Bedeutung gewinnt, wird auch als Host-Sprache für

[4] Bei objekt-orientierten Datenmodellen ist der Plural tats¨achlich angemessen, da, anders als beim relationalen Datenmodell, eine F¨ulle von z. T. erheblich differierenden Vorschl¨agen existieren. Es gibt einige Ans¨atze, Gemeinsamkeiten herzustellen und Standards zu etablieren [26, 45, 46].

objekt-orientierte Datenbanken eingesetzt. Die Java-Anwendungsschnittstelle für Datenbanken (*Java database connectivity*, JDBC) dient dem Verbindungsaufbau mit Datenbanken, Versenden von SQL-Befehlen und Bearbeiten der Anfrageergebnisse [96]. Mit SQLJ [28] sollen SQL und Java enger verbunden werden. Dabei werden die Möglichkeiten von SQL3 zur Definition von komplexen Typen mittels orthogonal anwendbarer Typkonstruktoren erweitert, z. B. indem Java-Klassen zur Typdefinition verwendet werden können [103, S. 533]. In ODMG-2.0 wurde ebenfalls ein Standard für Persistenz in Java (*Java persistence standard*) aufgenommen.

2.1.3 Das objekt-relationale Datenmodell

Die objekt-relationalen Datenbanken sind die konsequente Fortführung der Entwicklung von erweitert-relationalen Datenbanksystemen unter Einbeziehung einiger Konzepte der Objekt-Orientierung (objekt-relationale Datenbanken werden sogar als Umbenennung der erweitert-relationalen Datenbanken bezeichnet [103, S. 533]). Viele sehen als Grundlage der post-relationalen Datenbanktechnologie eine Vereinigung von relationalen und objekt-orientierten Systemen [117, S. 5 ff.].

Stonebraker ordnet objekt-relationalen DBMS vier Hauptmerkmale zu [176, S. xii]:

- Erweiterbarkeit der Basistypen,
- komplexe Objekte,
- Vererbung und
- ein Produktionsregelsystem (*production rule system*).

Die Erweiterbarkeit der Basistypen muß integraler Bestandteil der objekt-relationalen Datenbank sein und sich in dessen Konzepte einfügen, d. h. ein neu definierter Basistyp muß in Bezug auf Ein- und Ausgabe, Optimierung usw. den eingebauten Basistypen gegenüber gleichwertig sein. Für die neuen Basistypen müssen vom Benutzer Funktionen sowie geeignete Zugriffsverfahren definiert werden können [176, S. 21 ff.].

Um komplexe Objekte zu konstruieren, werden die folgenden Typkonstruktoren als notwendig erachtet[5]:

- Tupel (*record*),
- Kollektion (*set*) und
- Zeiger (*reference*).

Weitere nützliche Typkonstruktoren sind solche für Listen (*lists*), Stapel (*stacks*), Warteschlangen (*queues*) und Vektoren (*arrays*). Komplexe Objekte sollen Parameter und Resultatwerte von benutzerdefinierten Funktionen und Argumente von Operatoren sein können. Für komplexe Objekte können Pfadausdrücke (in [176] „kaskadierte Punktnotation“, *cascaded dot notation*, genannt) benutzt werden, um Komponenten zu spezifizieren.

[5] Die Typkonstruktoren müssen orthogonal anwendbar sein.

Die Erweiterbarkeit von Basistypen wird von der Definition komplexer Typen aus folgenden Gründen unterschieden:

- Typen ohne innere Struktur, wie z. B. ein Typ für positive, ganze Zahlen, sollen direkt als Basistyp, nicht über den Umweg eines komplexen Objekts, definierbar sein.
- Instanzen komplexer Typen ist eine Objektidentität zugeordnet. Sie können daher referenziert werden, verursachen aber einen zusätzlichen Speicheraufwand.
- Basistypen mit definierten Vergleichsoperatoren können durch traditionelle Zugriffspfade unterstützt werden.

Die Bedeutung der objekt-relationalen Technologie wird auch durch die Aktivitäten der ODMG unterstrichen, die 1998 eine neue Arbeitsgruppe zum Thema „Objekt-relationale (OR) Abbildung“ (*object/relational (OR) mapping*) ins Leben gerufen hat.

2.2 Groupware, Datenbanksysteme und das Web

Die Verbreitung und die Kommerzialisierung des World Wide Web in den letzten Jahren hat Web-Browser von einfachen Programmen zum Darstellen von statischen HTML-Seiten zu generischen Frontend-Programmen für verschiedene Anwendungen entwickelt. Sie stellen daher eine attraktive Möglichkeit dar, synchrone Gruppenarbeit über Web-Browser zugänglich zu machen. Neue Anwendungsbereiche, aber auch Herausforderungen bei der Entwicklung von Groupware-Anwendungen, sind dadurch entstanden [23, 197, 59].

Da große Mengen an Daten bereits in Datenbanken gespeichert sind, bedeutete diese Entwicklung für Datenbanksysteme eine Verbindung zu dieser neuen Art der Informationsverbreitung zu schaffen. In Beiträgen von *Benn* und *Gringer* im Informatik Spektrum [31], sowie von *Loeser* auf der BTW 97 [128], wird auf das gespaltene Verhältnis zwischen Datenbanken und World Wide Web aufmerksam gemacht. Demnach begannen kommerzielle DBMS-Anbieter nach einer Anfangsphase, in der Datenbanken und das Web ganz getrennte Wege gingen (*David DeWitt*: „*database systems as road kill on the Information Highway*” [56]), Web Server mit DBMS-Kopplung auf den Markt zu bringen. Diese generieren HTML-Seiten mit Datenbankinhalten ad hoc als Resultat einer Abfrage und versenden diese Seiten [132]. Trotzdem ist auch diese Kopplung überwiegend statisch. Erst durch den von *Loeser* [128] als dritte Generation bezeichneten Einsatz aktiver Elemente auf der Benutzerseite im Browser, etwa in Form von Java-Applets, entsteht eine dynamische, interaktive Verbindung. Die natürliche Folgeentwicklung ist die synchrone Kooperation räumlich verteilter Benutzer (Synchrone Gruppenarbeit über das Internet) mit einem Web-Browser als Benutzerschnittstelle.

Trevor und *Koch* schreiben z. B.:„*Almost any new CSCW system has some type of Web interface, whether it is purpose built for the Web, such as BSCW (Bentley et al. 1997), or provides generic access to features in non-Web applications, as with Poli-Web (Freund, 1996). Even existing systems like Lotus Notes with Domino have been extended to present their interfaces as Web pages.* " [S. 65][197].

Obwohl mehrere web-basierte Groupware-Produkte in den letzten Jahren erschienen sind (*QuickPlace* von *Lotus Development*, *eRoom* von *Instinctive Technology*, *Web-Groups* von *Punch Networks*), ist der erhofte Durchbruch ausgeblieben. *Dix* diskutiert die Problematik von Web-basierten Groupware-Applikationen und versucht die Gründe für den Mißerfolg dieser Anwendungen aufzustellen [59]. Die Zustandslosigkeit des HTTP-Protokolls kann allgemein als das grundlegende Problem für die Entwicklung von Web-basierten Applikationen betrachtet werden [199, 197]. Für synchrone Groupware-Anwendungen ist dieses Problem besonders wichtig, da der Austausch von Informationen zwischen Server und Applikation über die ganze Dauer der Anwendung stattfindet [197]. Hier müssen Mechanismen gefunden werden, die die Speicherung von Informationen sowohl bei dem Client als auch bei dem Server ermöglichen und diese über die Dauer einer HTTP-Anforderung behalten.

Ein zweites wichtiges Problem ist das Fehlen von Sprachen für den Austausch von strukturierten Daten. Die bisher verwendete Sprache im Internet, HTML, besteht im Prinzip aus logischen Beschreibungselementen (Tags), weicht aber stark von diesem Prinzip ab. Die weiteren Entwicklungen und Erweiterungen in HTML haben diese Sprache mehr zu einer Visualisierungssprache als einer Sprache zur logischen Beschreibung des Inhalts eines Dokuments gemacht. Diese Problematik stellt ein großes Hindernis für die Entwicklung von Standardprotokollen für webbasierte synchrone Groupware-Anwendungen dar.

2.2.1 Web Technologien

Kurz nach der Entstehung des World Wide Web und dem Erscheinen der ersten Browser 1992 wurde der Wunsch nach mehr Interaktionsmöglichkeiten und die Benutzung von Browsern für die Bereitstellung von Anwendungen von verschiedenen Seiten geäußert. Anders als in der ursprünglichen Idee für das HTTP-Protokoll, hat die Industrie, aber auch verschiedene Forschungseinrichtungen, die enormen Möglichkeiten erkannt, die über das World Wide Web durch die Interaktion des Benutzers entstehen könnten. Seit dieser Zeit sind für diesen Zweck verschiedene Technologien entwickelt worden. In diesem Abschnitt betrachten wir die verschiedenen Techniken für die Entwicklung von Web-basierten Anwendungen und die Möglichkeiten, diese für die Entwicklung von synchronen Groupware-Applikationen einzusetzen [199].

HTTP ist ein einfaches textbasiertes Protokoll. Wird eine Seite von einem Client (Web-Browser) gefordert, so wird eine Verbindung zum Server hergestellt, das Dokument an den Client übertragen und die Verbindung wieder beendet. Zur Identifizierung von

Dokumenten wird ein Adressierungsschema eingesetzt. Ein Dokument wird durch ein URI (*Uniform Resource Locator*) identifiziert [33]. Die meisten Dokumente werden mit der Sprache HTML [205] abgelegt [199].

Um Interaktivität in diesem einfachen und zustandslosen Verhalten zu erreichen, sind Erweiterungen entweder auf der Client-, oder Serverseite notwendig. *Bentley* [32] unterteilt Web-basierte Groupware-Anwendungen in vier Kategorien:

- *Purely W3-based*: Erweiterung nur durch serverseitige CGI-Skripte.
- *Customised servers*: Einsatz von speziellen Servererweiterungen.
- *Customised clients*: Einsatz von speziellen Clienterweiterungen.
- *Web-related*: Web-Schnittstelle aber geringe Web-Unterstützung.

Allgemein kann man zwischen *clientseitigen* und *serverseitigen* Web-Technologien unterscheiden.

Clientseitige Technologien

Java Applets

Die Programmiersprache Java wurde im Jahr 1995 von der Firma SUN eingeführt [82, 83]. Java ist eine plattformunabhängige Programmiersprache, die von den Funktionen eines Betriebssystems abstrahiert und dadurch die Ausführung von Java-Programmen, durch einen Web-Browser, sog. Java-Applets, ermöglicht. Java-Applets werden als sogenannter Byte-Code über das HTTP-Protokoll vom HTTP-Server zum Browser übertragen. Java ist eine mächtige Programmiersprache, die Funktionen zum Erstellen von komplexen grafischen Oberflächen, Netzwerkprogrammierung usw. zur Verfügung stellt. Die Entwicklung von komplexen Web-basierten Anwendungen wird dadurch ermöglicht. Ein großer Vorteil von Java-Applets für die Entwicklung von Web-basierten Groupware Anwendungen ist die Möglichkeit des Aufbaus einer *dauerhaften Netzwerkverbindung* zu dem Rechner, aus dem der Code kommt. Damit wird die Zustandslosigkeit des HTTP-Protokolls übergangen. Mit entsprechenden Browser-Plugins können auch andere Sprachen eingesetzt werden, so können z.B Tcl-Skripte, sog. *Tclets*, von Netscape genauso wie Applets interpretiert werden [100]. Java hat sich jedoch als Standardsprache etabliert und genießt daher breitere Unterstützung von allen gängigen Browsern.

Skriptsprachen und Javascript

Die Verwendung von Skriptsprachen auf Clientseite ermöglicht einem Browser die Ausführung von bestimmten Operationen während des Ladens des Dokuments oder nach dem Auftreten von bestimmten Ereignissen. Solche Skripte werden entweder direkt in einem HTML-Dokument oder in separaten Dateien gespeichert. Ein HTML-Dokument wird durch ein Objektmodell (*Document Object Model*, kurz DOM) in

Objekte der Programmiersprache beschrieben, und mittels einiger Funktionen der Programmiersprache dynamisch veränderbar gemacht. So tritt die Verwendung von Skripten oft in Verbindung mit *dynamischen HTML* auf [175]. Einige der Anwendungen von clientseitigen Skripten sind die Überprüfung von Benutzereingaben, und die Erzeugung von kleinen Animationen, die die Benutzerinteraktion mit einer HTML-Seite ermöglichen. Daher werden solche Skripte oft als Hilfsmittel für serverseitige CGI-Skripte verwandt. Die Sprache Javascript wird oft für clientseitige Skripte eingesetzt und wird von den meisten Browsern unterstützt. Der W3C legt aber keine bestimmte Programmiersprache für dieses Zweck fest [199]. So werden oft VB-Skript und Tcl für dieses Zweck benutzt.

Serverseitige Technologien

CGI

Eine der ältesten serverseitigen Technologien ist das *Common Gateway Interface* (CGI) [135]. Diese Technologie ermöglicht einem Web-Client die Ausführung von Programmen (meistens Skripten) auf der Serverseite. Die Ausgabe dieser Skripte wird als HTML-Dokument[6] an den Client übertragen. Auf dieser Weise ist diese Technologie für den Client transparent. Beim Anfordern eines Dokuments, das durch die Serverkonfiguration als ein ausführbares Skript markiert ist, startet der Server einen Interpreter, der dieses Skript ausführt. Dieser leitet die Ausgabe des Skripts an den Client weiter. Die Eingabe des Skripts wird entweder über globale Variablen oder von der Standardeingabe vom Web-Client an das Skript übergeben. Die Programmiersprache wird durch CGI nicht festgelegt, jedoch haben sich einige Programmiersprachen in diesem Bereich etabliert. Meistens werden Perl und Tcl für die Erstellung von CGI-Skripten verwendet. Der BSCW-Server benutzt CGI als Basistechnologie für die Implementierung der Erweiterungen des Web-Servers [32].

Ein großer Nachteil dieser Technik ist das Starten eines neuen Prozesses bei jedem Anfordern des Dokuments. Dieser Nachteil wird dadurch aufgehoben, daß der Web-Server die Interpreter für die Ausführung dieser Skripte bei dem ersten Auffordern startet und sie zur Ausführung weiterer Skripte wieder verwendet (*FastCGI*) [41]. Eine andere ähnliche Technik ist die Verwendung von Server-Bibliotheken (*Server Side Includes*), die zur Laufzeit ausgeführt werden können und denselben Adreßraum des Servers verwendet.

Servlets

Mit Servlets bezeichnet man eine in Java realisierte Server-API, die die Ausführung von serverseitigen Java-Programmen ermöglicht. Durch den objektorientierten Ansatz

[6] Die Ausgabe des Skriptes muß nicht ein HTML-Dokument sein. Das Skript kann beliebige Ausgabe, wie GIF-Bilder, Javascript oder PDF produzieren, am gebr¨auchlichsten ist jedoch eine HTML-Ausgabe.

stehen dem Programmierer verschiedene Basisklassen, die die Entwicklung solcher Anwendungen erleichtern, zur Verfügung. Daher bieten Servlets eine mögliche Alternative zu CGI und Server Side Includes, bauen aber auf demselben Prinzip auf.

2.2.2 Netzwerktechnologien

Im Zuge der Evolution der Netzwerkprogrammierung hat sich die einfache Aufteilung in Client-Server Applikationen zu komplexeren Architekturen entwickelt. So übernehmen Programme manchmal gleichzeitig beide Funktionen. Die Weiterentwicklung führt zu den sogenannten verteilten Anwendungen, die - möglicherweise auch gegenseitig - Dienste anderer Anwendungen für die Erledigung eigener Aufgaben nutzen. Bei der verteilten Programmierung ist die Aufteilung in Client-, und Server-Programme nicht vorhanden, stattdessen rufen Programme, die auf verschiedenen Plattformen laufen und die mit verschiedenen Programmiersprachen implementiert sind, Methoden von entfernten Objekten auf, und warten auf die Rückgabewerte dieser Methoden. Hier müssen also andere Mechanismen eingesetzt werden, die den Aufruf von Diensten bei entfernten Prozessen ermöglichen. Dazu sind verschiedene Technologien vorhanden.

RPC [172] (*Remote Procedure Call*) ist eine Technologie, die von Sun entwickelt wurde. Sie ermöglicht einen Prozeß, Prozeduren auf entfernten Maschinen auszuführen. Dabei können die Programme auf unterschiedlichen Plattformen laufen und sogar mit unterschiedlichen Sprachen implementiert sein. XML-RPC [228] ist eine neue XML-Technologie. Diese verwendet den HTTP-Protokoll zur Übertragung, und definiert eine XML-basierte Sprache zur Formulierung von entfernten Prozeduraufrufen.

CORBA (*Common Object Request Broker Architectur*) [203] stellt eine Architektur zur Entwicklung von verteilten Anwendungen zur Verfügung. Sie enthält eine standardisierte Sprache zur Definition von Schnittstellen (IDL, *Interface Definition Language*), mit der Objekte in einer sprachunabhängigen Form deklariert, und dann durch einen Übersetzer in verschiedene Programmiersprachen übersetzt werden können. Ein Programm (Dienst) stellt die Dienste seiner Objekte anderen Anwendungen zur Verfügung. Objekte werden in CORBA über sog. Referenzen identifiziert (Inter-ORB Protokoll, IIOP) und werden daher nur über Referenzen übergeben (*Call by Reference*). Die Übergabe der Objekte selbst ist nicht möglich. Über die Referenz zu einem Objekt können entfernte Programme die Methoden dieses Objekts aufrufen. Der grundlegende Bestandteil von CORBA bildet der *Object Request Broker* (ORB). Dieser erleichtert die Kommunikation zwischen den Objekten. Hierzu werden eine Reihe von Leistungsmerkmalen bereitgestellt (Objekte über Referenzen auffinden, Übertragung des Formats von Parametern usw.). CORBA bietet weiterhin Tools zur Erzeugung von Code für die Kommunikation zwischen den Anwendungen, sog. *Client-Stubs* und *Server-Skeltons*, die aus der IDL-Definition erzeugt werden.

Die RMI-Schnittstelle (*Remnote Method Invocation* [180]) gestattet Java-Objekten die Ausführung von Methoden von Objekten auf einer anderen Java-Maschine. Dazu definiert ein Objekt eine Schnittstelle (sog. *Remote Interface*), die angibt, welche Methoden von anderen Objekten aufgerufen werden können. Die Einschränkung von RMI auf Java grenzt allerdings das Einsatzgebiet von RMI stark ein.

2.2.3 XML (eXtensible Markup Language)

Die Entscheidung XML für die Definition der Beschreibungssprache zwischen dem Groupware-Server und dem Client wurde aus mehreren Gründen gefällt (siehe Abschnitt 3.3). Da in der Implementierung des Prototyps XML und ihre verwandten Technologien eingesetzt werden, wird hier ein Überblick über XML gegeben[7].

XML ist eine Metabeschreibungssprache, mit der verschiedene Markup-Sprachen definiert werden können. Sie stellt eine Basistechnologie für die Definition der Struktur von Markup-Sprachen wie HTML, SVG [213], MathML [212] usw. zur Verfügung. XML kommt in zwei Hauptanwendungsgebiete zum Einsatz: als Basistechnologie für die Entwicklung von Markup-Sprachen sowie als Dokumentenaustauschsprache.

Für die Entwicklung von Client-Server-Applikationen ist der Austausch von Daten notwendig. Zur Bearbeitung der Daten durch ein Programm müssen die Daten in einer internen Darstellung vorliegen. Für den Austausch müssen sie allerdings zunächst in ein plattform-unabhängiges Format umgewandelt werden. Meistens werden dazu textbasierte Protokolle definiert, die die Struktur der Nachrichten festlegen (HTTP, FTP, SMTP). Diese einfache Art des Nachrichtenaustauschs wird sehr erfolgreich eingesetzt. Nachrichten müssen dabei auf syntaktische Korrektheit überprüft werden. Hier fehlt eine standardisierte Sprache, mit der man solche Protokolle beschreiben kann und die den Übersetzungsprozeß zwischen der internen und externen Darstellung vereinheitlichen. Genau diese Aufgabe kann durch den Einsatz von XML gelöst werden.

Die meisten im Internet verfügbaren Informationen liegen in Textform vor [17]. Der Wunsch nach einer Sprache für den Austausch dieser Informationen zwischen den verschiedenen Anwendungen und vor allem zwischen den verschiedenen Sichtweisen und Interpretationen dieser Daten wurde mit dem Wachsen des Internets immer größer. Diese Wünsche sind mit dem Einfluß von unterschiedlichen Richtungen (Datenbankwelt, Design und Visualisierungswelt, Netzwerkprogrammierung) in XML eingeflossen.

Seit der Einführung von XML Ende der 90-er Jahre ist der Interesse an XML, und ihre Akzeptanz in den verschiedensten Bereichen stark gestiegen [8]. Die wichtigsten Gründe für diesen Erfolg können folgendermaßen zusammengefaßt werden [155]:

[7] F¨ur mehr Information siehe [99, 155, 17] sowie die XML-Standards bei der W3C [210, 215, 204, 206, 216].

- XML ist ein offener Standard. Verschiedene Richtungen haben an die Entwicklung mitbewirkt, so daß die Einsatzbereiche von XML sehr groß und unterschiedlich sind.
- Die Definition der Struktur der Daten ist frei wählbar. So können kleine Sprachen für bestimmte Aufgaben maßgeschneidert definiert werden.
- XML ist textbasiert und hat eine einfache und klare Syntax, so daß XML-Dokumente sowohl von menschlichen Lesern, als auch von Maschinen einfach zu lesen und zu verarbeiten sind.
- Für die Bearbeitung von XML-Dokumenten existieren verschiedene sprach- und plattform-unabhängigen Schnittstellen (DOM, SAX). Dies vereinfacht den Austausch von Daten zwischen Anwendungen.
- XML trennt den Inhalt von der Visualisierung eines Dokumentes. Durch Stylesheets können XML-Dokumente in verschiedene Darstellungsformate (HTML, SVG, PDF, Formatting Objects) transformiert werden. Dokumente können für verschiedene Zwecke unterschiedlich visualisiert werden. Dies macht XML attraktiv für die Veröffentlichung von Informationen.

Zu XML gehört eine ganze Breite von Sprachen und Standards zur Schemadefinition, Formulierung von Abfragen, Navigation, Visualisierung usw. Im folgenden wird in kürze auf die wichtigsten XML-Technologien eingegangen.

DTD und XML Schema: Zur Definition der Struktur und Festlegung der Elemente und der Regeln einer Sprache in XML werden sog. *Data Type Definition* DTD eingesetzt [210]. Diese stammen ursprünglich aus der SGML-Welt [80]. DTD-Dokumente sind selbst keine XML-Dokumente sondern beschreiben durch eine einfache Syntax die erlaubten Elemente und die Struktur eines Dokuments. Da die Beschreibungsfähigkeit von DTD zu schwach ist (nur Textelemente als atomare Einheit, keine Vererbung usw.), wurde eine weitere Schemasprache verabschiedet: XML Schema [215]. XML Schema Dokumente sind selbst XML-Dokumente und beinhalten reichere Sprachelemente zur Beschreibung einer Sprache. Insbesondere werden Vererbung, präzisere Einschränkung der Elemente und neue Datentypen wie Zahlen, Währung, Datum etc. eingeführt.

XSL: XSL ist ein Standard für die Darstellung von XML-Dokumenten. XSL besteht aus drei Komponenten [204]: mit der *eXtensible Stylesheet Transformation Language* XSLT [207] können XML-Dokumente in andere XML-Dokumenten transformiert werden. Dabei werden Regeln (*Templates*) für die XML-Elemente angegeben und Anweisungen spezifiziert, die bei der Bearbeitung dieser Elemente ausgeführt werden

[8] Von dem ersten bis zum dritten Quartal des Jahres 1998 hat sich der Einsatz von XML bei IT-Branchen in den USA von 1% auf 16% gesteigert (Zona Research) [143]

(siehe Beispiele in Anhang B). Mit Hilfe von XSLT kann man verschiedene Versionen aus demselben XML-Dokument erzeugen. Der zweite Teil von XSL ist XPath wird weiter unten besprochen. Der dritte Teil von XSL sind *Formatting Objects* (fo). Sie bilden eine ausgereifte Desktop-Sprache, die eine genaue Darstellung eines Dokuments beschreibt. Sie ist daher vergleichbar mit Sprachen wie PDF. Es existieren bis jetzt keine Browser, die Formatting Objects anzeigen können, jedoch existieren schon Tools, die PDF aus Formatting Objects erzeugen können [9].

Die DOM-Schnittstelle: Um den Inhalt eines XML-Dokuments zu lesen muß man das Dokument parsen und die Informationseinheiten davon gewinnen. Zu diesem Zweck existieren Standardschnittstellen, die Programmen ermöglichen, auf eine einheitliche Weise Inhalte von XML-Dokumenten zu lesen und zu verarbeiten. Die bekanntesten Schnittstellen sind: das *Document Object Model* (DOM), und die „*Simple API for XML*" (SAX)[10]. Das Document Object Model [208] wurde ursprünglich für einheitliche Browser-übergreifende Darstellung von HTML-Dokumenten in Java und Javascript entwickelt und wurde für XML übernommen. DOM ist eine sprachunabhängige Baumdarstellung eines XML-Dokuments. Die DOM-Schnittstelle ist in mehreren Interfaces in IDL (*Interface Definition Language*) definiert. Die Implementierung dieser Interfaces bleibt dem Programmierer überlassen. Mithilfe der DOM-Schnittstelle können Anwendungen XML-Dokumente dynamisch verändern und einen wahlfreien, nicht sequentiellen, Zugriff auf den Inhalt des Dokuments haben.

XPath und XQuery: XQuery [216] hat sich als Abfragesprache durchgesetzt und wurde von der W3C als Standardvorschalg verabschiedet. Dabei ist diese Sprache aus mehreren Vorschlägen, die aus verschiedenen Unternehmen und Forschungsinstituten kamen, entstanden. XPath ist an sich eine eigene Pfadsprache zum identifizieren von Elementen in XML-Dokumenten. Dabei werden Teile von XPath z. B. in XSLT benutzt, um Templates auf bestimmte Elemente oder Attribute im Dokument anwenden zu können. XPath wird aber auch in anderen Sprachen benutzt, z. B. in XLink, wo ein XPath-Ausdruck bestimmte Elemente in einem Zieldokument identifizieren soll.

XML und Datenbanken

Um Daten, die in Datenbanken gespeichert sind, zu veröffentlichen werden diese meistens in HTML-Dokumente exportiert. Das Gewinnen von Informationen aus einem HTML-Dokument ist allerdings sehr schwierig, da die Elemente keine semantische Auszeichnung erfahren. Ähnlich ist das Problem, aus HTML-Dokumenten interne Objekte in einer objektorientierten Sprache zu erzeugen [69]. Hier ist also das Problem, ein geeignetes Datenaustauschformat zu definieren, das den Austausch von Daten zwischen verschiedenen internen Darstellungen derselben Daten ermöglicht.

[9] Siehe z. B. den FOP-Prozessor `http://xml.apache.org/fop`.

[10] JDom ist eine weitere Schnittstelle, die eine ¨ahnliche Zugriffsm¨oglichkeit wie DOM bietet, jedoch eine andere interne Darstellung benutzt [136].

Abiteboul gliedert die unterschiedliche Darstellung der Daten in drei „Kulturen": „*the everything-is-a-document, the everything-is-an-object, the-everything-is-a-relation culture*". XML entwickelt sich als Bindeglied zwischen diesen drei Kulturen [17]. Auf der einen Seite können Austauschdaten mit einer beliebigen Struktur definiert werden und damit deren Inhalt mit einer semantischen Bedeutung versehen werden, auf der anderen Seite können XML-Dokumente auf einer einheitlichen Weise durch Standardbibliotheken behandelt werden. Die Verwendung von XML in diesem Sinne kann hauptsächlich in zwei Kategorien eingeteilt werden:

Speichern von XML-Dokumenten in vorhandenen Datenbanksystemen: XML findet ihre Verwendung unter anderem als Austauschsprache zwischen verschiedenen Anwendungen. Eine neue Aufgabe, die dadurch entsteht, ist die Generierung von XML-Dokumenten aus vorhandenen Datenbeständen und der Export von XML-Dokumenten in andere Formate.

Am Forschungsinstitut *INRIA* wurde ein Algorithmus vorgestellt, der mit einem generischen relationalen Schema beliebige XML-Dokumente in einer relationalen Datenbank abspeichert [133]. Abfragen auf XML-Dokumente (XQuery) werden anhand dieses Algorithmus in SQL-Abfragen umgewandelt. Das Ergebnis der SQL-Abfragen wird wieder in XML transformiert und an die Applikation zurückgeliefert.

Das Abbilden von XML-Dokumenten auf das relationale Datenmodell erweist sich allerdings als äußerst schwierig und ineffizient [69]. Die extrem unterschiedliche Darstellung der Daten von XML-Dokumenten und dem relationalen Datenmodell erzwingen die Erzeugung von sehr komplexen Abfragen mit verschiedenen Joins auf die relationalen Tabellen bei jeder einfachen Abfrage oder Modifikation auf das XML-Dokument. Dieser Ansatz eignet sich daher nur für die Bereitstellung von vorhandenen relationalen Datenbanken in Form von XML-Dokumente, nicht aber für eine allgemeine dauerhafte Lösung für die persistente und effiziente Speicherung von XML-Daten. Der andere Ansatz wäre die Speicherung von ganzen Dokumenten als atomare Einheit, was einfache Abfragen auf XML-Dokumenten (XPath, XQuery) verhindern würde.

XML-basierte Datenbanksysteme: Dieser Ansatz basiert darauf, eine geeignete interne Darstellung von XML-Dokumenten zu realisieren (Baumstrukturen) und Operationssemantiken auf diesen Darstellungen zu implementieren. So können XQuery und XPath als Abfragesprachen auf XML-Dokumenten betrachtet werden. Die ersten XML-basierten Datenbanksysteme sind schon erschienen. Bekannt sind z. B. Tamino [143], Natix [69] und Xindice [16].

In dieser Arbeit wird der Datenbankeditor ESCHER verwendet, dem das eNF^2-Datenmodell zugrunde liegt. Durch geeignete Darstellung von eNF^2-Tabellen als XML-Dokumente mit einem generischen Schema und die Bereitstellung von generischen Stylesheets, kann man ESCHER sogar als eine XML-Datenbank betrachten [222].

2.3 Mobile Geräte

Mobile Geräte können grob in drei verschiedene Kategorien[11] unterteilt werden: Mobiltelefone, PDA's und Pocket PC's. Mobiltelefone sind die meistverwendeten mobilen Geräten, deren primärer Gebrauch die Stimmübertragung ist. Mit der Entwicklung mobiler Übertragungstechniken kamen neue Funktionen wie SMS (*Short Message Service*) hinzu. Die heutige Generation mobiler Telefone (2. Generation) ist mit verschiedenen Funktionalitäten, wie WAP-Unterstützung (*Wireless Application Protocol*) und einen WML-Browser (*Wireless Markup Language*) versehen. Beispiele solcher Geräte sind Nokia 7210, Siemens ME45 und Ericsson T39. Die 3. Generation besitzt schnellere Prozessoren, mehr Speicher und kann somit für verschiedene Anwendungen wie Terminplaner oder Adreßbuch eingesetzt werden. Beispiel hierfür ist das Motorola Accompli and Sendo Smart Phones.

Die zweite Kategorie mobiler Geräte sind PDA's (*Personal Digital Assistent*). Aufgaben für die PDA's eingesetzt werden, sind z. B. Terminplaner oder Adreßdatenbanken. Verschiedene Arten von PDA's werden von Palm, Hewlett-Packard, Sony, Toshiba und anderen angeboten. Die Kapazitäten (Speicher, Prozessor und Grafik) dieser Geräte haben sich in letzter Zeit enorm vergrößert.

Pocket PC's sind vergleichbar mit dem Arbeitsplatzrechner am Anfang der 90-iger Jahre [226]. Sie verfügen über schnelle Prozessoren, große Hauptspeicher (RAM und ROM) und Grafikgeräte, die für die Ausführung von komplexen Multimedia-Anwendungen ausreichend sind. Ein Beispiel für solche Geräte ist der HP-Compaq iPAQ (3800-er, 3900-er Serie) mit dem Betriebssystem Microsoft Windows CE. Andere Firmen wie Casio, NEC oder Toshiba bieten ähnliche Geräte an.

Die Tendenz zeigt einen immer kleiner werdenden Unterschied zwischen diesen verschiedenen Gerätetypen. Sowohl Mobiltelefone als auch PDA's erhalten immer mehr Rechenpower, Speicherkapazität und Grafikfähigkeiten, so daß sie einem Pocket PC ähnlicher werden. In Zukunft werden sich diese Geräte gleichen, so daß es im Prinzip eine *einzige Art* von mobilen Geräten geben wird. Diese werden als Telefon, Adreßbuch oder für Multimedia-Anwendungen und den Internetzugang genützt.

2.3.1 Das mobile Internet

1957 wurde in Deutschland das erste Mobilfunknetz in Betrieb genommen - das sogenannte A-Netz. Dabei gab es 137 geografische Rufzonen. Um einen Teilnehmer zu erreichen mußte man seine geografische Zone kennen, und eine Vermittlungsperson zur Weiterleitung an den Gesprächspartner kontaktieren. Das A-Netz wurde 1972 durch

[11] F¨ur Details siehe [226, 21, 121, 39]

das B-Netz ersetzt, das zwar eine direkte Verbindung zum Gesprächpartner ermöglichte, jedoch auch verschiedene Rufzonen hatte. Erst mit der Einführung des C-Netzes gab es eine einheitliche Vorwahlnummer für ein gesamtes Land. A, B und C-Netze basierten auf analoger Technologie und werden daher als erste Generation (1G) bezeichnet. Standards der 1G waren *Advanced Mobile Phone System* (AMPS), *Nordic Mobile Telephone* (NMT) und *Total Access Communication System* (TACS).

Bereits zu Beginn der 80-iger Jahre (1982) erkannte man, daß die Zukunft der Digitalisierung auch vor der Mobilfunktechnik nicht halt machen wird, da diese Technik neue und komfortable Dienstleistungen erst effizient möglich macht. 1988 wurde das GSM-Standard (*Global System for Mobile Communication*) vom ETSI (*European Telecommunications Standards Organization*) verabschiedet. GSM entspricht dem Mobilfunkstandard der zweiten Generation. Anfang der 90-iger Jahre hat die zweite Generation (2G) ihren Anfang genommen. Maximale Übertragungsrate der 2G liegt bei 14.4 KB/s.

Die sich im Aufbau befindende 3. Generation soll etwa im Jahr 2005 zum weltweiten kommerziellen Einsatz kommen [4]. In Japan sind jedoch die ersten 3G-Netze im Jahr 2001 zu kommerziellem Einsatz gekommen [12]. Die 3G zeichnet sich durch eine sehr hohe Übertragungsrate von bis zu 2 Mbit/s (*Megabits per second*) aus, was die Übertragung von multimedialen Daten wie Postkarten, Filmen und Musik ermöglichen wird. Eine wichtige Verbesserung dieser Generation ist die weltweite Vereinheitlichung der Übertragungsprotokolle [139].

Übertragungstechniken

Analoge Übertragungstechniken der 1G wurden durch das erste digitale Netz, GSM-System, ersetzt. GSM war eine revolutionäre Entwicklung in der Geschichte mobiler Kommunikation. Es unterstützte mobile Sprach- und Datenübertragung und ermöglichte durch die Nutzung fremder Netze (Roaming) zum ersten Mal einen internationalen Zugriff über die gleiche Rufnummer. Die Übertragsrate erreichte 14.4 KB/s.

Als Verbesserung von GSM gelten GPRS (*General Packet Radio Service*) und HSCSD (*High-Speed Circuit Switched Data*). Die Übertragungsleistung von GPRS und HSCSD liegt bei 48000 Bit/s. EDGE (*Envolved Data for GSM Evolution*) gilt als Zwischenlösung zwischen GSM und UMTS. Die Technologie von EDGE ist eine Weiterentwicklung von GPRS und HSCSD mit einer Übertragungsleistung von bis zu 384000 Bit/s.

Die Technologie der nächsten Generation, UMTS (*Universal Mobile Telecommunications System*), wird zwischen 2004 und 2005 auf den Markt kommen. Ziel ist es, daß weltweit mit den gleichen Geräten gearbeitet werden kann und daß in strukturschwachen Gebieten das mobile Netz das Festnetz ersetzt. Als minimale Übertragungsleistung wird 38400 Bit/s angegeben, also das Maximum, was EDGE leistet. Die reale Übertragungsleistung soll bei etwa 2 Mbit/s liegen.

Für PDA's und Pocket PC's kommen andere Übertragungstechniken wie WLAN oder Bluetooth in Frage. Bluetooth ist ein offener Standard, der 1998 von der *Bluetooth Special Interest Group* (SIG) geschaffen wurde. Hierbei handelt es sich um eine Funktechnik für den Nahbereich, die sogenannte *Personal Area*, welche die drahtlose Anbindung mobiler Endgeräte ermöglicht, so daß sie untereinander Daten austauschen können [121].

Jedes Bluetooth-fähige Gerät kann innerhalb eines drahtlosen Kleinnetzes, eines sogenannten Piconetzes, mit einer Reihe von anderen Geräten kommunizieren. Alle Geräte unterhalten eine logische Verbindung untereinander, die nur im Falle der tatsächlichen Datenübertragung in eine physikalische Verbindung verwandelt wird. Die Identifizierung der einzelnen Bluetooth-Komponenten erfolgt durch eine eindeutige Adresse, ähnlich der MAC-Adresse im Ethernet-Netzwerk. Dadurch werden Rechte, Funktionen und Sicherheit der miteinander kommunizierenden Teilnehmer geregelt. Die Brutto-Übertragungsrate von Bluetooth beträgt 1 MB. Die Reichweite einer Bluetooth Verbindung liegt bei maximal 10 Meter. Beim *Long-Range-Bluetooth* können mit Hilfe eines Verstärkers 100 Meter erreicht werden, die dann aber einen entsprechend höheren Stromverbrauch haben [121, 6].

Ein WLAN (*Wireless Local Area Network*) ermöglicht drahtlosen Netzzugang über Funk. Seit 1997 gibt es eine Normierung dieser Übertragungstechnik durch das *Institute of Electrical and Electronics Engineers* (IEEE) in dem IEEE 802.11 Standard. Aktuell spricht man bei der Verwendung von dem Begriff Wireless LAN in der Regel vom IEEE 802.11b Standard. Wireless LAN nach dem IEEE 802.11b Standard erlaubt momentan Bruttoübertragungsraten von bis zu 11 MBit/s. Durch die aufwendigen Übertragungsverfahren, Verschlüsselung und Protokolle bleiben netto maximal 6 MBit/s an Bandbreite übrig.

Die Anbindung der drahtlosen Clients erfolgt über sogenannte *Access Points* (APs), die die Verbindung zum Ethernet herstellen. Die Reichweite eines Senders im Funknetz ist abhängig von den räumlichen Gegebenheiten vor Ort. Im Freien oder in einer großen Halle sind bis zu 520 m erreichbar. Mit zunehmender Entfernung des Clients vom Access Point sinken Qualität und Transferrate der Übertragung. Entscheidende Faktoren, die die Qualität der Übertragung beeinflussen, sind dabei die Baumaterialien der Wände, deren Dicke und die Anzahl der Wände zwischen Access Point und Client.

Zugriff auf Web-Daten

Zur Kommunikation zwischen einem Web-Server und einem mobilen Gerät, muß eine Verbindung zwischen beiden Geräten hergestellt werden. Diese Verbindung kann nicht direkt hergestellt werden. Daher muß eine Zwischenschicht zwischen den beiden Geräten vermitteln. Hierfür existieren drei verschiedene Ansätze.

WAP steht als Abkürzung für das *Wireless Application Protocol*. Es wurde 1997 vom WAP-Forum [13] spezifiziert, dem Hersteller, Netzbetreiber und Webentwickler angehören. Die Spezifikation definiert verschiedene Eckwerte, wie mobile Geräte auf das Internet zugreifen können. WAP eignet sich in erster Linie für mobile Geräte wie Mobiltelefone und PDAs, die über den GSM-Standard oder dessen Nachfolger, wie GPRS oder UMTS, mit dem Internet kommunizieren.

Beim WAP-System liegt zwischen einem Client und dem Server ein Gateway. Dieses ist für den nahtlosen Übergang von der mobilen Übertragung zum Internet zuständig. Diesem Gateway (auch *WAP-Proxy* genannt) fällt die Aufgabe zu, zwischen dem Telefonnetz und dem Internet zu übersetzen. Vom Gateway zum Internetserver werden hier die Protokolle auf HTTP und TCP/IP umgesetzt und die dann erhaltenen Daten vom Gateway zum Mobiltelefon binär codiert und komprimiert, um die Datenmengen zu verkleinern und Bandbreite zu sparen. Das WAP-Gateway kann physikalisch wahlweise bei dem Mobilfunk-Anbieter (D1, D2, E-Plus, usw.), bei einem Service-Provider oder bei dem, den Inhalt liefernden Unternehmen installiert werden.

Die zweite Technologie nennt sich *Web Clipping* [21]. Sie wurde von *Palm Computing* zur Darstellung von Internetinformationen auf dem Palm entwickelt. Dieser Ansatz geht von der Annahme aus, daß das Ziel des Anwenders darin besteht, gezielt Informationen wie Börsenkurse, Nachrichten, Postleitzahlen etc. abzufragen. Von Interesse sind im Wesentlichen nur die ,,Roh-Daten", die eine Abfrage liefert und nicht die komplette Web-Seite, in welche die Daten eingebettet werden [156].

Zu diesem Zweck erschuf 3COM das Konzept der *Palm Query Applications* (PQA). Jede PQA besteht aus Abfrage- und Antwortformularen, die speziell für einen Internet-Dienst programmiert worden sind, d.h. für jede ,,Informationsart" ist eine eigene Applikation notwendig. Nach der Installation der PQAs befindet sich der HTML-Code lokal auf dem Palm. Der PDA muß nur noch die vom Benutzer gewünschten Daten in die entsprechenden Formulare integrieren und anzeigen. Der PQA kommuniziert nicht direkt mit dem Internet, sondern mit dem *Web-Clipping-Proxy-Server* des *3COM-Data-Center*. Der Proxy-Server setzt die Anfrage um und gibt diese an den adressierten Internet-Dienst weiter. Der Rückweg erfolgt analog.

Eine Dritte Möglichkeit zum Zugriff auf Internetdaten von mobilen Geräten aus ist *i-mode*. Sie wurde von einer japanischen Firma (*NTT DoCoMo*) entwickelt, und benutzt *compact HTML* (cHTML) zur Darstellung.

Anzeigesprachen

Zur Anzeige der Daten müssen standardisierte Beschreibungssprachen verwendet werden. Mehrere Geräte enthalten einen Browser (z. B. den *Microsoft Mobile Explorer* MME) zum Anzeigen von üblichen HTML- oder WML-Seiten. Der Vorteil bei der Verwendung von HTML liegt darin, daß schon vorhandene Informationen ohne Umwandlung angezeigt werden können.

Die Sprache HDML (*Handheld Device Markup Language*) wurde 1997 für die Anzeige von Web-Inhalt erstellt. Mit dem UP-Browser können HDML-Dokumente angezeigt werden. HDML enthält jedoch einige Einschränkungen, die im WAP und WML aufgehoben wurden [21].

WML (*Wireless Markup Language*) ist ein Teil des WAP-Standards. WML basiert auf XML und enthält Elemente zur Darstellung von Text, Hierarchien und sogenannte *Screens* (Auch *„decks"*, oder *„cards"* genannt). Zwischen diesen „Karten" können links verweisen. Zu WML gehört die Skriptsprache *WMLScript*, die eine Programmierung im Rahmen des WAP-Protokolls ermöglicht.

Eine weitere Sprache zu Inhaltsbeschreibung ist cHTML (*compact HyperText Markup Language*). Sie besteht aus einer Untermenge von HTML 3.0, die für den Einsatz bei mobilen Telefone optimiert ist.

2.3.2 Anwendungen auf mobilen Geräten

Für die Entwicklung von mobilen Client-Server Anwendungen sind Entwicklungen in drei unterschiedlichen Bereichen notwendig [109].

- Die Erweiterung von mobilen Geräten durch spezielle Software, um Einschränkungen dieser Geräte für den Anwendungsentwickler transparent zu machen (*Mobile-aware Adaptation* [142]).
- Erweiterung des Client-Server Modells (*Extended Client-Server Model*). Dabei werden Clients und Server so erweitert, daß Aufgaben von jeder Seite auf der anderen verlagert werden können [112].
- Techniken zur Datenübertragung wie Struktur und Aufbau von Nachrichten, Aufbau von Cache und Bewahrung von Konsistenz beim Client usw. (*Mobile Data Access*) [178].

Für die Entwicklung von Anwendungen für mobile Geräte, muß man deren Einschränkungen bezüglich Anzeige, Übertragungsgeschwindigkeit und Visualisierung berücksichtigen. In naher Zukunft kann sich die Prozessorleistung, und Speicherkapazität noch vergrößern. Gegenüber Arbeitsplatzrechnern werden diese Geräte aber immer eine schwächere Leistung haben, da sie auch kleiner und leichter werden müssen.

Grundsätzlich gibt es zwei Möglichkeiten, Anwendungen für mobile Geräte zu entwickeln. Bei der ersten werden Sprachen wie C oder Assembler verwendet, bei denen man internes Wissen über die Zielgeräte benötigt. Im zweiten Ansatz werden vorhandene Entwicklungsumgebungen verwendet. Dadurch werden interne Details verborgen und dem Anwendungsentwickler ermöglicht, sich auf die Funktionalität der Anwendung zu konzentrieren. Zwei solche Umgebungen sind Java (*Java 2 Micro Edition* oder *PersonalJava* [140]), und *.NET Compact Framework von Microsoft*.

Java 2 Micro Edition

Der große Vorteil von Java ist ihre Plattformunabhängigkeit. Für die Ausführung von Java Byte Code ist eine Java Virtuelle Maschine (JVM) zuständig. Daher könnte man prinzipiell Java-Programme auf jedem Gerät zur Ausführung bringen, wenn eine entsprechende JVM auf diesem Gerät existiert. Für mobile Geräte mit begrenztem Speicher und Rechenkapazität ist eine übliche JVM überdimensioniert. Auch Standardklassen, die zu der Sprachspezifikation ebenso gehören, sind für kleine Geräte meistens zu groß. Ein sehr kleiner Teil dieser Klassen ist in der Regel auf solchen Geräten notwendig.

Sun hat verschiedene Java-Spezifikationen für mobile Geräte entwickelt. Die erste war PersonalJava 1.0 [181]. Sie wurde für die Ausführung von Java-Anwendungen mit der Java 1.1 Plattform auf mobilen Geräten entwickelt, und benutzt die gleiche JVM wie JDK 1.1. In der Spezifikation werden die Softwarebibliotheken bestimmt, die eine PersonalJava-Implementierung haben muß. Unter anderem unterscheiden sich die Pakete `java.io` und `java.lang` von denen bei JDK 1.1. Ein Beispiel für eine Implementierung von PersonalJava ist die *Jeode VM* [104], die auf verschiedenen Geräten läuft.

Schnell hat man erkannt, daß es weitere Spezifizierungen bzw. Einschränkungen von Java-Umgebungen für die verschiedenen Geräte geben sollte. Bei der Java 2 Plattform wurden drei sogenannte *Editions* definiert: *Java 2 Standard Edition* (J2SE) für Arbeitsplatzrechner, *Java 2 Enterprise Edition* (J2EE) für serverseitige Anwendungen und *Java 2 Micro Edition* (J2ME) für mobile Geräte. Eine Edition stellt die gesamte Java-Umgebung zur Ausführung von Java-Anwendungen mit der dazugehörigen JVM und spezifiziert die erforderlichen Java API-Klassen [21].

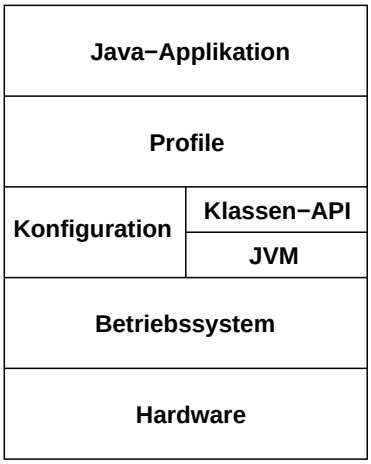


Abbildung 2.1: Editions, Konfi gurationen und Profi les

Innerhalb einer Edition können sogenannte *Configuration* angegeben werden. Eine solche Konfiguration ist die Spezifikation der minimalen Java-Bibliotheken und Funktionalitäten der VM, die der Entwickler von einem Gerät, das diese Konfiguration implementiert, erwarten kann. Eine Konfiguration bezieht sich also auf eine Gruppe von Geräten, die ähnliche Merkmale aufweisen. Die Entscheidungsmerkmale, zu welcher Konfiguration ein Gerät gehört, liegen bei dem Speicher, Netzwerkfähigkeiten und Prozessorleistung. Innerhalb von J2ME sind die zwei Konfigurationen CDC [183] (*Connected Device Configuration*), und CLDC [182] (*Connected Limited Device Configuration*) vorhanden.

CLDC benutz die KVM (*The Kilobyte Virtual Maschine* [184]), die von Sun Microsystems Laboratories in einem Forschungsprojekt entwickelt wurde. Sie wurde in C geschrieben und braucht wenige Kilobytes Speicher zur Ausführung (128 - 256 KB) und eignet sich gut für 16/32-bit Geräte [184]. CLDC zielt auf Geräte mit 128 KB ROM (zur Ausführung der JVM), 32 KB RAM, eingeschränkter Benutzerschnittstelle, niedrigen Stromverbrauch und einer Netzwerkunterstützung. Das Betriebssystem solcher Geräte sollte in der Lage sein, die JVM und Java-Programme in den Speicher zu laden und zur Ausführung zu bringen. Die API-Klassen von CLDC sind von denen bei J2SE abgeleitet, und sind Untermengen von `java.lang`, `java.util`, `java.io` etc. Neue Klassen, z. B. für grafische Benutzerschnittstellen, Netzwerkverbindung und persistente Datenspeicherung sind im Paket `java.microedition` zusammengestellt.

Die zweite Konfiguration innerhalb J2ME, CDC, bildet eine Obermenge von CLDC. Sie besteht aus CVM (*The C Virtual Machine* [183]) und Standardklassen zur Ausführung von Java-Programmen auf leistungsstärkeren Geräten [183]. CDC zielt auf 32-Bit Geräte mit 2 MB Hauptspeicher (RAM und ROM) und einer Netzwerkverbindung. Die CVM ist in C und Assembler geschrieben und enthält volle Unterstützung der Spezifikation der virtuellen Maschine in J2SE 1.3 [183].

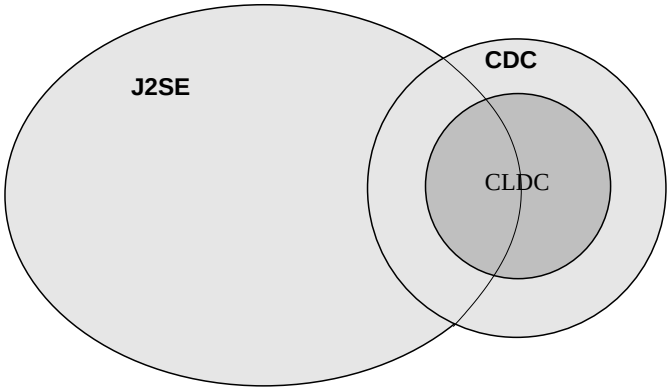


Abbildung 2.2: Beziehung zwischen CLDC, CDC und J2SE API.

Innerhalb einer Konfiguration können verschiedene *Profiles* angegeben werden. Ein Profile ist eine Erweiterung einer Konfiguration. Es stellt eine Klassensammlung für die Entwicklung von Anwendungen auf bestimmten Gerätetypen dar. Da unterschiedliche Geräte unter dieselben Konfiguration fallen, müssen mehrere Profiles für bestimmte Geräte entwickelt werden. Z. B. können PDA's und Mobiltelefone zur gleichen Konfiguration gehören (CLDC), aber unterschiedliche Charakteristiken (z. B. verschiedene Anzeigen) haben und somit zu unterschiedlichen Profilen gehören.

Der *MID-Profile* (*Mobile Information Device Profile* [185]) ist das bekannteste Profile innerhalb CLDC und zielt auf Mobiltelefone mit geringem Speicher (128 KB RAM, 8 KB ROM) und einer Anzeige von 96 X 54 Pixeln [185]. Es unterstützt Funktionen für grafische Oberflächen, Netzwerkverbindung und persistente Datenspeicherung. MIDP-Anwendungen werden in Anlehnung an Applets, *MIDlets* genannt. Das *Bluetooth Profile* ist ebenfalls innerhalb CLDC definiert, und richtet sich an Geräte mit 512 KB Speicher und Bluetooth-Unterstützung. Das *PDA-Profile* zielt auf PDA-Geräte mit größerem Speicher (zwischen 512 KB und 16 MB) und größeren Anzeigegeräten. Im Unterschied zum MID, unterstütz das PDA-Profile AWT (*Abstract Windowing Toolkit*).

Innerhalb CDC sind *The foundation Profile*, *Personal Basis Profile*, *Remote Method Invocation Profile*, *Game Profile* und der *Personal Profile* definiert. Dabei baut der Personal Profile auf dem Personal Bassis Profile, und zielt auf Geräte, die in der älteren Spezifikation, PersonalJava unterstützen[12].

2.4 Kontrolle der Nebenläufigkeit

Bei Groupware-Anwendungen werden im wesentlichen die folgenden Verfahren für die Nebenläufigkeitskontrolle eingesetzt [38, 179]:

Optimistische Verfahren: Das optimistische Modell geht von der Annahme aus, daß Konflikte in CSCW-Systemen eine Seltenheit sind, da die Dokumente normalerweise in kleineren Einheiten strukturiert sind. Ein Vertreter dieses Verfahrens ist das Knowledge Management System (KMS) [20].

Sperrverfahren: Sperrverfahren versuchen eine sichere Konsistenz zu gewährleisten. Dabei können Sperrverfahren grob in zwei Kategorien unterteilt werden: *Explizite* und *Implizite* Sperrverfahren. Bei der Verwendung von Sperrverfahren gibt es im wesentlichen drei Probleme :

[12] Nach den Spezifikationen in J2ME unterstützt z. B. die Jeode Runtime Environment, die eigentlich PersonalJava implementiert, das Personal Profile.

- Erstens entsteht ein Zusatzaufwand durch das Anfordern und Setzen der Sperre. Wartezeiten sind zu berücksichtigen, falls die Sperre nicht sofort gewährt werden kann.
- Zweitens hat die Sperrgranularität entscheidenden Einfluß auf den erreichbaren Grad der Nebenläufigkeit. Es muß geklärt werden, ob nur ein einzelnes Zeichen, ein Satz oder gar das ganze Dokument gesperrt werden soll.
- Schließlich muß man klären, wann eine Sperre gesetzt werden soll. Ist beispielsweise das Bewegen eines Cursors alleine schon ausreichend, um die betroffene Zugriffseinheit zu sperren?

Floor-Passing-Verfahren: Floor-Passing-Verfahren versuchen die Nebenläufigkeitskontrolle durch das Abgeben der Kontrolle abwechselnd an die Benutzer zu regeln. Dabei werden verschiedene Algorithmen benutzt, um die Kontrolle abzugeben. Grundsätzlich besteht wieder der Unterschied zwischen *explizitem* Floor-Passing, bei dem die Benutzer die Kontrolle vom zentralen System auffordern, und nach Warteschlangenprinzip bekommen, und *implizitem* Floor-Passing, wo das System die Vergabe der Kontrolle an alle Benutzer in einer Reihe durchführt.

Transformations-Verfahren: In diesem Verfahren werden Kopien der Objekte bei jedem Client erzeugt, und Operationen der Benutzer bei den anderen Benutzern durch eine *Transformations-Matrix* so transformiert, daß die Operation auf jeder Seite die gleiche Auswirkung auf das gesammte Dokument hat [63]. Dieses Verfahren wurde von *Ellis* für die Nebenläufigkeitkontrolle in dem group editor *Grove* entworfen, das eine replezierte Datenhaltung verwendet [179]. In [146] bauen die Autoren eine Erweiterung dieses Verfahrens für eine gruppenbewußte Tabellenkalkulation-Anwendung. Dieses Verfahren ist das einzige, das nicht aus der Welt der Datenbanken und Betriebssysteme stammt.

In den existierenden Verfahren wird von den Autoren bemerkt, daß die Kontrolle der Nebenläufigkeit durch die Awareness beeinflußt wird [88]. Jedoch definiert keines dieser Verfahren diese Abhängigkeit. Diese Verfahren berücksichtigen also nicht diesen Zusammenhang, sondern werden wie bei Datenbanksystemen unabhängig von Awareness entwickelt. Es wird von Leseoperationen und Lesesperren gesprochen, ohne zu unterscheiden, ob das Lesen für die Visualisierung dient oder Voraussetzung für die Weiterverarbeitung in anderen Transaktionen ist (siehe Abschnitt 3.6).

2.5 Awareness

Für die synchrone Gruppenarbeit mit gemeinsamen Informationsräumen ist Awareness der wichtigste Aspekt [87, 42]. Bei der Bearbeitung von gemeinsamen Informationsräumen geben Awareness-Informationen jedem Benutzer *Vorstellung über die Aktivitäten anderer Gruppenmitglieder*. Für die Behandlung der Awareness in dieser Arbeit entnehmen wir die Definition von *Dourish* [60, S. 107]:

Definition 2.1 (Awareness)
„[...] awareness is an understanding of the activities of others, which provides a context for your own activity. This context is used to ensure that individual contributions are relevant to the group's activity as a whole, and to evaluate individual actions with respect to group goals and progress. The information, then, allows groups to manage the process of collaborative working."

In der Literatur wird zwischen vier Awareness-Arten unterschieden [95]:

Informal-Awareness: Gibt Informationen über mögliche Kommunikationspartner. Dies dient zur Erleichterung der Kommunikationsaufnahme.

Group-Structural Awareness: Gibt Informationen über beteiligte Benutzer, deren Rollen und Aufgaben innerhalb der Gruppe.

Social-Awareness: Gibt soziale Informationen über vorhandene Benutzer. Diese Informationen sind der Art: was macht ein Benutzer gerade? Achtet er auf meine Aktivitäten? usw.

Workspace-Awareness: *Greenberg* definiert Workspace Awareness wie folgt: *„We define workspace Awareness as the up-to-the minute knowledge a person requires about another group member's interaction with a shared workspace if they are to collaborate effectively"* [87, S. 2]. Workspace-Awareness ist also das aktuelle Wissen der Benutzer über die momentanen Aktivitäten anderer Gruppenmitglieder im gemeinsamen Informationsraum. Workspace-Awareness überschneidet sich natürlich mit anderen Awareness-Arten und kann daher andere Awareness-Arten ersetzen [42]. Daher bezieht sich in dieser Arbeit der Ausdruck Awareness, wenn nicht anders angegeben, immer auf Workspace-Awareness.
Workspace-Awareness kann aus Sicht eines Interessenten (Benachrichtigungsempfängers) in vier Modi unterteilt werden [13]:

- Synchronität
- Symmetrie
- Interessenspezifizierung
- Kopplung

Dabei kann man bezogen auf diese Modi zwischen *synchroner* und *asynchroner*, *symmetrischer* und *asymmetrischer*, *impliziter* und *expliziter* sowie *gekoppelter* und *ungekoppelter* Awareness unterscheiden. Tabelle 2.1 zeigt die Qualifizierung von Awareness anhand dieser Modi im GroupDesk-Modell [77].

Für die Generierung und die Darstellung von Awareness-Informationen muß ein sogenanntes Awareness-Modell entwickelt werden. Ein Awareness-Modell wird wie folgt definiert [42, S. 76]:

[13] F¨ur detailliertere Information siehe [87, 95, 42, 77]

	synchron	**asynchron**
gekoppelt	Was geschieht momentan im aktuellen Arbeitsbereich?	Was geschah bisher im aktuellen Arbeitsbereich?
ungekoppelt	Was ist momentan sonst noch von Interesse?	Was war bisher sonst noch von Interesse?

Tabelle 2.1: Qualifi zierung von Awareness-Informationen nach [77]

Definition 2.2 (Awareness-Modell)
Ein Awareness-Modell beschreibt den Mechanismus der Verteilung der für Awareness relevanten Informationen. Es

- *beschreibt die Repräsentation und die Art der Spezifikation des Interessenprofils (explizit und/oder implizit)*
- *beschreibt den Aufbau und Inhalt der Ereignismeldungen/Benachrichtigungen (Informationsmodell) und*
- *beschreibt, wie die Filterung oder Bewertung von Benachrichtigungen erfolgt (symmetrisch/asymmetrisch gekoppelt/ungekoppelt)*

Darüber hinaus muß ein Awareness-Modell natürlich auch *die Metapher erfinden, die diese Informationen auf der Benutzeroberf läche auf intuitiv verständlicher Weise darstellen*. In der Literatur werden vier verschiedene Awareness-Modelle genannt:

Subscribe-Modell: Bei dem Subscribe-Modell werden die Interessenprofile der Benutzer explizit vom Benutzer erhalten. Das Awareness-Modul entscheidet anhand dieser Informationen, ob ein Benutzer über eine bestimmte Operation benachrichtigt wird. Hierzu ist keine Filterung notwendig, sondern nur eine einfache Entscheidung zwischen Benachrichtigen/Nicht-Benachrichtigen. Das Subscribe-Modell wird in verschiedenen Implementierungen benutzt. Beispiel CSCW3 [93], Elvin [168].

Sitzung-Modell: Beim Sitzung-Modell basiert die Filterung von Awareness-Informationen anhand der Zugehörigkeit eines Benutzers zu einer Sitzung. Eine Sitzung umfaßt die Benutzer, die zu dieser Sitzung gehören sowie die gemeinsamen Objekte, die von diesen Benutzern bearbeitet werden. Für das Awareness-Modul ist die Zugehörigkeit des Benutzers zu einer Sitzung interessant, nicht aber die Art, wie diese Sitzungen entstehen und wie ein Benutzer eine Sitzung betritt oder verläßt. Die Filterung von Awareness-Informationen ist einfach, jedoch können gleiche Objekte gleichzeitig von Benutzern verschiedener Sitzungen bearbeitet werden. In diesem Fall müssen Awareness-Informationen zu Benutzern unterschiedlicher Sitzungen geleitet werden. Anderenfall kann dies zu Konflikten bei der Lösung der Nebenläufigkeitskontrolle führen. Einige Implementierungen sind GroupKit [158] und wOrlds [72, 71].

Groupdesk-Modell: *„Das GroupDesk-Modell unterscheidet vier sogenannte* Orientierungsmodi*, die sich aus der Kombination der Awareness-Modi bezüglich der Kopplung und bezüglich der Synchronität ergeben.*" [42, S. 81] (siehe Tabelle 2.1). Bei diesem Modell wird der Informationsraum durch Objekte dargestellt, die *Relationen* untereinander haben. Benutzer werden auch als Objekte modelliert. Bearbeitet ein Benutzer ein Objekt so besteht zwischen ihm und dem Objekt eine *Arbeitsrelation*. Für die Filterung von Awareness-Informationen werden sog. *Interessenkontexte* benutzt. Sie werden vom Systementwickler bezüglich der Objekte eingestellt. Benutzer können explizit ihr Interesse an den Ereignissen für bestimmte Objekte angeben. Eine Implementierung erfolgt durch das gleichnamige Projekt GroupDesk [77].

Fokus/Nimbus-Modell (Spatial Modell): Das Fokus/Nimbus-Modell ist ein räumliches Modell. Awareness-Nachrichten werden anhand der Überschneidung der sog. *Foki* und *Nimbi* zweier Benutzer berechnet. Dabei ist der Fokus eines Benutzers die Menge der Objekte, die der Benutzer sieht (sehen will). Der Nimbus ist der Bereich, wo der Benutzer agiert. Ein Benutzer wird über die Objekte in seinem Fokus benachrichtigt. D. h. je mehr ein Fokus von dem Nimbus eines anderen Benutzers enthält, desto mehr weiß er über seine Aktivitäten und umgekehrt. Das Fokus/Nimbus-Modell ist ein theoretisches Modell und wurde bis jetzt in keinem System implementiert[14].

2.6 Zugriffskontrolle

Wie schon erwähnt ist die Zugriffskontrolle ein Randthema in dieser Arbeit. Allgemein spielt die Zugriffskontrolle in CSCW-Systemen jedoch eine zentrale Rolle [64, 65, 91, 169, 153]. *Dewan* und *Shen* bekräftigen, daß die Zugriffskontrolle weniger Aufmerksamkeit in der Forschung genießt: *„there has been relativeley little work done in controlling access to the collaboration. Most collaborative systems give all collaborators the same rights to all objects and expect to be controlled by social protocol*" [169, S. 51]. In der Literatur existieren trotzdem verschiedene Modelle und Vorschläge für die Zugriffskontrolle in Groupware-Systemen.

In [169] stellen die Autoren ein allgemeines Modell für die Zugriffskontrolle in Groupware-Applikationen vor, das Besonderheiten der Zugriffskontrolle bei der Gruppenarbeit berücksichtigt. Das Modell schafft neue Rechte z. B. „ *Recht zum Ändern der gemeinsamen Anzeige*" und „*Recht zur Abkupplung der Anzeige mit anderen Benutzern*". Das Modell setzt ein bestimmtes Datenmodell voraus, das die Strukturierung der Daten in einer feinen Granularität erlaubt (*active variables*). Rechte können auf dieser Basis vererbt werden. Besonders hervorzuheben ist die Möglichkeit, die Rolle eines Benutzers dynamisch zu verändern (*dynamic user roles*).

[14] Unser Awareness-Modell ist dem Fokus/Nimbus-Modell sehr ¨ahnlich (siehe Kapitel 3). Eine weitere Implementierung wurde in dem Projekt AETHER [163] realisiert. Dieses Projekt wird allerdings nicht mehr weiter gepflegt!

Greif bekräftigt, daß traditionelle Verfahren für die Zugriffskontrolle zwar von einigen Groupware-Applikationen eingesetzt worden sind, aber nicht in ausreichendem Maß: „*Even with the ability to control access to abstract operations, the use of conventional access criteria based on owner, group and public is rarely sufficient to implement the rich policy desired in a cooperative application*" [91, S. 200]. In dieser Arbeit [91] wird ebenfalls ein modifiziertes Modell vorgestellt, das die Zugriffskontrolle anhand verschiedener annehmbarer Rollen eines Benutzers verwalten kann.

Bei der Untersuchung von verschiedenen Groupware-Applikationen und einigen Modellen aus der Literatur wurde festgestellt, daß einige neue Anforderungen an die Implementierung der Benutzerkontrolle bei Groupware-Applikationen gestellt werden. Diese unterscheiden sich aber kaum von der Zugriffskontrolle etwa bei Betriebssystemen oder Datenbanksystemen. Die Implementierung der Zugriffskontrolle in vorhandenen Systemen bestätigt diese Aussage. Die Schaffung von neuen Rechten und verschiedenen Benutzerrollen kompliziert eher die Anwendung für den Benutzer, eröffnet trotzdem keine neuen Möglichkeiten. So lassen sich z. B. rollenbasierte Zugriffskontrollen durch die Möglichkeit, die Gruppe eines Benutzers zu ändern, durchaus „simulieren". Die Vergabe von Rechten in feineren Granularität als an ganze Dokumente ist ein schon bekanntes Konzept aus der Datenbankwelt.

2.7 Beispiele vorhandener Anwendungen

Nachdem die wichtigsten Aspekte der synchronen Gruppenarbeit im Detail in den vorigen Abschnitten betrachtet wurden, sollen in diesem Abschnitt verschiedene existierende Anwendungen für die synchrone Gruppenarbeit, sowie auf gemeinsame Informationsräumen basierende Systeme betrachtet werden. Dabei erhebt diese Betrachtung keinen Anspruch auf Vollständigkeit[15], sondern versucht unterschiedliche Ansätze für die Implementierung synchroner Gruppenarbeit vorzustellen und die Relevanz einiger Anwendungen für diese Arbeit zu untersuchen.

2.7.1 Grove

Grove [62] ist einer der ersten Gruppeneditoren mit Unterstützung für Concurrency und Awareness. Im Gegensatz zu damals vorhandenen Gruppeneditoren (CES [92], Quilt [70], und Shared Books [125]) unterstützt Grove „real-time groupware" [63].

In Grove können Benutzer ein gemeinsames Dokument synchron bearbeiten. Dabei hat jeder Benutzer ein lokales Bild des gemeinsamen Fensters. Änderungen von einem Benutzer in dem Dokument werden an andere Benutzer übertragen (*notification*).

[15] Für mehr Information siehe [62, 160, 138, 7, 32, 129, 73]

Für die Lösung der Nebenläufigkeit wurde ein neues Konzept entwickelt, das lokal ausgeführte Operationen bei den anderen Benutzern nach einer geeigneten Transformation ausführt. Der Editor unterstützt Awareness-Anzeige, so daß die Benutzer sehen können, wer momentan an dem Dokument arbeitet, und von welchem Benutzer eine Veränderung kommt.

Die verwendeten Mechanismen in Grove lassen sich schwer auf andere Anwendungen übertragen. Das verwendete Datenmodell ist nicht generisch, sondern speziell für die Speicherung von Textdokumenten ausgelegt. Das Verfahren für die Nebenläufigkeit eignet sich möglicherweise für Texteditoren, jedoch nicht etwa für gemeinsames Editieren von Datenbankinhalten oder gemeinsame Zeichenprogramme.

2.7.2 GroupKit

GroupKit [160, 89, 90] ist ein Tcl/Tk-basiertes Framework für die Entwicklung von Groupware, das an der Universität Calgary von *Greenberg* et al. entwickelt wurde. Hauptsächlich stellt GroupKit gruppensensitive grafische Komponenten und Mechanismen zur Verfügung, die bei der Entwicklung der Oberfläche eingesetzt werden können. Neue Paradigmen wie *telepointer* und *shared scrollbars* sind Beispiele dafür. Außerdem stellt GroupKit Mechanismen für *session management* und automatisches Weiterleiten von Awareness-Informationen bereit. Diese basieren jedoch hauptsächlich auf Änderungen auf der grafischen Basis, nicht aber auf generischen Operationen, die Veränderungen des gemeinsamen Informationsraumes propagieren. In GroupKit ist es schwer zu erkennen, wie gemeinsame Daten strukturiert werden (Datenmodell) und wie die Lösung der Nebenläufigkeit bei Zugriff auf gemeinsame Daten von mehreren Benutzern gelöst wird.

In der Praxis eignet sich GroupKit für die Entwicklung von Anwendungen mit geringen Zugriffen auf gemeinsame Informationsräume, wie gemeinsame Zeichenflächen, Diskussionsräume und gemeinsames Lesen von Dokumenten.

2.7.3 NetMeeting und SameTime

NetMeeting (Microsoft) [138] und SameTime (Lotus) [7] sind zwei Programme für die Unterstützung von Gruppenarbeit über *Application-Sharing*. Die Grundidee bei diesem Ansatz ist es mehreren Benutzern zu ermöglichen, ein Einbenutzerprogramm zu benutzen und dabei die Anzeige dieses Programms bei den Gruppenmitgliedern zu verteilen. Mit zusätzlichen Tools für die Unterstützung der Kommunikation (Chat, Stimm- und Videoübertragung) eignen sich solche Programme für Konferenzen und Diskussionsforen, Software Support sowie virtuelle Seminare.

Die Idee des Application-Sharing ist keinesfalls neu. Systeme für *Screen-Sharing* (Propagieren des gesamten Bildschirms) wie SharedX [78], sowie Shared Window (Propagieren eines einzigen Fensters), existieren schon seit Ende der 80-er Jahren. Dieser Ansatz ermöglicht jedoch nicht den Zugriff der Benutzer auf einen gemeinsamen Informationsraum. Zwar kann man bei Netmeeting die Kontrolle über das Dokument einem anderen Gruppenmitglied übergeben, kann aber nicht mehreren Benutzern gleichzeitig den Zugriff auf dasselbe Dokument ermöglichen. Die Sperrgranularität gilt immer für das ganze Dokument [29].

2.7.4 BSCW

BSCW (*Basic Support for Cooperative Work*) wurde 1995 an der GMD entwickelt [32]. Damals war BSCW als eine Erweiterung eines Web-Servers um Funktionen für die Gruppenarbeit gedacht. BSCW ermöglichte räumlich verteilten Benutzern den (asynchronen) Zugriff auf einen gemeinsamen Informationsraum. Der gemeinsame Informationsraum kann verschiedene Arten von Informationen wie Dokumente, URL-Links auf anderen Web-Seiten, Kontaktinformationen usw., enthalten. Der Inhalt eines Informationsraums wird in Form einer Verzeichnishierarchie dargestellt [23, 24]. Benutzer können sowohl Daten aus diesem Informationsraum *herunterladen* als auch lokale Informationen in das gemeinsame Informationsraum *hochladen* (*upload*).

Awareness-Informationen werden in BSCW in Form von *Events services* generiert. Diese stellen Informationen über die Aktivitäten anderer Benutzer im gemeinsamen Informationsraum dar. Allerdings beziehen sich diese Informationen auf ganze Dokumente wie „*Dokument wird gelesen*“, „*Neues Dokument erstellt*“ usw. [23]. Die Kontrolle der Nebenläufigkeit basiert ebenfalls auf Dokumentebene, wobei Dokumente explizit vom Benutzer gesperrt werden können [23]. BSCW bietet natürlich viele weitere Funktionen wie Such- und Sortierfunktionen usw..

2.7.5 Lotus Notes

Lotus Notes [129] ist eine der ersten kommerziellen Groupware-Anwendungssoftware, die 1989 aus dem Produkt Plato Notes[16] entstanden ist. Im Jahr 1996 wurden mit der Version 4.0 Erweiterungen für die Verwendung von Lotus im Internet geschaffen.

Die Architektur von Notes beruht auf einem zweistufigen Client/Server-Modell. Abweichend vom dreistufigen Modell, das aus Präsentation-, Applikations- und Datenschicht besteht, wird die Applikations- und Datenschicht in Notes Datenbanken zusammengefaßt [73]. Die Präsentationschicht wird entweder über einen HTTP- oder *Notes-Client* zur Verfügung gestellt.

[16] Plato Notes wurde im *Computerbased Education Research Laboratory* der Universit¨at von Illinois entwickelt.

Die Grundtechnologie von Notes ist die Bereitstellung eines gemeinsamen Informationsraumes, der mehrere *Lotus Datenbanken* enthalten kann. Das verwendete Datenmodell basiert im wesentlichen auf dem hierarchischen Datenmodell [73]. Ein wesentlicher Unterschied von Notes-Datenbanken zu „üblichen" Datenbanken ist, daß Anwendungslogik, Strukturinformationen und Dokumente zusammen gespeichert werden. Die Gestaltungselemente, die die Struktur der Datenbank beschreiben, untergliedern sich in: *Masken, Ansichten, Feld-Definitionen, Navigatoren, Agenten und Skript-Bibliotheken*. Dabei bietet Notes eine integrierte Skriptsprache (LotusScript) für die Entwicklung von *Notes Anwendungen*. Dokumente sind die kleinste Einheit, in der Informationen gespeichert werden.

Notes wird von einigen Autoren als *Meta Groupware* bezeichnet, da Notes ein Softwaresystem ist, mit dessen Hilfe Groupware-Applikationen unterschiedlicher Ausprägungen implementiert werden können. Jedoch begrenzen sich die Entwicklungsmöglichkeiten auf die asynchrone Gruppenarbeit[17]. So ermöglicht Notes nicht den gleichzeitigen schreibenden Zugriff auf ein gemeinsames Dokument [73], und bietet keine Möglichkeiten für Awareness im synchronen Fall.

2.8 Zusammenfassung

Wir haben verschiedene Datenmodelle vorgestellt, und die Vor- und Nachteile dieser Modelle für die synchrone Gruppenarbeit diskutiert. Dann haben wir verschiedene Awareness-Modelle und Mechanismen für die Kontrolle der Nebenläufigkeit, die in verschiedenen Anwendungen zum Einsatz kommen, betrachtet. Anschließend führten wir mögliche Web-Technologien und XML als mögliches Austauschformat zwischen einer Groupware-Anwendung und einem Groupware-Server ein.

Der Einsatz von synchroner Gruppenarbeit für die Bearbeitung von gemeinsamen Informationsräumen findet selten in vorhandenen Groupware-Applikationen statt. Wir haben in unseren Betrachtungen festgestellt, daß für die Entwicklung von solchen Anwendungen ein geeignetes Datenmodell für die interne Darstellung der Daten notwendig ist und haben den Einsatz von XML diskutiert. Eine neue Anforderung an das Datenbankmodell ist dadurch entstanden: Die Speicherung von XML-Dokumenten mit möglichst geringem Aufwand. Die Generierung von XML-Dokumenten sowie die Ausführung von Operationen auf XML-Dokumenten sind ebenfalls Anforderungen an das Datenmodell.

Wir haben festgestellt, daß Awareness bei dieser Art von Anwendungen ein sehr wichtiges Merkmal ist, genauso wie die Kontrolle der Nebenläufigkeit ohne große Behinderung der Arbeit der Benutzer. Die Forderung nach der Konsistenz einerseits und der Bereitstellung von Awareness-Informationen andererseits führt zu einem Zusammenhang zwischen Awareness und der Kontrolle der Nebenläufigkeit.

[17] Ausgenommen sind Tools f¨ur synchrone Kommunikation.

Kapitel 3

Modell

In diesem Kapitel wird das Modell vorgestellt. Für die in der Aufgabenstellung besprochenen Punkte werden Lösungen innerhalb dieses Modells behandelt. Dabei wird hier lediglich versucht, eine grobe Gesamtsicht über dieses Modells und seine einzelnen Komponenten zu geben. In Kapitel 4 werden die einzelnen Submodelle vorgestellt.

3.1 Gesamtmodell und Entwicklungsschritte

Die grundlegende Idee ist die Bereitstellung von Informationen in einer einfachen hierarchischen Struktur und die Definition einer intuitiven *Operationssemantik* auf diesen Daten. Inhalt und Visualisierung werden voneinander getrennt, so daß Anwendungen in der Lage sind, dieselben Daten unterschiedlich darzustellen. Dabei soll die konkrete Visualisierung der Daten auf die Anwendungsebene verlagert werden. Durch eine navigierende Schnittstelle wird die Lokalisierung eines Benutzers innerhalb des Informationsraums, und damit die Bereitstellung von Awareness-Informationen, ermöglicht; der Cursor, mit dem er Operationen ausführt gibt seinen Aktivitätsbereich an. Sein Sichtbarkeitsbereich definiert sich durch die sichtbaren Objekte in seinem Fenster.

Die Schwierigkeit bei der Entwicklung eines solchen Modells liegt in der Tatsache, daß man einerseits Teilmodelle identifizieren und isolieren kann, für die es auch partielle Lösungen gibt und deren Anforderungen man nicht vermischen sollte, andererseits große Abhängigkeiten zwischen den Teilmodellen existieren [64, 65, 187]. So lassen sich vier relativ disjunkte Teilmodelle herausarbeiten:

- das Interaktionsmodell mit Navigation und Cursor
- das Datenmodell mit baumartiger Struktur des Datenraums
- das Visualisierungsmodell mit Browser beim Client
- das Awarenessmodell mit Kontrolle der Nebenläufigkeit.

Die vier Teilmodelle, speziell in dieser Reihenfolge, entsprechen auch in etwa der historischen Entwicklung der Fachgruppe Datenbanken an der Universität Kassel. Diese Entwicklung ist wiederum eine natürliche Progression von einem proprietären single-user Editor [221, 149] hin zu einem Web-basierten multi-user Tool auf der Basis von XML [210], das sich auch für die synchrone Gruppenarbeit einsetzen läßt [19, 79] (siehe Abschnitt 4.1.1).

Beispiele für die Abhängigkeiten sind etwa Navigationsoperationen im Interaktionsmodell, die den Traversierungsschritten und Manipulationen von Knoten und Ästen im Datenmodell 1:1 entsprechen. Genauso gibt es eine sehr enge Verflechtung zwischen dem Fokus/Nimbus-Ansatz der Awareness (siehe Abschnitt 4.3), dem Cursor aus dem Interaktionsmodell, sowie dem Darstellungsfenster aus dem Visualisierungsmodell.

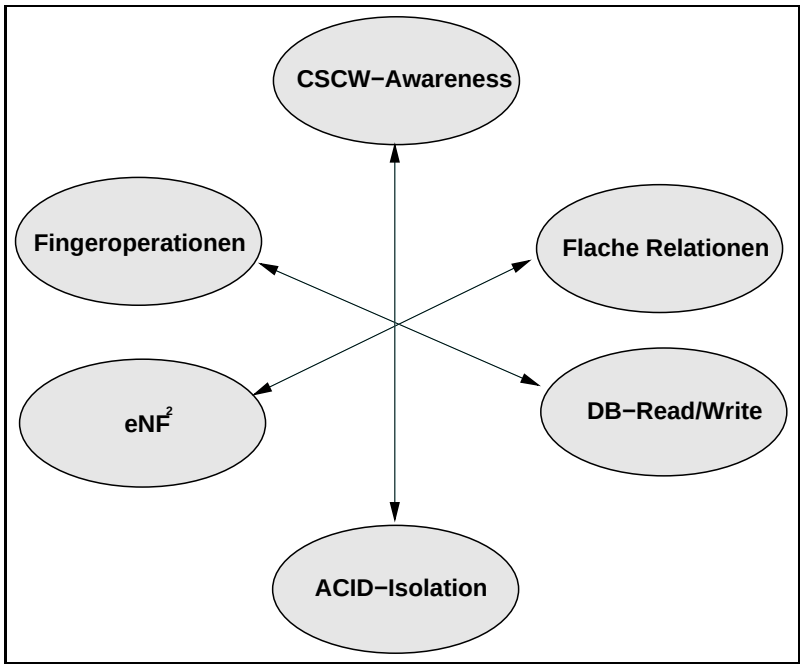


Abbildung 3.1: Gegens¨atzlichkeiten der verschiedenen Teilmodelle

Trotzdem sind die Submodelle weitgehend unabhängig, denn der baumartige Datenraum aus dem Datenmodell ließe sich auch mit XQuery [216] und XPath [206] deklarativ statt navigierend bearbeiten, genauso könnte die Navigation auf flachen relationalen Tabellen statt auf hierarchischen Strukturen erfolgen. Des weiteren könnte Awareness durch *shared whiteboards* mit frei beweglichen Pointern erfolgen und nicht syntaxgesteuert durch Cursor auf Datenobjekten [94, 42].

Die Auswahl eines geeigneten Interaktions- und Datenmodells gilt dabei als wichtigste Entscheidung, aus der geeignete Visualisierungs- und Awarenessmodelle abzuleiten sind. Für die Auswahl eines passenden Interaktions- und Datenmodells werden folgende Anforderungen aufgestellt [91, 174]:

- Benutzer sollen durch intuitive Operationen (Navigieren) auf der Benutzeroberfläche die Datenbank bearbeiten können [201, 200].
- Die Operationssemantik soll die Lokalisierung der Präsenz von Benutzern im gemeinsamen Informationsraum vereinfachen [61].
- Das Abbilden von dem, auf der Anwendungsebene verwendeten Datenmodell und dem Datenmodell selbst muß schnell und einfach erfolgen. Da Daten zwischen dem Server und den Anwendungen durch das Aktualisieren der Anzeige ständig ausgetauscht werden, ist dies von zentraler Bedeutung.
- Das Abbilden der anwendungsspezifischen Funktionen auf Operationen des Modells soll möglichst einfach bleiben.

Die folgenden Abschnitte gehen auf die Teilmodelle im einzelnen ein, wobei die Bezüge zu den Entwurfsentscheidungen der anderen Subkomponenten zwangsläufig zur Sprache kommen.

3.2 Interaktionsmodell

Versteht man synchrone Gruppenarbeit als das gemeinsame Arbeiten auf strukturierten Datenräumen, dann bietet sich ein Interaktionsmodell an, das Navigation besonders unterstützt. Alternative Modelle wären videogestützte Konferenzmodelle oder gemeinsames pixelorientiertes Malen auf einem Whiteboard [86].

Letztere Modi erscheinen für die gestellten Aufgaben wenig geeignet, weil Navigation hier auch das Eintauchen in die Datenräume [19, 223], das Anfassen, Ändern und Entfernen (Cut & Paste) von Datenobjekten einschließen soll.

Dies unterscheidet den Modus auch vom Browsen im Web, das zwar eine linkgesteuerte Navigation von Seite zu Seite und auch innerhalb einer Seite mittels interner Links ermöglicht, trotzdem nur eine passive Interaktion nach dem Prinzip „*look - don't touch*" darstellt. Geht man davon aus, daß Koordinationsaufgaben auf der Grundlage computergestützter Materialien erfolgen, also z. B. auf der Basis von Stundenplänen, Ablaufplänen, PERT- und Gantt- Diagrammen, Materialflußplänen usw., die in der Regel eine baumartige Struktur haben, dann macht es Sinn, auf die wohldefinierten Elemente dieser Pläne und Strukturen zu verweisen, und diese Elemente zur Einheit von Modifikationen und Ersetzungen zu machen.

Dazu führen wir den Begriff des Cursors ein, der auf ein Datenelement (Datenobjekt) zeigt, nennen ihn aber hier *Finger*, um die Verwechslung mit dem mausgesteuerten Cursor (Pfeil) der graphischen Anzeige oder einer Texteingabe zu vermeiden [220].

Dabei sei klargestellt, daß die Navigation eines Fingers nicht notwendigerweise das manuelle Weitersetzen des Datencursors an die gewünschte Position ist. Vielmehr kann es durchaus auch das Setzen mittels einer Suchfunktion sein, wie man es von „Suchen-und-Ersetzen" bei Textverarbeitungssystemen her kennt. Ähnlich bieten Suchmaschinen eine Navigation mit Sprüngen an, bei der Suchresultate wieder als Seiten dargestellt werden. Ein weiteres Beispiel für den erweiterten Navigationsbegriff sind neuerdings XQuery [216] und XPath[206] mit einer Mischung aus Pfadausdrücken und Lokalität im (DOM-)Datenbaum [208] (dort context genannt), oder DOM Traversal [209].

Navigation wird aber vor allem vom normalen Benutzer auf intuitive Weise in vielen alltäglichen Anwendungen verwendet. Für Navigationsoperationen auf geschachtelten Strukturen (isomorph: Bäume) werden die, in Abbildung 3.2 dargestellten Operationen mit Symbolen und Bezeichnungen verwendet.

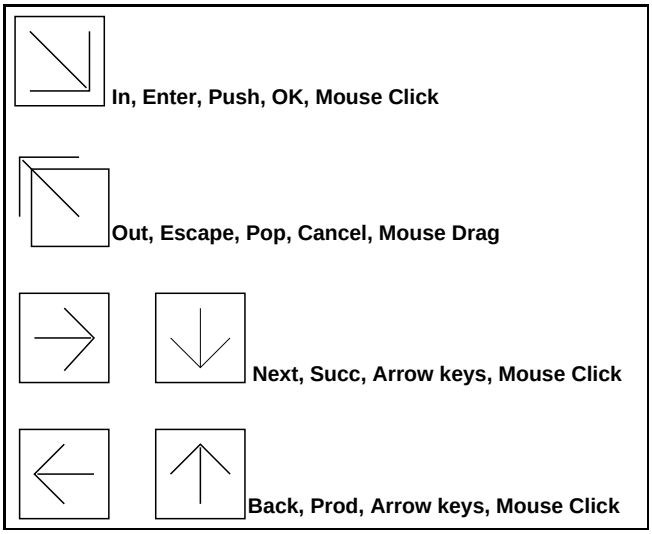


Abbildung 3.2: Generische Navigationsoperationen

Generell kann man sicherlich von einer Renaissence der Navigation sprechen, die ja im Zeitalter der hierarchischen Datenbanken, man denke an die weitverbreiteten IMS-Systeme, üblich war und die jetzt durch das WWW wieder aktuell ist [201]. Dazu kommt, daß die aufkommende Verbreitung mobiler Endgeräte, etwa Mobiltelefone und PDAs, mit unzulänglichen Eingabe- und Anzeigefähigkeiten den navigierenden gegenüber einem deklarierenden Modus fördert [223].

Um das Argument mit einem Beispiel zu unterstützen, denke man an zwei Geschäftspartner, die sich bezüglich der Uhrzeit für eine Bahnfahrt, sagen wir von *Kassel* nach *Frankfurt* zwischen 8 und 9 Uhr, synchron in einer CSCW-Anwendung verständigen. Hier bietet sich das gemeinsame Navigieren ($Reisen \rightarrow Bahn \rightarrow Abfahrt \rightarrow Kassel \rightarrow Wilhelmshöhe \rightarrow 9Uhr \rightarrow \ldots$) zu einer virtuellen Abfahrtstafel des Bahnhofs Kassel-Wilhelmshöhe eher an als eine textuelle Eingabe der Abfrage

$SELECT\ Abfahrtszeit,\ Zugart,\ Gleis\ FROM\ Fahrplan$
$WHERE\ Abfahrtsort = "Kassel - Wilhelmshöhe"\ AND$
$Abfahrtszeit > "8:00"\ AND$
$Abfahrszeit < "9:01"\ AND$
$Zielort = "Frankfurt - Hbf"\ AND\ \ldots$

In Abschnitt 4.1.1 beschreiben wir im Detail, was wir unter dem Fingerparadigma verstehen und welche navigierenden Operationen im Modell angeboten werden.

3.3 Datenmodell

Wie im Abschnitt 3.2 oben ausgeführt, ergibt sich für die Aufgabenstellung der synchronen Gruppenarbeit auf strukturierten Datenräumen mit Betonung einer navigierenden Interaktion zwangsläufig ein Datenmodell, bei dem die Objekte baumartig (hierarchisch) angeordnet sind. Der Begriff „baumartig" deutet dabei an, daß das Datenmodell kein Baum im strengen Sinn, sondern ein gerichteter Graph (mit Zyklen) ist. Dies entspricht der Linkstruktur im Web und gilt auch für XML-Dokumente, die Verweise enthalten. Trotzdem dominiert die Baumsicht mit einer Wurzel zum Einstieg in ein Dokument oder eine geschachtelte Tabelle, bzw. mit Blättern, die *parsed character data* im Fall von XML und die *atomare Felder* im Fall der geschachtelten relationalen eNF^2-Tabellen entsprechen [222].

Alternative Datenmodelle, etwa ein relationales Datenmodell mit Tabellen in erster Normalform (flachen Tabellen) lassen sich kaum vorstellen. Speziell die Navigation auf großen flachen Tabellen ist unattraktiv, man denke an eine Tabelle aller Abfahrtszeiten eines Bahnhofs oder gar innerhalb Deutschlands oder eine Tabelle aller lieferbaren Bücher, usw. Was mit einer deklarativen Abfrage und Resultatmengenbildung im relationalen Modell gut funktioniert, ist für den visuellen Ansatz mit manueller Steuerung weitgehend ungeeignet.

Andere Datenräume, etwa ein Dokumentenraum auf der Basis von XML oder das real existierende Web, erweisen sich bei näherem Hinsehen nur als isomorphe Spezialfälle des oben genannten baumartigen Datenmodells. Im folgenden werden wir diese Isomorphie weiter betonen, beschränkt auf drei Sichtweisen:

- geschachtelte relationale Tabellen (eNF^2-Modell)
- XML-Dokumentenmodell
- abstraktes Baummodell mit inneren Knoten, Blättern, Ästen und Pfaden (konkretisiert etwa durch das Document Object Model).

Die Austauschbarkeit der Sichtweisen ist an den letzten beiden Beispielen am augenfälligsten, ist doch der DOM-Baum nur eine geparste Realisierung von XML-Daten im Hauptspeicher. Genauso leicht läßt sich eine eNF^2-Tabelle als XML-Dokument speichern und umgekehrt, vorausgesetzt das Dokument hat entweder eine

variantenfreie Struktur und genügt damit einem eNF^2-Schema (siehe Programm 5.10, vgl. [222].), oder das eNF^2-Schema hat variante Strukturen (vgl. [113]), worauf wir hier nicht weiter eingehen wollen.

3.3.1 eNF^2-Datenmodell

Die erste Vorschrift des *Relationalen Datenmodells* besagt, daß sich die Daten einer Relation immer in der *ersten Normalform* (1NF) befinden müssen, d.h. der Wertebereich eines jeden Attributes besteht lediglich aus elementaren Werten. Der Wert eines Attributes darf damit nicht selbst wieder eine Menge, eine Relation oder ein anderes strukturiertes Objekt sein. Das hat bei vielen Datenbankanwendungen einen großen Nachteil, vor allem wenn die zugrunde liegenden Daten in natürlicher Weise hierarchisch gegliedert sind. Zur Speicherung von hierarchisch geschachtelten Daten ist eine Erweiterung des Relationalen Datenmodells nötig [176].

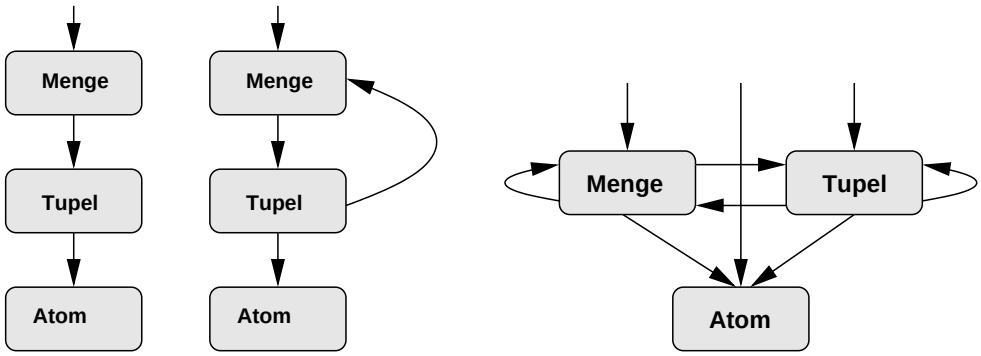
Um solche komplexen Daten mit dem Relationalen Datenmodell zu verwalten, müssen die Daten auf verschiedene Tabellen aufgeteilt und die Tabellen durch Fremdschlüssel miteinander verbunden werden. Doch diese Lösung macht die Anwendung komplexer und langsamer dadurch, daß die Daten zur Laufzeit aus mehreren Tabellen zusammengestellt werden müssen (Joins).

Die bessere Art zur Verwaltung solcher Daten ist, das Datenmodell so zu erweitern, daß das Speichern und die Bearbeitung von komplexen Daten in ihrer natürlichen Form ermöglicht wird. Diese Erweiterung beruht einfach darauf, auf die erste Normalform zu verzichten und zu erlauben, daß die Daten einer Relation wiederum Relationen enthalten können. Dieses Modell wird als *das geschachtelte Relationale Datenmodell* oder *NF^2-Datenmodell* (als Kurzform für *Non First Normal Form*) bezeichnet [202].

In einer NF^2-Relation wird für die Komponenten der Tupel zugelassen, daß sie selbst wieder Relationen sind. Also wird auf die Forderung nach der Einfachheit der Domänen verzichtet. Die Anordnung der strukturierenden Elemente, nämlich der Relationen, wird aber vorgegeben: Eine NF^2-Relation ist eine Menge von Tupeln, deren Komponenten atomar sind oder selbst wieder Relationen. Dabei ist zunächst nur von einer einfachen „Schachtelung“ der Relationen ausgegangen worden [131], d. h. die Komponenten der Tupel waren atomar oder 1NF-Relationen. Später wurde sie rekursiv angewendet, so daß die Struktur einer NF^2-Relation wie folgt definiert ist: Eine NF^2-Relation ist eine Menge von Tupeln, deren Komponenten atomar oder selbst wieder NF^2-Relationen sind [165, 195].

Durch die Vorgabe, daß eine Relation stets eine Menge von Tupeln ist, ist die Reihenfolge der strukturierenden Elemente (Mengen und Tupel) immer noch sehr eingeschränkt. Gibt man diese Forderung auf, gelangt man zum erweiterten NF^2-Datenmodell. Die strukturierenden Elemente werden als **Konstruktoren** bezeichnet. Relationales, NF^2- und eNF^2-Datenmodell unterscheiden sich also strukturell darin, in welcher

Reihenfolge die Konstruktoren angeordnet werden dürfen. Graphisch ist der Unterschied in Abb. 3.3 dargestellt[1]. In Abb. 3.3(c) wird durch die drei „Einstiegspunkte“ deutlich, daß die Daten des eNF^2-Datenmodells nicht mehr nur Relationen von Tupeln sind, sondern prinzipiell beliebige Strukturen sein können, im Extremfall sogar einfache atomare Objekte.



(a) 1NF (b) NF^2 (c) eNF^2

Abbildung 3.3: Reihenfolge der Konstruktoren

Eine weitere Neuerung des eNF^2-Datenmodells ist die Unterscheidung von verschiedenen Konstruktoren für Sammlungen von gleichartigen Objekten: Neben der Menge (im mathematischen Sinn, d. h. ohne Ordnung und Duplikate) wurden Listen und Multimengen zugelassen. Diese Konstruktoren werden unter dem Oberbegriff Kollektionen zusammengefaßt.

Eine Implementierung des NF^2-Datenmodells wurde z. B. mit DASDBS [54, 55, 147] realisiert, das eNF^2-Datenmodell wurde in AIM-P [50, 52, 162] umgesetzt, in Teilen wurde eine Umsetzung in SQL3 versucht [137].

Hierarchische Daten lassen sich leicht als strukturierte Dokumente speichern und darstellen. Als Auszeichnungssprache (*Markup Language*) bietet sich die Metabeschreibungssprache XML [210] an[2]. Gründe dafür sind:

- XML ist ein W3C Standardvorschlag. Verschiedene Anzeigegeräte werden daher in der Zukunft XML-Dokumente bearbeiten und darstellen können (siehe Abschnitt 1.2.1).
- XML-Dokumente sind reine ASCII-Dokumente und können daher über gängige Standards etwa das HTTP-Protokoll zwischen unterschiedlichen Plattformen ausgetauscht werden.

[1] Diese Darstellung ist aus der im AIM-P Projekt entwickelten Darstellung zum Vergleich der Datenmodelle abgeleitet worden (siehe z. B. [52]).

[2] Dies wurde z. B. in den Anwendungen [196, 98, 116]) bereits ausgenutzt.

- XML ist eine Metabeschreibungssprache. D.h. man kann eine geeignete Sprache für die Darstellung der Daten mit Hilfe von XML definieren.
- Durch Verwendung von XSL-Stylesheets und anderen Mechanismen[3] können XML-Dokumente bei verschiedenen Anwendungen unterschiedlich dargestellt werden [97].
- Verschiedene, auf XML-basierende, Visualisierungssprachen wie SVG [213] und SMIL [214], oder Sprachen zur Beschreibung von grafischen Oberflächen wie XUL [81] und XForms [217], sind schon entwickelt worden, was die unterschiedlichen Möglichkeiten der Visualisierung von XML-Dokumenten bereichert.

XML an sich ist kein Datenmodell, sondern eine Metabeschreibungssprache. Sie definiert einige Regeln, die jedes XML-Dokument einhalten muß, um als ein korrektes, d. h wohlgeformtes XML-Dokument zu gelten. Man kann also XML-Dokumente als hierarchisch strukturierte Datenbankobjekte betrachten. Speziell kann man eine eNF^2-Tabelle als XML-Dokument darstellen und umgekehrt. Durch die Einführung einer Operationssemantik und die Definition von Integritätsbedingungen (z. B. mit XML Schema) kann man XML als Grundlage für ein Datenbankmodell betrachten.

3.4 Visualisierungsmodell

Unter einer Sicht (view) versteht man im Datenbankbereich eine inhaltliche Auswahl von Daten nach einem vom Anwender gewählten externen Schema, das sich vom globalen konzeptuellen Schema ableitet. Im relationalen Datenmodell werden diese Sichten zur Laufzeit aus Basistabellen und anderen Sichten mittels Abfragen jeweils aktuell generiert. Gegebenenfalls können Sichten materialisiert werden, d.h. es existieren Kopien (ggf. veralteter) Daten.

Für die visuelle Auswertung von Daten sind in den letzten Jahren verstärkt Anstrengungen unternommen worden, die selektierten Daten graphisch aufzubereiten. Somit kann man hier von einer Sicht im wörtlichen Sinne des Wortes sprechen. Beispiele wären eine Tabelle als Sicht auf eine Basistabelle des statistischen Bundesamtes, die alle Städte und Gemeinden in Hessen mit über 100.000 Einwohner enthält (Selektion) mit Angabe des Städtenamens und der Ausgaben für Sozialhilfe je Einwohner (Projektion). In einer Visualisierung könnten diese Daten dann auf eine Hessenkarte projiziert werden, wobei die Höhe der Sozialausgaben/Einwohner als Balken dargestellt wird.

Die im Rahmen des Datenmodells angesprochene Isomorphie der Datenrepräsentation, etwa eNF^{2}versus XML versus Baum, wiederholt sich hier. Abbildungen 4.7 und 4.8

[3] Z. B der Aufbau einer grafischen Oberfläche aus einem DOM-Baum. Siehe Beispiel in 5.3.6.

zeigen ein Beispiel mit Projektdaten aus einer Projekttabelle (siehe Abb. 3.4). Die Ansichten der Rohdaten erfolgt als Gantt-Diagramm sowie als geschachtelte Tabelle (siehe das zugehörige XML-Dokument in B.1).

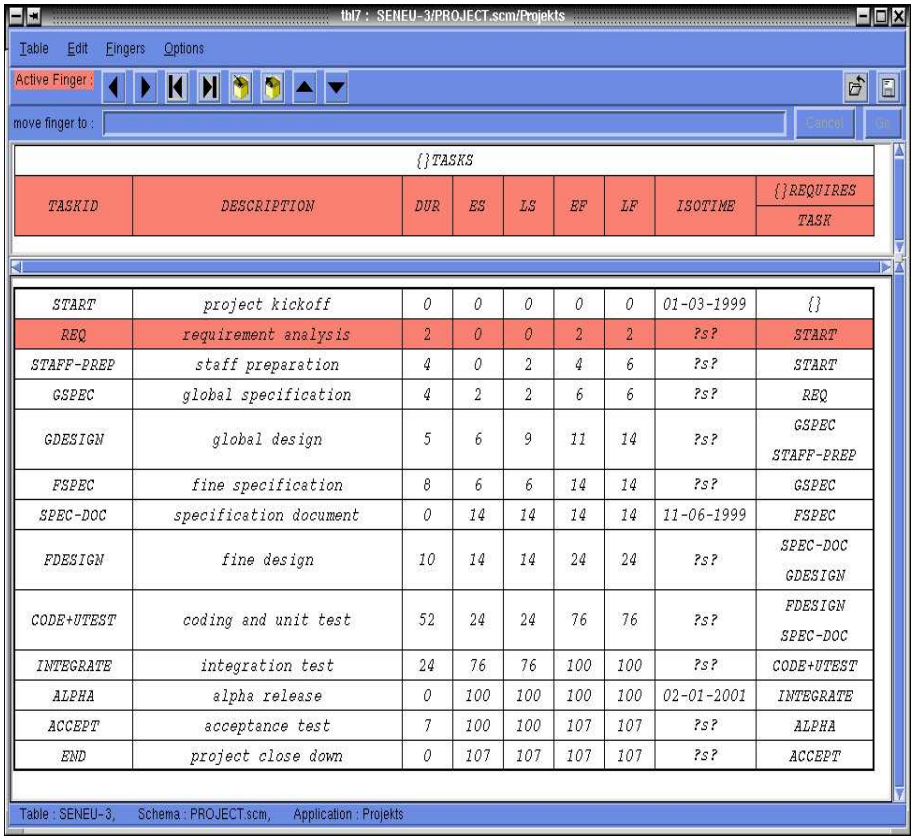


Abbildung 3.4: Jedes Tupel der Menge `TASKS` enth¨alt Daten ¨uber einen Projektabschnitt (Task). Das mengenwertige Attribut `Requires` enth¨alt alle Task's, auf die ein Projektabschnitt warten muß

Für die Aufgaben der synchronen Gruppenarbeit kommt der Visualisierung eine große Bedeutung zu. Erstens soll sie einen möglichst komfortablen und intuitiven Zugang zu den Daten ermöglichen. Zweitens die Funktion der Awareness und die Kontrolle der Nebenläufigkeit unterstützen, ohne den Anwender mit Daten zu überfluten. Drittens soll sie wegen der Bedeutung der Ereignissignalisierung schnellen Bildaufbau bieten. Viertens soll sie weitgehend geräteneutral funktionieren und fünftens möglichst generisch das CSCW-Modell unterstützen.

Da dies zum Teil konfliktäre Ziele sind und die Gestaltung graphischer Schnittstellen weiterhin sehr aufwendig ist, sind Entwurfsentscheidungen hier besonders kritisch.

Aufbauend auf Erfahrungen aus Vorläuferarbeiten [191, 192] bietet es sich zunächst an, eine generische Darstellung in Tabellenform anzubieten, die wiederum das eNF^2-Datenmodell anschaulich visualisiert . In diese Tabellensicht lassen sich wiederum die Finger aus dem Interaktions-Submodell gut projizieren.

Zu beachten ist dabei, daß diese Tabellen in realen Anwendungen sehr groß werden. Daher muß man bei der Visualisierung berücksichtigen, daß nur einen Ausschnitt aus diesem Datenraum gezeigt wird. Im folgenden wird dieser Ausschnitt *Viewport* genannt und entspricht im wesentlichen einem Fenster, das über die Tabelle gescrollt werden kann. Man beachte, daß diese Visualisierungsnavigation unabhängig von der semantischen Fingernavigation ist!

Der Begriff des Viewports ist wiederum verbunden mit dem Awareness-Teilmodell. Offensichtlich kann ein Anwender nur die Dinge beobachten, die sich ihm im Fenster darstellen. In Anlehnung an das weiter unten im Detail vorgestellte Fokus/Nimbus-Modell von *Rodden* und *Benford* [157, 30] bezeichnen wir den Viewport auch als Fokus („was im Licht ist"), die Finger aus dem Interaktionsmodell dann als Nimbus („wo man agiert").

Neben der „generischen" Visualisierung als Tabelle sollen weitere Darstellungen möglich sein, wobei sogar vorstellbar ist, daß in einer CSCW-Anwendung verschiedene Teilnehmer einer Gruppenanwendung verschiedene, ggf. an ihre Endgeräte angepaßte Darstellungen des gemeinsamen Datenraums haben. Dies schließt einfache (Pixel-)Kopien eines serverseitig gehaltenen Bildes im Sinne von SharedX [78] aus, zumal Viewport und Finger der Anwender individuell setzbar sind.

Zuletzt sei darauf hingewiesen, daß für die Entwicklung von Gruppenanwendungen, speziell auch wenn sie spontane Teilnahme und eine lockere Kooperation der Anwender vorsehen, das Internet mit den Standardbrowsern für das Web die universelle Schnittstelle geworden sind. So erfreulich diese Vereinheitlichung auch ist, so wenig geeignet ist das Web mit dem zustandslosen Protokoll HTTP, das dem „connect-get-close-Paradigma" folgt, für interaktive Anwendungen.

Auch der konsequente Übergang von HTML auf XML und der Einsatz von Stylesheets für die individualisierte Anzeige lösen zunächst nicht das Problem der fehlenden Interaktion und der fehlenden Möglichkeit, Updates im Datenraum, etwa aufgrund konkurrierender Aktivitäten anderer Anwender, beim Client in der Datensicht zu aktualisieren (Push-Modell versus übliches Pull-Modell).

Im nächsten Kapitel wird gezeigt, wie dieser Problematik durch Einführung virtueller Clients (VClient) im Server begegnet wird und welche minimalen portablen Codefragmente im Browser des Anwenders benötigt werden, um synchrone Darstellungen des Datenraums und der Navigation darauf, sowie der Manipulation einzelner Datenwerte darin, zu ermöglichen.

3.5 Awareness-Modell

Wie schon beschrieben, operieren Benutzer auf der grafischen Schnittstelle mit navigierenden Finger-Operationen auf Baumstrukturen. Die sichtbaren Objekte im View-

port eines Benutzers geben seinen Interessenbereich an. Er kann diesen explizit auf unterschiedliche Weisen verändern, z.B indem er das Fenster vergrößert, oder zu einer anderen Stelle im Dokument scrollt. Der Bereich, in dem der Benutzer zu einem gegebenen Zeitpunkt agiert, ist eindeutig durch die Position seines aktiven Fingers bestimmt[4]. Aus der Benutzersicht schafft dieses Modell die Möglichkeit, Awareness-Informationen auf einfache und intuitive Weise anzuzeigen, ohne die Benutzer mit überflüssigen Awareness-Nachrichten zu irritieren.

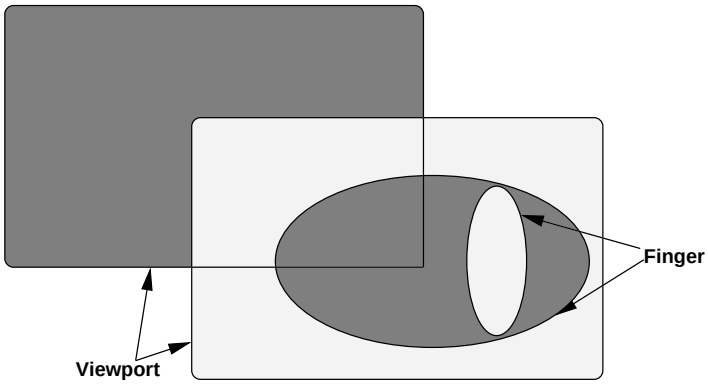


Abbildung 3.5: Fokus und Nimbus zweier Benutzer

Für die Berechnung der Interessenbereiche der Benutzer und die Bestimmung, welche Benutzer einander sehen, bzw. ob sie über die Auswirkungen einer bestimmten Operation benachrichtigt werden, wird ein ähnliches Modell wie das Fokus/Nimbus-Modell [30, 157] entwickelt. Im vorliegenden Modell werden die Begriffe Fokus und Nimbus mit der folgenden Bedeutung verwendet:

Definition 3.1 (Fokus)
Der Fokus eines Benutzers ist die Menge aller Datenbankobjekte des gemeinsamen Informationsraums, die ganz oder teilweise im Viewport des Benutzers sichtbar sind.

Definition 3.2 (Nimbus)
Der Nimbus eines Benutzers ist die Menge aller Datenbankobjekte, auf die der aktive Finger des Benutzers zeigt.

Dabei ist zu erwähnen, daß das Navigationsmodell Fingersyntax nur auf ganze, ggf. komplexe, Objekte vorsieht. Das Zeigen auf eine zusammenhängende Teilmenge, oder gar simultan auf disjunkte Elemente, ist nicht realisiert, wäre aber denkbar.

[4] Tats¨achlich ist diese Form der Interaktion aus den Verhaltenswissenschaften wohlbekannt und heißt dort deiktisches Handeln (*deictic reference*, „do-it-where-I-am-looking"). Psychologische Untersuchungen lassen vermuten, daß sich unser Gehirn deiktischer Strategien bei motorisch-sensorischen T¨atigkeiten bedient und sein Kurzzeitged¨achtnis in Intervallen von weniger als einer Sekunde mit deiktischen Aktionen auffrischt [27].

Im Visualisierungsmodell realisieren wir das „Zeigen" des Fingers auf ein Datenobjekt durch Einfärben des Datenobjekts. Andere Darstellungen durch Blinken, Umrandung oder Unschärfe der Umgebung wären möglich.

3.6 Kontrolle der Nebenläufigkeit

Synchrone Gruppenarbeit auf strukturierten Datenräumen ist eine Form der Kollaboration, die sich für Verhandlungen, Projektplanung, Korrekturen gemeinsamer Dokumente, Koordinierungsaufgaben usw. besonders eignet [154]. In Abschnitt 1.2.1 wurden hierzu Beispiele aufgeführt.

Dabei werden von den Beteiligten der Gruppe Entscheidungen auf der Basis angezeigter Daten getroffen, die man als Transaktionen im üblichen Sprachgebrauch bezeichnen könnte. Transaktionen werden ausgehandelt, wobei bei offenen Verhandlungen Vorschläge, Forderungen, Angebote und Alternativen für alle frei einsehbar auf den Tisch kommen sollten. Deshalb kommt der Awareness bei der rechnergestützten Umsetzung eine entscheidende Rolle zu. Wie unten ausführlicher aufgezeigt, ersetzt sie die Isolation aus der Definition der klassischen Datenbanktransaktion.

Gleichzeitig erwarten die Gruppenmitglieder, daß Festlegungen und Zusagen dokumentiert und eingehalten, bzw. umgesetzt werden. Üblicherweise steht am Ende solcher Verhandlungen die übereinstimmende Zustimmung zu der gerade verhandelten Angelegenheit, häufig per Handschlag, formal dann per Unterschrift, manchmal mit einer Frist für die Möglichkeit des Widerrufs. Die Parallelen zur Persistenz (*durability*) und zur Atomizität (*atomicity*) aus der gerade genannten DB-Transaktionsdefinition drängt sich deshalb auf.

Das strenge, traditionelle Transaktionsprinzip bei Datenbanksystemen muß sich also mehr den menschorientierten Interaktionsformen öffnen. *Korth* fordert [119, S. 4]: „*In this environment, the metric of TPS, transactions per second, is not the issue. Rather the quantity to be maximized is human information transfers per second (HITS)*". Damit wandeln sich nach *Korth* die ACID Eigenschaften in der folgenden Art [119, S. 4]:

- atomar → strukturiert, aber flexibel
- konsistent → meistens konsistent, mit klar markierten Ausnahmen
- isoliert → kooperativ, verhandelbar
- dauerhaft → nachweisbare Buchungsspur der Verantwortung

Die interessanteste Änderung ist die Aufgabe der Isolation zugunsten einer Signalisierung von Nebenläufigkeit (*concurrency awareness*) [88, 91, 38]. Benutzer können sich gegenseitig *beobachten* und bekommen die Veränderungen der Daten im gemeinsamen Informationsraum mit. Dies schafft eine Verbindung zwischen dem Awareness-Modell

und dem Verfahren für die Nebenläufigkeitskontrolle. Die Dauerhaftigkeit von Ergebnissen und ein global konsistenter Zustand erscheinen dagegen immer wünschenswert, auch wenn Ergebnis und Zustand ggf. durch Verhandlungen erreicht werden und nicht durch vorprogrammierte Operationen.

Der Verzicht auf Mechanismen für die Kontrolle der Nebenläufigkeit ist allein durch die Bereitstellung von Awareness-Informationen nicht angemessen. Bürger behauptet nach einem Zitat von [225, S. 837]: „*[...] Durch die damit erzeugte Awareness können die Benutzer sich gegenseitig bei der Benutzung des Objekts koordinieren, ohne daß eine Kontrolle durch den Rechner notwendig wäre [...]*" [42, S. 44]. Dies setzt aber voraus, daß die Benutzer den Umgang mit der Software beherrschen und daß die Benutzer immer in einem homogenen Team arbeiten und sich gegenseitig berücksichtigen. In vielen Situationen der synchronen Gruppenarbeit ist dies aber nicht der Fall. Eine spontan gebildete Gruppe, die sich im WEB bildet und auf gemeinsame Daten zugreift, bildet kein solches homogenen Team.

Borghoff schreibt [38, S. 226]: „*. . . In CSCW-Systemen kann es legitim sein, daß zwei Schreiber gleichzeitig Änderungen am selben Datenobjekt vornehmen. Erstellt werden soll nun kein Ergebnis, bei dem der eine oder der andere Schreiber erfolgreiche Modifikationen durchführen kann, sondern ein Ergebnis, bei dem beide Schreiber erfolgreich modifizieren. Vom CSCW-System wird nun gefordert, daß man beide Schreibvorgänge bzw. Modifikationen im Objekt erkennt*".

Gerade wegen Awareness kann dies nicht erlaubt werden, da jeder Benutzer zur Zeit der Ausführung seiner Operation von einem anderen Zustand ausgeht. *Greenberg* [88] bestätigt dies und zeigt sogar, daß durch Awareness selbst die strikte Serialisierbarkeit [67, 34] zwar einen konsistenten Zustand der Datenbank gewährleistet, aber oft *unbeabsichtigte Ergebnisse* liefert, da die Benutzer von dem Inhalt in ihren Viewport für die Ausführung von Operationen ausgehen, der manchmal mit dem Datenbankinhalt nicht übereinstimmt. Das Verfahren für die Nebenläufigkeitskontrolle soll also diese unbeabsichtigten Ergebnisse ausschließen können.

Die vorliegende Arbeit verwendet daher den Begriff „*visuelle Transaktion*" für die Zusammenfassung von Aktionen eines an der Gruppenarbeit beteiligten Mitglieds (siehe Definition 4.11). Alternativ könnte man den Ablauf der Schritte auch als „Verhandlung" (*negotiation*) oder einfach „Handel" (engl. *deal*) bezeichnen.

3.6.1 Visuelle Transaktionen versus ACID-Transaktionen

Für die grundlegende Behandlung visueller Transaktionen bietet es sich an, zunächst auf Begriffe und Vorgehensweisen aus der Datenbankwelt zurückzugreifen, die hier kurz wiederholt werden[5]. Ausgehend von den Gemeinsamkeiten können dann die Besonderheiten herausgearbeitet werden.

[5] Die Darstellung wurde dem Kapitel 4 aus [222] entnommen.

Transaktionen in der Datenbankwelt sind Folgen von Einzelschritten (Operationen), die mittels einer Anfangsmarke (BEGIN-TRANSAKTION) und einer Endmarke (COMMIT oder ABORT) geklammert werden. Die Reihenfolge und Art der Schritte wird dabei als extern vorgegeben angenommen.

Zur Realisierung des Transaktionskonzepts verlangt man die Einhaltung der sog. ACID-Eigenschaften.

- *Atomicitiy* (atomares Verhalten): „Alles-oder-Nichts-Prinzip", d.h. eine Transaktion wird vollständig durchgeführt bis zum COMMIT oder bei Abbruch (ABORT) ganz ungeschehen gemacht.
- *Consistency* (Bewahrung der Konsistenz, der logischen Integrität): d.h. jede mit COMMIT abgeschlossene Transaktion bewahrt per Definition die Konsistenz der Datenbasis.
- *Isolation*: schützt den einzelnen Benutzer vor den Auswirkungen des Mehrbenutzerbetriebs, d.h. die Ausführung der Transaktion erfolgt, wie wenn der Benutzer alleine wäre; dies wird meist durch Sperren von Datensätzen erreicht.
- *Durability* (Beständigkeit der Daten): garantiert, daß mit COMMIT abgeschlossene Transaktionen auch über Systemabstürze und Datenträgerverlust hinaus ihre Gültigkeit dadurch behalten, daß die gemachten Modifikationen sicher gespeichert sind.

Zur Realisierung des Transaktionsprinzips muß man eine Steuerung der Nebenläufigkeit (*concurrency control*) und Maßnahmen zur Wiederherstellung konsistenter Zustände (*recovery*) einführen.

3.6.2 Sicherung vor Datenverlust (Recovery)

Recovery-Mechanismen bieten Schutz vor Datenverlust und Verfälschung (Inkonsistenzen) bei verschiedenen Arten von Versagenssituationen, etwa

- Transaktionsabbruch (z. B. ungenügende Deckung bei Zahlungstransfer; Vorgang muß rückgängig gemacht werden)
- Systemfehler („Systemabsturz", Fehler im DBMS, Betriebssystemfehler, usw; flüchtige Daten im Hauptspeicher sind weg)
- Verlust des Datenträgers („*Headcrash*", Brand, Diebstahl usw.)

Für synchrone Gruppenarbeit erscheint Recovery zunächst unwichtig, wenn man voraussetzt, daß das zugrundegelegte Speichersystem (ein klassisches DBMS oder ein

Datenrepository) sich um Fragen der Persistenz und Ausfallsicherheit kümmert. Allerdings erkennt man schnell, daß der sicheren, verfälschungsgeschützten Speicherung von Daten und Botschaften der beteiligten Teilnehmer eine wichtige Rolle zukommt. Als Beispiel denke man an verteilte Abstimmungen, an Angebote in Versteigerungen, an empfangene Aufforderungen und Mitteilungen zu Ereignissen, deren Existenz je nach Interessenlage vorgespiegelt oder fälschlich geleugnet werden könnte.

Noch wichtiger ist allerdings die Möglichkeit, Einzelschritte rückgängig zu machen (*undo*) und Schritte zu wiederholen (*redo*). Einerseits ist dies in der Gruppenarbeit, die ja überwiegend von menschlichen (und daher fehleranfälligen) Akteuren betrieben wird, notwendig, wie etwa von *Prakash* und *Knister* beschrieben [150]. Andererseits sind undo/redo-Operationen die Grundlage der Recovery um

- mittels undo-Operationen die Datenbank zurückzufahren und damit Auswirkungen noch nicht abgeschlossener Transaktionen ungeschehen zu machen oder
- mittels redo-Operationen die Datenbank vorwärtszufahren und damit verlorene Auswirkungen bereits abgeschlossener Transaktionen wieder herzustellen („*repeat history*").

Allerdings bemerken *Prakash* und *Knister*, daß diese Schritte in CSCW-Anwendungen problematisch sein können: „*...it may not make sense to undo one user's changes without undoing the other users' changes. In this case, there are dependencies between the changes made that neeed to be taken into account during an undo*." [150, S. 273].

Generell muß man fordern, daß von anderen Teilnehmern oder systembedingt ausgelöste undo- und redo-Operationen, die dazu führen, daß dargestellte Objekte und Werte plötzlich verschwinden oder unmotiviert auftauchen, möglichst zu vermeiden sind. Das ist jedoch für ein CSCW-Modell wesentlich schwieriger durchzusetzen als für eine Datenbank, da CSCW-Anwendungen prinzipiell auch das Lesen (Beobachten) von noch nicht abgeschlossenen Transaktionen zulassen.

Ein Ausweg ist, diese undo-redo-Aktionen auch visuell explizit zu machen, etwa indem ein virtueller „Systemschiedsrichter" eingeführt wird, der wie ein zusätzlicher Teilnehmer agiert und korrigierend eingreift. Das Grundprinzip ist hier auch wieder den Datenbanken entliehen, wo inverse Operationen für ein undo als „normale" Operationen ins *Write-Ahead-Log* (WAL) geschrieben werden und ihrerseits auch wieder korrigierbar sind.

3.6.3 Kontrolle der Nebenläufigkeit in der Datenbankwelt

In der Datenbankwelt verlangt der Mehrbenutzerbetrieb die Abschirmung der einzelnen Transaktion vor Auswirkungen des quasi-parallelen Betriebs (vgl. ACID-Eigenschaften oben). Ziel ist also die Isolation der Anwender. Dies wird erreicht durch Kontrolle der Nebenläufigkeit.

Im Fall der Gruppenarbeit ist das Gegenteil gewünscht. Teilnehmer an einer synchronen Arbeitssitzung sollen sich mittels Awareness gegenseitig koordinieren. Unerwünscht sind nur die Auswirkungen des parallelen Betriebs, die aufgrund fehlender Informationen, etwa wegen veralteter Daten, versteckter Änderungen, inkonsistenter (Phantom-)Anzeigen usw. den einzelnen zu nicht revidierbaren Aktionen veranlassen, die er so nicht ausgeführt hätte, wenn er den aktuellen (korrekten) Zustand des Systems gekannt hätte.

Betrachtet man zunächst das Standardbeispiel der Sitzplatzbuchung in einer Datenbankanwendung. Dabei wird zunächst die Anzahl verfügbarer Plätze gelesen. Ist noch ein Sitz frei, wird die Anzahl freier Plätze um einen Platz verringert und der neue Wert wird zurückgeschrieben. Zwei verzahnt ablaufende Transaktionen T_1 und T_2 können dabei ein falsches Ergebnis liefern (z. B. ein Platz noch frei und jede Transaktion will einen Platz), wenn beide den selben Wert lesen und denselben (nur) um eins (statt um zwei) verringerten Wert zurückschreiben.

Dieses Ergebnis wäre nicht eingetreten, hätte man die Transaktionen T_1 und T_2 hintereinander (seriell) ablaufen lassen. Bei der Ablauffolge (*schedule*) $< T_1, T_2 >$ hätte T_1 den Platz bekommen, bei der Folge $< T_2, T_1 >$ hätte T_2 ihn bekommen. Damit kann man aber jetzt ein Korrektheitskriterium für den Mehrbenutzerzugriff definieren: *Eine nebenläufige Ablauffolge S von Operationen heißt serialisierbar, wenn es eine serielle Ablauffolge S' gibt, die in ihren Auswirkungen äquivalent zu S ist.*

Allerdings gibt es zwei Formen der Serialisierbarkeit: die *Sichtserialisierbarkeit* und die *Konfliktserialisierbarkeit*.

Die Äquivalenz zweier Folgen bei Sichtserialisierbarkeit ist definiert durch die beiden Bedingungen: Jede der Leseoperationen in den Folgen liest Datenwerte, die von denselben Schreiboperationen in beiden Folgen erzeugt wurden, und die letzte Schreiboperation für jedes Datenobjekt ist die gleiche in beiden Folgen.

Bei der Konfliktserialisierbarkeit betrachtet man nur sog. konfliktäre Operationen, das sind alle Lese-/Schreiboperationen zweier Transaktionen auf demselben Datensatz, darunter mindestens eine Schreiboperation. Die anderen Operationspaare gelten als kompatibel und kommutieren, worauf viele undo-Verfahren zur erweiterten Serialisierbarkeit beruhen (vgl. etwa [164]).

Ein bekanntes Resultat ist, daß Sichtserialisierbarkeit mächtiger ist als Konfliktserialisierbarkeit, aber in der Praxis nicht effizient genug garantiert werden kann.

Generell gilt, daß Kontrollmechanismen, die nur serialisierbare Folgen von Operationen erzeugen, korrekte Nebenläufigkeiten garantieren, wobei zu beachten ist, daß nur die Äquivalenz zu einem seriellen Ablauf gefordert wird. Laufen n Transaktionen $T_1, T_2, \ldots, T_n$ parallel, gibt es $n!$ unterschiedliche serielle Folgen. Die Anzahl der möglichen Ablauffolgen der Operationen ist sogar noch sehr viel größer, da jede Transaktion Ti aus mehreren Operationen besteht. Wie oben bereits betont, wird der

Ablauf der Lese- und Schreiboperationen innerhalb einer Transaktion als gegeben und nicht veränderbar angesehen.

Speziell läßt sich das Testen auf Konfliktserialisierbarkeit auf das Prüfen der Zyklenfreiheit in Abhängigkeitsgraphen (Operationen auf gleichen Datensätzen werden durch gerichtete Kanten verbunden) reduzieren. Ein Problem, das in größenordnungsmäßig n^2 Schritten lösbar ist.

In der Praxis läßt sich korrekte Nebenläufigkeit von Transaktionen im wesentlichen mit den drei Techniken 2-Phasen Sperren (2PL, *2-phase locking*), Zeitmarken und den optimistischen Verfahren, die nur eine nachträgliche Prüfung im Falle eines Konflikts vornehmen, erreichen.

3.6.4 Kontrolle der Nebenläufigkeit f¨ur Gruppenarbeit

Bei der synchronen Gruppenarbeit agieren Benutzer ausgehend von den Objekten in ihrem Viewport und erwarten daher, daß diese mit dem Datenbankinhalt übereinstimmen. Daher ist es wichtig, zwischen dem *Sehen* eines Objekts im Viewport und dem *Lesen* aus der Datenbank zu unterscheiden. Benutzer erwarten, daß das Sehen eines Objekts gleichbedeutend ist mit dem Lesen des Objekts aus der Datenbank. Schreiboperationen müssen aus diesem Grund auch innerhalb einer Transaktion unmittelbar nach ihrer Ausführung propagiert werden, nicht erst nach Abschluß der Transaktion.

Betrachtet man wieder das Standardbeispiel einer Platzreservierung, diesmal jedoch in einer Anwendung für synchrone Gruppenarbeit: Zwei Mitarbeiter A und B einer Firma sollen zusammen eine Schulung besuchen. Es werden mehrere Termine und Orte angeboten, zum Teil sind die Plätze aber knapp.

In einem strukturierten Datenraum werden die Kurse mit Teilnehmerlisten aufgeführt (siehe Tabelle 3.1). Zur Reservierung schreibt man den eigenen Namen mit Abteilungsangabe in die Liste. Die Liste hat nur eine begrenzte Anzahl an leeren Zeilen, in die man sich eintragen kann. A und B telefonieren miteinander und sind in der Kurstabelle navigierend unterwegs. Ihre jeweilige Position und der Zustand des Datenraums werden synchron den Teilnehmern übermittelt.

A und B haben sich nach ausgiebigem Stöbern (*browsing*) auf einen Termin und einen Ort geeinigt (UNIX in Kassel). A liest (sieht) die Teilnehmermenge und stellt fest, daß noch zwei Plätze frei sind. B liest kurz danach auch die Liste und stellt fest, daß A in der Liste aktiv ist. Er sieht, wie A gerade seinen Namen eingetragen hat (Schreiboperation). Darauf trägt auch B seinen Namen ein. Dies sieht wiederum A und beschließt, den *Commit-Knopf* zu betätigen, der die Buchung für ihn abschließt und bestätigt (commit). B sieht ggf. auch dies und macht ebenfalls commit. Aus Datenbanksicht wäre die Ablaufhistorie wie folgt, wobei T der gemeinsame Datensatz, hier die Teilnehmerliste, sei:

$$S:\ R_A(T),\ R_B(T),\ W_A(T),\ R_B(T),\ W_B(T),\ R_A(T),\ R_B(T),\ commit_A, commit_B$$

ID	ORT	Datum	{} Teilnehmer
			TId
UNIX	Kassel	01.10.2002	Hugo
			Otto
			?
			?
SQL	Heidelberg	15.10.2002	Schmitt
			Maier
			?
			?
XML	Heidelberg	15.10.2002	Otto
			Maier
			...
			?

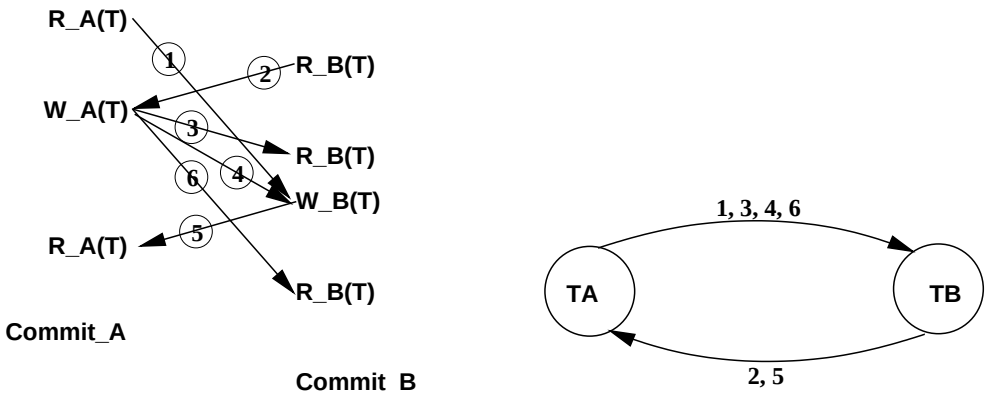
Tabelle 3.1: Kurse mit Teilnehmerlisten in einer eNF^2-Tabelle

Dabei ist zu beachten, daß z. B. die erste Leseoperation nicht explizit von A angestoßen wurde, sondern implizit durch Scrollen zu dieser Position im Dokument ausgeführt wurde. Dies könnte auch einige Zeit zurückliegen. Das System muß trotzdem zusichern, daß der gesehene Zustand des Objekts im Viewport mit dem Inhalt dieses Objekts in der Datenbank übereinstimmt. Hier entspricht also $R(T)$ dem Wert von T nach der letzten zu Ende ausgeführten Schreiboperation auf T.

Die Ablauffolge enthält fünf konfliktäre Paare, die zusammen einen Zyklus ergeben (siehe 3.6). Sie wäre klarerweise nicht serialisierbar. Tatsächlich ließe sich in einem klassischen datenbankbasierten Buchungssystem eine derartige bedingte Buchung (commit nur wenn beide erfolgreich) zweier unabhängiger Reservierungen (etwa eine Urlaubsreise gebucht in zwei getrennten Reisebüros) nicht wirklich garantieren und würde üblicherweise als Buchung von zwei Plätzen durch eine Person in einer Transaktion realisiert.

Um Mißverständnisse zu vermeiden, sollte aber auch vermerkt werden, daß diese Anwendung auch durch zwei getrennte, speziell programmierte Transaktionen, etwa unter Verwendung des 2-Phasen-commit-Protokolls, realisiert werden könnte.

Nehmen wir jetzt an, Benutzer B möchte sich in zwei Kurse am selben Ort eintragen. Er würde sich in SQL nur dann eintragen, wenn er sich auch für XML anmelden könnte. In diesem Fall muß B die Möglichkeit gegeben werden diese beiden Operationen *zusammenzufassen*, z. B. indem er erst die Teilnehmerliste im SQL-, und XML-Tupel explizit *reserviert*, und dann die beiden Operationen ausführt. Kann er sich in einer



(a) Konflikträre Operationen (b) Abhängigkeitsgraph

Abbildung 3.6: Konflikte zwischen den einzelnen Operationen

dieser Listen nicht eintragen, so würde er die ganze Transaktion abbrechen.

Für den CSCW-Fall ist es nun interessant zu untersuchen, in welchen Fällen die nebenläufige Abfolge der obigen Buchungsschritte mit Awareness zu Fehlern führen könnte. Entscheidender Bedeutung kommt dabei offensichtlich dem gemeinsamen commit (oder gemeinsamen abort) zu. Die Sicherheit alle Aktionen der anderen Teilnehmer beobachten zu können und ein commit erst dann abgeben zu müssen, wenn bestimmte Werte eingetragen oder erreicht wurden und andere ein commit gemacht haben. Diese Tatsache heilt das in der Datenbankwelt sonst verbotene „*dirty read*", d. h. das Lesen von Werten, die von Transaktionen geschrieben wurden, die noch nicht mit commit abgeschlossen wurden (und daher potentiell wieder durch ein abort der anderen Transaktion ungültig werden können).

Bei Sperren unterscheidet man verschiedene Arten (*shared*, *exclusive*, *intentional*) und Granularitäten (Feld, Tupel, Seite, Tabelle). Transaktionen, die eine benötigte Sperre nicht erlangen können, warten auf die Freigabe (Gefahr der Verklemmung) oder werden zurückgesetzt (Gefahr des Aushungerns). Sperren werden meist über das Zweiphasen-Sperrprotokoll (2PL) realisiert, das in der Wachstumsphase alle benötigten Sperren erlangt und in der Reduzierungsphase Sperren aufgibt, ohne neue zu erlangen. Diese Technik ist schnell und hat eine einfache Implementierung. Sie ist hinreichend für Serialisierbarkeit, genügt aber nicht für Isolation, wenn andere Transaktionen Resultate vor dem commit lesen können. Dies vermeidet man, wenn alle Sperren bis zum commit gehalten werden (striktes Zweiphasen-Sperrprotokoll).

Alternativ kann mit Zeitmarken (*time stamps*) gearbeitet werden, wozu jede Transaktion T_i eine eindeutige Zeitmarke $TS(T_i)$ z.B. mittels Rechneruhr oder über einen Zähler erhält. Transaktionen werden so ausgeführt, als ob sie in Zeitmarkenordnung serialisiert wären. Alle Operationen führen die Zeitmarke ihrer Transaktion mit. Spe-

ziell kann eine Schreiboperation nie jünger sein als eine vorher erfolgte Leseoperation auf dem selben Datenobjekt. Generell läßt sich zeigen, daß es Ablauffolgen durch den Zeitmarkenalgorithmus gibt, die durch 2PL nicht erzeugbar wären und umgekehrt.

Wie ausgeführt liegt dem Transaktionsbegriff in Datenbanken das Konzept der Konsistenz zugrunde, als eine statische Eigenschaft. Mittels Nebenläufigkeitskontrolle gilt es diese zu erhalten, wobei Serialisierbarkeit (*serializability*) den theoretischen Hintergrund dazu liefert.

Nach *Gray* und *Reuter* [84] läßt sich dies unter dem umfassenden Begriff Isolation behandeln, der ja in dem „I" in den ACID-Eigenschaften steckt. Beim Übergang zu den Anwendungen der synchronen Gruppenarbeit, wird diese Isolation aufgegeben zugunsten der Awareness. Damit müssen wir untersuchen, welche durch die Isolationzusicherung des DBMS verhinderten Konflikte jetzt neu von der Benutzern durch die vom CSCW-System ermöglichte Awareness zu lösen sind.

Ein generelles Abhängigkeitsmodell (vgl. [84] Abschnitt 7.3) betrachtet die Mengen der Objekte, die eine Transaktion T_i liest, genannt die *Eingabemenge* I_i. Die Objekte, die T_i schreibt (modifiziert) ist dann die *Ausgabemenge* O_i.

Eine Menge von Transaktionen $\{T_i\}$ kann parallel arbeiten ohne Nebenläufigkeitsanomalien, wenn ihre Ausgabemengen disjunkt sind von den Eingabe- und Ausgabemengen der anderen Transaktionen: $O_i \cap (I_j \cup O_j) = \emptyset$, für alle $i \neq j$.

In einer CSCW-Anwendung sind in erster Näherung die in den Viewports sichtbaren Daten die Eingabemengen. Modifizierte (editierte) Objekte sind die Ausgabemengen. Es ist geradezu beabsichtigt, daß in einer CSCW-Anwendung, die nebenläufiges Editieren auf einem gemeinsam sichtbaren Datenraum erlaubt, Lesemengen eines Anwenders einen nichtleeren Durchschnitt mit den Schreibmengen eines anderen Teilnehmers haben. Auch wenn nebenläufiges Editieren nicht gestattet ist, will man aber in jedem Fall Schreiboperationen anderer Teilnehmer verfolgen, d. h. auch transiente Objektzustände während einer Modifikation des Objekts durch einen anderen Teilnehmer beobachten.

Beispiel: Zwei Studenten telefonieren, während sie in ihrem Viewport die Vorlesungsankündigungen sehen. Gleichzeitig editiert die Studiendekanin den Plan, etwa indem sie die Uhrzeit einer Veranstaltung ändert. Das kann dazu führen, daß die Studierenden beide noch den alten Wert, beide schon den neuen Wert, oder einer den alten und der andere den neuen Wert sieht.

Nach dem angestrebten Korrektheitskriterium der Gruppenarbeit sollte das System den beiden kooperierenden Studierenden eine konsistente Sicht anbieten, auf der sie ihre Entscheidungen treffen. Ist zum Zeitpunkt der Koordinierung klar, daß eine Schreiboperation ansteht (Finger eines schreibberechtigten Teilnehmers auf dem Objekt oder einem umgebenden Objekt?), dann sind die lesenden Teilnehmer darauf aufmerksam

zu machen („*have to be made aware of pending changes*"). Dazu können auch Sperren gehören (vgl. Abschnitt 4.4.4).

Die statische Vorherbestimmung der Eingabe und Ausgabemengen einer Transaktion (und Zurückweisung einer Transaktion bei vorhersehbarem nichtleeren Durchschnitt mit anderen laufenden oder geplanten Transaktionen) wurde in Datenbanksystemen ab 1972 als Technik aufgegeben zugunsten einer dynamischen Bestimmung, etwa mit der Erlangung von Sperren oder Zeitstempeln. In der Gruppenarbeit, bei der Transaktionen (Kooperationen) spontan und situationsbedingt ablaufen, wäre eine statische Vorausbestimmung aller potentiellen Foci und Nimbi schwer vorstellbar, d. h. auch hier müssen Konfliktsituationen dynamisch bestimmt werden.

Bei ACID-Transaktionen stellt sich nun heraus, daß unter der Menge der möglichen Abhängigkeiten von Schreib- und Leseoperationen nur drei zu vermeidende zyklische Abhängigkeitsgraphen auftreten, die unter den Namen verlorene Modifikation (*lost update*), Lesen transienter Werte (*dirty read*) und nichtwiederholbares Lesen (*unrepeatable read*) bekannt sind.

Betrachten wir jetzt diese drei Anomalien im Lichte einer Gruppenarbeitsanwendung.

Lost Update

Die Anomalie tritt auf, wenn eine Transaktion, sagen wir T_1, einen Wert o liest, danach eine andere Transaktion T_2 diesen Wert ändert, T_1 dann ebenfalls eine Änderung macht, also auch o schreibt, ohne die vorherige Änderung durch T_2 zu kennen („ohne ihr gewahr zu sein!"). Wir betrachten also die Abblauffolge $T_1R(o),\ T_2W(o),\ T_1W(o)$.

In einer CSCW-Anwendung entspräche dies einer Situation, in der T_1 eine Modifikation an o vornimmt, ohne die vorherige Modifikation durch T_2 zu kennen. Dies könnte sein, weil T_1 die Anzeige (Visualisierung) der Modifikation noch nicht erhalten hat oder weil er sie ignoriert (nicht anschaut).

Im Fall, daß die Schreiboperation durch T_1 auf einer noch nicht aktualisierter Sicht beruht, soll in dieser Arbeit ein Protokoll entworfen werden, mit dem dies entdeckt und vermieden wird, wodurch T_1 die Operation $W(o)$ dann abbrechen muß.

Das System wird also ein erneutes $T_1R(o)$ einschieben, wodurch die *Unrepeatable Read*-Situation (siehe unten) eintritt. Dies ist aber in einer CSCW-Anwendung zu erwarten, d. h. innerhalb einer Sitzung ändern sich während der Editierarbeit Werte. Das ist akzeptabel, wenn der Teilnehmer dessen gewahr ist.

Gelingt es einer Transaktion T_1 in einem klassischen DBMS eine Lesesperre auf o zu erlangen, kann T_2 nicht o modifizieren. T_1 kann dann auch eine Schreibsperre auf o erlangen und die Anomalie tritt nicht auf.

Für den Übergang zur CSCW-Anwendung ist interessant zu prüfen, wie ein Zeitmarkenverfahren mit *Lost Update* umgeht.

Der Algorithmus benutzt die Zeitmarke TS einer Transaktion (Zeitstempel des Transaktionsstarts). Jeder Satz hat eine Lese- und eine Schreibzeitmarke (RTM und WTM).

Bei einer Leseoperation durch eine Transaktion mit Zeitmarke TS wird geprüft:

Wenn $TS < WTM(o)$	dann zurückweisen der Leseoperation, Zurückrollen und Neustart mit neuer Zeitmarke
sonst	Lesen und setzen $RTM(o) := max(RTM(o), TS)$.

Bei einer Schreiboperation durch eine Transaktion mit Zeitmarke TS wird geprüft:

Wenn $TS < RTM(o)$	dann Zurückweisen der Schreiboperation, Zurückrollen und Neustart mit neuer Zeitmarke
sonst	wenn $TS < WTM(o)$ dann ignoriere Schreiboperation sonst Schreiben, $WTM(o) := TS$.

Fall 1: Die Zeitmarke TS_1 (von Transaktion T_1) sei $< TS_2$. und $RTM(o)$ sowie $WTM(o) < TS_1$, d. h. die letzten Lese und Schreiboperationen darauf lagen vor dem Start von T_1 und damit auch T_2.

Dann bedeutet T_1 $R(o)$, daß $TS_1 > WTM(o)$ und damit ein Lesen und setzen von $RTM(o)$ auf TS_1. Bei der folgenden Schreiboperation durch T_2 gilt $TS_2 \geq TS_1 = RTM(o)$. Damit wird geprüft, ob $TS_2 < WTM(o)$, was nicht der Fall ist, und somit wird geschrieben und anschließend gilt $WTM(o) = TS_2$.

Nun will T_1 schreiben. Weil $TS_1 = RTM(o)$, also TS_1 **nicht** echt kleiner $RTM(o)$, wird der sonst-Zweig betreten. Es wird geprüft, ob $TS_1 < WTM(o) = TS_2$, was der Fall ist. Dann wird die Schreiboperation ignoriert (weggeworfen), es gilt weiterhin der von T_2 gesetzte Wert.

Lost Update wird damit zu *Ignored Update*. Die Begründung ist, daß niemand den Wert der geplanten Schreiboperation von T_1 gelesen hat, sonst wäre die Lesemarke kleiner gewesen.

Fall 2: Die Zeitmarke TS_1 sei $> TS_2$, also T_1 startet nach T_2. Die Leseoperation von T_1 wird durchgeführt und setzt $RTM(o)$ auf TS_1.

Die Schreiboperation von T_2 prüft, ob $TS_2 < RTM(o) = TS_1$. Da laut Annahme T_1 nach T_2 gestartet ist, ergibt der Vergleich wahr ($TS_1 > TS_2$) und die Schreiboperation von T_2 wird zurückgewiesen, T_2 wird abgebrochen und mit einer neuen, späteren Zeitmarke gestartet.

Somit haben wir einmal eine ignorierte Schreiboperation und einmal einen Abbruch, je nachdem wie der Start der Transaktionen erfolgte.

Bezogen auf CSCW-Anwendungen bedeutet die *Ignored Update*-Situation beim Zeitmarkenverfahren im Fall 1 (T_1 startet vor T_2), daß der Server entscheidet, daß ein Teilnehmer T_1 seine eigene Modifikation nicht sieht und sie auch allen anderen nicht mitgeteilt wird mit dem Argument, die Modifikation sei bereits überholt durch eine nachfolgende Modifikation durch Teilnehmer T_2.

Dies ist in unserem System nicht vorgesehen, in jedem Fall müßte man T_1 eine Meldung machen, warum seine Modifikation ihm nicht angezeigt wird. Tatsächlich wird unser Verfahren eine Meldung an T_1 schicken, wonach der Server bereits eine konfliktäre Operation von T_2 ausgeführt und zur Visualisierung verschickt hat. Wichtig wäre bei T_1, daß die Visualisierung der Operation von T_2 ihm nicht angezeigt wird, bevor eine Meldung kommt.

In der Praxis kann dies tatsächlich auch dazu führen, daß der Teilnehmer T_1 dann auf seine Modifikation verzichtet, besonders wenn die zu schreibenden Werte gleich sind. Wichtig erscheint, daß der Server auf der Basis der eintreffenden Operationsauslösungen arbeitet und sofort mit der Ausführung beginnt, wenn er erkennen kann, daß die Ausführung auf aktuellen Visualisierungen beim Teilnehmer beruhen.

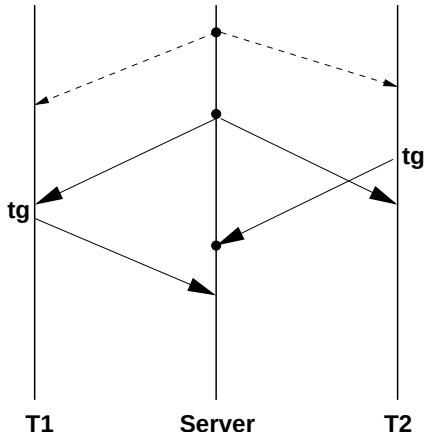


Abbildung 3.7: $TS_1 > TS_2$ (T_1 startet nach T_2)

Dies ist schwieriger bei Fall 2. Hier wird beim Zeitmarkenverfahren T_2 abgebrochen, weil T_2 vor T_1 gestartet ist, T_1 aber noch den alten Wert gesehen hat. Das Problem ist, daß der Abbruch der Schreiboperation des Teilnehmers T_2 vor dem Eintreffen der Schreiboperation des Teilnehmers T_1 erfolgt. Der Server muß also einen Lese/Schreibkonflikt erkennen. Gerechtfertigt wäre der Abbruch, wenn tatsächlich die Auslösung der Schreiboperation auf nicht aktuellen Visualisierungen bei T_2 beruht. Dazu muß der Server nach einer Visualisierungsbotschaft suchen, die an T_2 verschickt wurde und einen Zeitstempel hat, der größer ist als der der Auslösebotschaft (vgl. Abb. 3.7). Dies wäre eine zeit-/visualisierungsbezogene Prüfung. Ist dies möglich, wäre der Abbruch konsistent mit dem angestrebten Korrektheitskriterium.

Für Teilnehmer T_2 wäre es unter Umständen hart, zu sehen, daß seine Operation abgewiesen wird, die von T_1 danach aber durchgeht, obwohl T_2 vor T_1 eine Schreiboperation abgesandt hat. Im übrigen können auch T_1 und T_2 beide scheitern, wenn in beiden Fällen sich die Operationsauslösung mit einer noch nicht angezeigten Visualisierung kreuzt.

Dirty Read

Die Anomalie entsteht bei $T_1W(o)$, $T_2R(o)$, $T_1W(o)$ und bezeichnet das lesen eines Wertes einer noch nicht mit Commit oder später mit Abort abgeschlossenen Transaktion. Prinzipiell würde man gerne bei CSCW-Anwendungen Dirty Read zulassen, wenn es den Benutzern bekannt gemacht wird.

Beim DBMS Zeitmarkenverfahren führt die Verletzung der Isolation zu folgenden Abbrüchen.

Im ***Fall 1*** startet T_1 vor T_2 ($TS_1 < TS_2$). Die Schreiboperation durch T_1 stellt fest, daß $TS_1 > RTM(o)$ und weil $TS_1 > WTM(o)$ wird geschrieben und $WTM(o) := TS_1$.

Jetzt folgt die Leseoperation durch T_2. Weil $TS_2 > WTM(o) = TS_1$ wird gelesen und $RTM(o) := TS_2$.

Als drittes will T_1 nochmal Schreiben. Weil $TS_1 < RTM(o) = TS_2$ wird T_1 abgebrochen.

Die Interpretation im DBMS-Sinn ist, daß der Lese-/Schreibkonflikt zwischen T_1 und T_2 aufgedeckt wurde und weil T_2 gelesen hat, bevor T_1 (nochmal) schreibt, aber nach Annahme T_1 vor T_2 läuft, muß abgebrochen werden.

Im ***Fall 2*** startet T_2 vor T_1 ($TS_1 > TS_2$).

Zunächst schreibt T_1. Das funktioniert, da $TS_1 > RTM(o)$ und $TS_1 > WTM(o)$. Anschließend wird $WTM(o)$ auf TS_1 gesetzt.

Jetzt will T_2 Lesen. Weil $TS_2 < WTM(o) = TS_1$ wird diese leseoperation sofort zurückgewiesen und T_2 muß neu starten.

Die Interpretation für ACID-Transaktionen ist, daß T_2 einen Wert liest, den T_1 bereits geschrieben hat, obwohl nach Annahme T_2 vor T_1 läuft.

Man sieht, daß Dirty Read als Verletzung der Isolation stark auf die Klammerung der Operationen durch eine Transaktion abhebt und nur Werte nach draußen gegeben werden sollen, die mit einem Commit „abgesegnet" wurden. In vielen DBMS-Anwendungen könnte ein Dirty Read aufgrund semantischer Zusatzinformation zugelassen werden.

Bei CSCW-Anwendungen wird man in der Regel keinen solchen starken Transaktionsbegriff wollen. Wie ausgeführt ist das Korrektheitskriterium der Start von Modifikationen immer auf der Basis aktueller Information.

Vorstellen könnte man sich die Zurückweisung der zweiten Schreiboperation bei T_1, wenn diese ausgelöst wurde, bevor T_1 die Visualisierung der eignen ersten Schreiboperation angezeigt bekommt. Es steht dann nämlich zu vermuten, daß auch andere die Visualisierung der ersten Schreiboperation sehen wenn schon eine weitere Modifikation in Bearbeitung ist. Diese Bedingung zu entdecken wäre bei dem in dieser Arbeit vorgestellten System möglich.

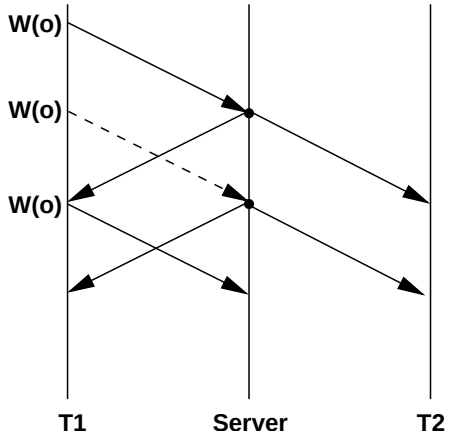


Abbildung 3.8: $TS_1 < TS_2$ (T_1 startet vor T_2)

Im Fall 2 hat man die Situation, daß T_2 bei Dirty Read nur liest. Man kann sich eine Fehlermeldung oder einen Abbruch irgendwelcher Art bei T_2 schwer vorstellen. Der Vermeidung von Dirty Read bei Datenbanksystemen entspricht in CSCW-Systemen die Vermeidung der Verbreitung von Sichten, die das System widerruft. Das kann man aber vermeiden, indem nur „gültige" Visualisierungen verschickt werden, also solche, bei denen die Operationsauslösung auf aktuellen Sichten beruhte. Teilnehmer, die solche Aktionen ausgelöst haben, können diese auch nicht mehr zurücknehmen, es sei den, sie klammern mehrere Operationen und machen sie erst nach einem Commit sichtbar (vgl. Abschnitt 4.4.4).

Unrepeatable Read

Diese dritte Form der Anomalie bei fehlender Isolation beruht auf der Folge $T_1R(o)$, $T_2W(o)$, $T_1R(o)$

Innerhalb der noch nicht abgeschlossenen Transaktion T_1 schreibt eine zweite Transaktion T_2, wodurch T_1 bei zweiten Lesen nicht mehr den alten Wert sieht.

Für die Anwendung des Zeitmarkenverfahrens unterscheiden wir wieder Fall 1 ($TS_1 < TS_2$) und Fall 2 ($TS_1 > TS_2$).

Fall 1: Zunächst ist $TS_1 > WTM(o)$, deshalb wird gelesen und danach die Lesemarke $RTM(o)$ auf TS_1 gesetzt. Jetzt will T_2 schreiben, es gilt $TS_2 > RTM(o) = TS_1$. Der Test im Sonst-Fall ergibt $TS_2 > WTM(0)$, deshalb wird geschrieben und $WTM(o) := TS_2$. Nun erfolgt das zweite Lesen durch T_1, aber $TS_1 < WTM(o) = TS_2$, T_1 wird abgewiesen und muß neu starten. Die Anomalie wird beim zweiten Leseversuch entdeckt, wenn die früher gestartete Transaktion T_1 nochmals lesen will, der Wert aber schon von der nachfolgenden Transaktion T_2 modifiziert wurde.

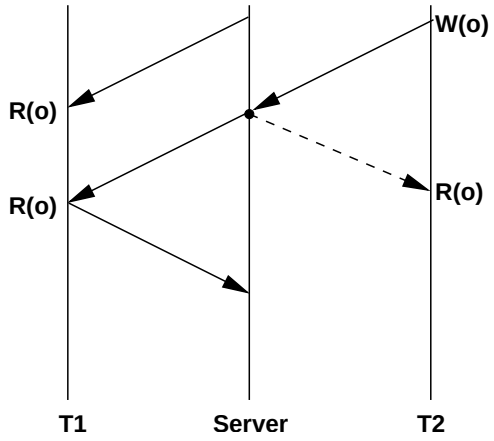


Abbildung 3.9: $TS_1 < TS_2$ (T_1 startet vor T_2)

Fall 2: (T_1 startet nach T_2, also $TS_1 > TS_2$): Auch hier ist zunächst $TS_1 > WTM(o)$, es wird gelesen und $RTM(o)$ auf TS_1 gesetzt. Beim Schreiben von T_2 ist $TS_2 < RTM(o) = TS_1$, der Schreibversuch wird abgebrochen und T_2 startet neu mit einem späteren Zeitstempel. Die Anomalie wird schon beim Schreibversuch entdeckt.

Generell versucht im Fall 2 die früher gestartete Transaktion T_2 einen Wert zu überschreiben, den eine später gestartete Transaktion schon gelesen hat, d. h. T_2 ist zu langsam. Hätte den alten Wert „niemand" gelesen und hätte den Wert eine später gestartete Transaktion bereits überschrieben, dann könnte man die Schreiboperation von T_2 Fallenlassen und weitermachen (vgl. 3.6.4), so aber ist der alte Wert nach draußen gedrungen und T_2 muß neu starten.

Bezogen auf CSCW-Anwendungen entspricht Fall 1 einer Situation, bei der ein Teilnehmer T_1 einen speziellen Wert sieht. Kaum hat er begonnen, darauf zu reagieren, sieht er, wie sich der Wert ändert. Man denke etwa an ein Sonderangebot. Kaum hat der Teilnehmer mit der Ehefrau telefoniert und sich verständigt, das Angebot zu nutzen, ist es weg, er kann den Wert nicht mehr finden. Ist dem Teilnehmer der Wechsel des Werts bewußt („*user is made aware of the change*"), kann er darauf reagieren und entweder seine Aktivitäten abbrechen, im Beispiel vom Kauf absehen, oder den neuen Wert akzeptieren und weitermachen (vgl. Abb. 3.9).

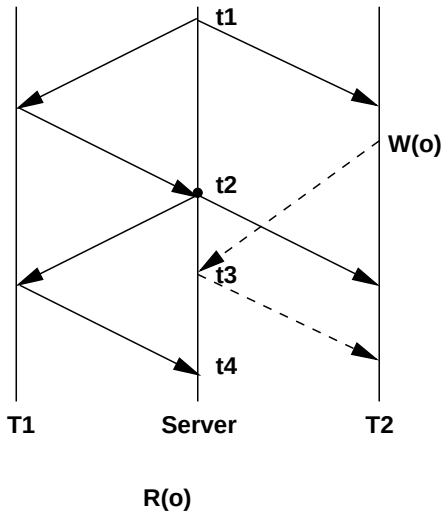


Abbildung 3.10: $TS_1 > TS_2$ (T_2 startet vor T_1)

Im zweiten Fall will ein spät reagierender Teilnehmer Objekte modifizieren. obwohl andere ihre Entscheidung schon auf der Basis dieser alten Werte getroffen haben. Dies könnte der Server nach den in dieser Arbeit vorgestellten Korrektheitskriterien entdecken (vgl. Abb. 3.10).

Kapitel 4

Systemarchitektur

Während im vorigen Kapitel die Grundideen und das theoretische Modell vorgestellt wurden, geht es in diesem Kapitel um die konkrete Realisierung. Aufbauend auf einer Schichtenarchitektur, werden das Daten- und Interaktionsmodell sowie die Modelle für Awareness und die Kontrolle der Nebenläufigkeit besprochen.

Dabei wird versucht, die von verschiedenen Autoren für spezielle Aufgaben der synchronen Gruppenarbeit entwickelten Verfahren in ein kohärentes Gesamtmodell zu integrieren und eine prototypische Implementierung dafür zu entwickeln. Besonderen Wert wird auf ein angepaßtes Korrektheitskriterium für *visuelle Transaktionen* gelegt.

4.1 Schichtenarchitektur

Im vorhergehenden Kapitel wurden vier Teilmodelle identifiziert:

- Interaktionsmodell
- Datenmodell
- Visualisierungsmodell
- Awarenessmodell mit Kontrolle der Nebenläufigkeit

Es gilt nun, die Teilmodelle in kohärenter Weise in eine Systemarchitektur abzubilden. Dabei ist offensichtlich, daß die Anwender „Clients" darstellen, man also eine Art von Client-Server-Architektur verwenden wird. Genauso klar ist, daß man Teile der Visualisierung sowie das Aufsammeln der Interaktionsereignisse - das Frontend also - dem Client überlassen kann. Die genaue Form wird aber zu diskutieren sein, etwa ob ein Versand der Objektdaten oder ein Versand der Visualisierungsdaten des Objekts erfolgen soll. Hinweise zu Alternativen enthält unter anderem die Arbeit von *Prakash* et al. mit einem Abschnitt über *Architectural Considerations for Groupware Systems* [152, S. 216].

Auf der Serverseite wird man vernünftigerweise die Aufgaben der Datenhaltung, der Interaktion mittels Finger, wie in 3.2 angedeutet, sowie den schwierigen Teil der Awareness und Nebenläufigkeitskontrolle in eine Art Schichtenarchitektur aufteilen. Hierzu kann man sich an dem Modell von *Ellis* [64] sowie dem Modell von *Teege* [187] orientieren. Letzteres ist eigentlich eine Erweiterung der Arbeiten von *Ellis.*

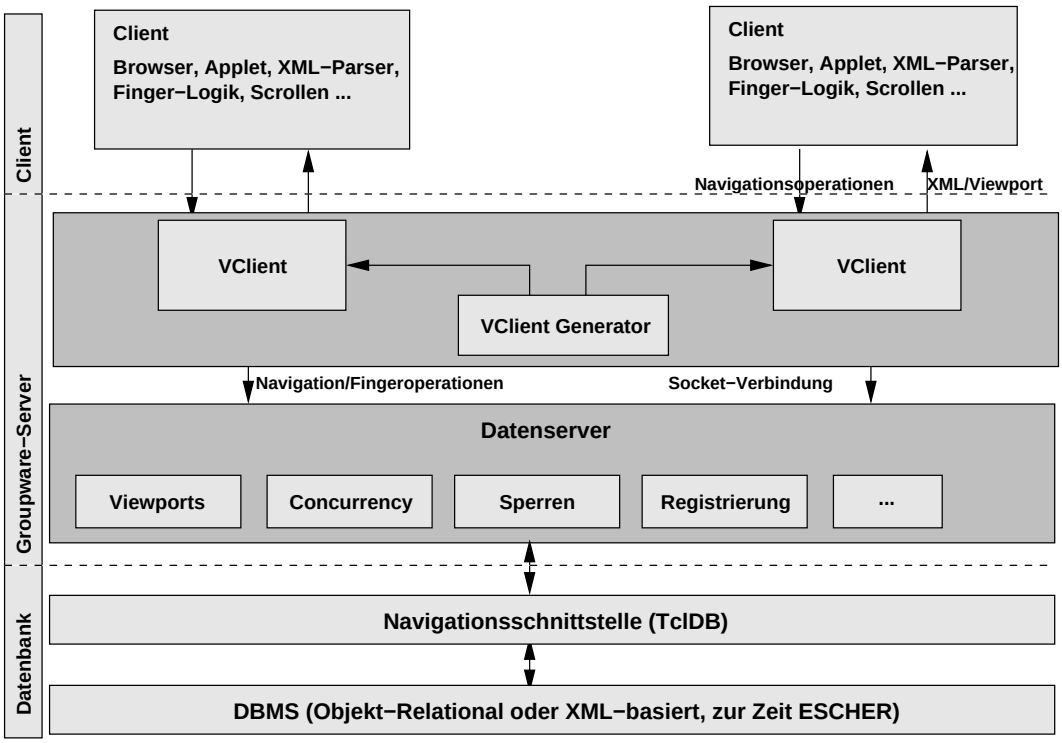


Abbildung 4.1: Die Schichtenarchitektur

Das Modell kann in drei Hauptschichten gemäß Abbildung 4.1 aufgeteilt werden:

- User front end/Client
- Groupware-Server
- Daten- and Interaktion Server

4.1.1 Daten- and Interaktion Server

Für den gemeinsamen Informationsraum wird das eNF^2-Datenmodell verwendet. Wie schon erwähnt, ist dieses Modell nichts anders als eine isomorphe Darstellung von XML-Dokumenten oder einer Baumdarstellung [222]. Die schon beschriebene Operations- und Navigationssemantik für dieses Datenmodell wurde bereits früher für einen Datenbankeditor ESCHER implementiert und kann daher am einfachsten für die Implementierung des Interaktionsmodells verwendet werden. Für die Realisierung des

Interaktionsmodells dient das in ESCHER verwendete *Fingerparadigma*. Dabei wird der Zugriff auf eine eNF^2-Tabelle durch eine Tcl-Schnittstelle, genannt TclDB [18], realisiert. Vorstellbar ist ebenfalls eine Realisierung basierend auf XML-Datenbanken, etwa auf Basis von XQuery/XPath [216, 206] oder XDNL [107] (siehe Kapitel 6).

ESCHER

Die Ursprünge von ESCHER[1] gehen auf das Jahr 1986 zurück. Am wissenschaftlichen Zentrum der IBM in Heidelberg wurde in den achtziger Jahren von *Dadam, K¨uspert* und anderen ein Prototyp zum fortschrittlichen Informations-Management (*Advanced Information Management Prototype*, AIM-P) entwickelt [51]. Dieser sollte im wesentlichen das eNF^2-Datenmodell realisieren.

Die Konzepte der Interaktion mit geschachtelten Datenobjekten durch Cursor, in ESCHER „Finger“ genannt, und eines Meta-Schemas als Schema aller Schemata inklusive seiner selbst , wurden von *Wegner* erstmals 1989 in Tokyo publiziert [221]. Die dortige Konferenz wurde zur ersten aus der Reihe *„Visual Database Systems“*. Sie war insofern wegweisend, als sie erstmals die visuellen Aspekte von Datenbanken, sowohl was die Formen des Umgangs mit den Daten als auch die Art der Daten selbst anbetrifft, in den Vordergrund stellte. Viele Autoren des Konferenzberichtes [123] haben die Entwicklung im Bereich Multimedia und komplexe Objekte massiv beeinflußt.

Die Systemarchitektur von ESCHER, ist in [219] beschrieben. Sie umfaßt für die unteren drei Schichten eine Aufteilung in

- Record-Manager (RM),
- Data-Manager (DM) und
- Object-Manager (OM).

Der Record-Manager [218] ist für die persistente Speicherung von Byte-Folgen zuständig und vergibt für diese persistenten „Objekte“ Identifikatoren. Dabei wurde das aus System R [25] bekannte Konzept der invarianten Tupel-Identifikatoren (*tuple identifier*, TID) übernommen, für das im Record-Manager der Name *Record Identifier* (RID) verwendet wird. Record Identifier können als Objektidentifikatoren betrachtet werden, sie sind aber implementationsnäher ausgelegt. Aus heutiger Sicht würde man diese Schicht als Objektspeicher (*object repository*) bezeichnen, wobei ein Transaktionskonzept, das Mehrbenutzerfähigkeit durch Kontrolle der Nebenläufigkeit (*concurrency*), Wiederherstellbarkeit bei Absturz (*recovery*) usw. ermöglichen würde, fehlt. Insofern kann es nicht mit anderen experimentellen Objekt-Speichern wie etwa Shore [43] oder EOS [35, 36] auf eine Stufe gestellt werden.

[1] Der Namenspatron von ESCHER ist der holl¨andischen K¨unstler M. C. Escher (1898–1972), der f¨ur seine selbstreferenzierenden Graphiken bekannt ist.

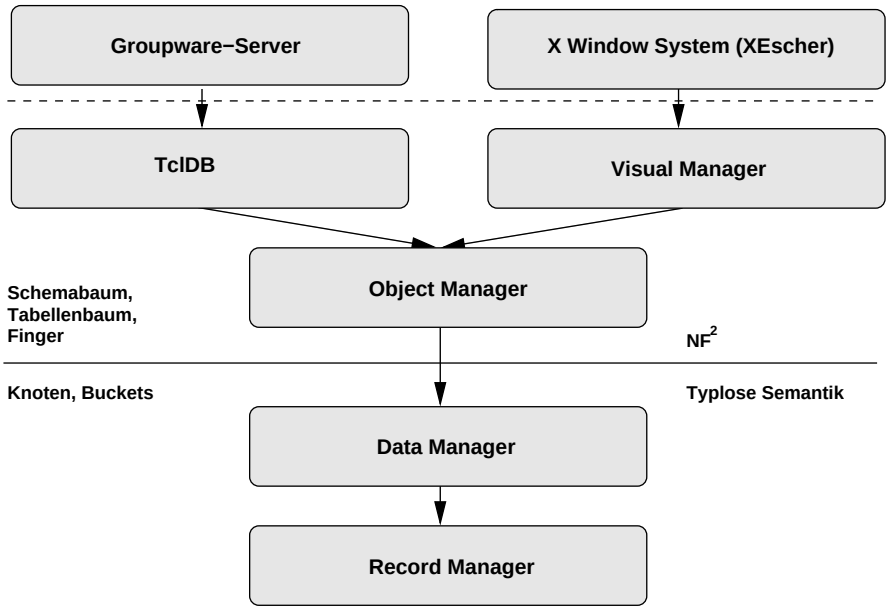


Abbildung 4.2: Die Schichtenarchitektur aufbauend auf ESCHER

Auf der nächsthöheren Ebene ist der Data-Manager [220] für die Strukturierung der Byte-Folgen verantwortlich. Er greift anhand von RIDs auf die Byte-Folgen zu und interpretiert sie. Dazu hat er ein Repertoire an elementaren Speichertypen (z. B. Integer, String) und verwaltet diese in baumartigen Strukturen.

Über dem Data-Manager liegt schließlich der Object-Manager. Dieser versteht die komplexen Datentypen des eNF^2-Datenmodells, die dem Anwender an der DBMS-Schnittstelle angeboten werden. Er kann z. B. zwischen einer (geordneten) Liste und einer Menge unterscheiden, kann einer Komponente eines Tupels seine physische Position im Vektor, der das Tupel realisiert, zuweisen usw. Er kennt Finger und bietet eine Fülle von Funktionen an, die der Navigation und Datenmanipulation mit diesen „Objektzeigern“ dienen.

Die unteren drei Schichten von ESCHER implementieren insgesamt das eNF^2-Datenmodell, wobei in der mittleren Schicht, dem Data-Manager, der Übergang zwischen typlosen hierarchischen Strukturen und getypten komplexen Objekten liegt.

Interessanterweise haben sich die in [219, S. 346] aufgelisteten Funktionen und Sprechweisen bis heute als ausreichend und im Wesentlichen nicht veränderungsbedürftig erwiesen.

Sprachlösungen, die sowohl auf Datendefinitions- als auch Datenmanipulationsebene mit komplexen Objekten umgehen können und auf modernen Klassenkonzepten basieren, wurden in den Dissertationen von Paul [148] und Thelemann [194] vorgeschlagen, mangels Implementierung aber nicht eingesetzt — anders als z. B. der für AIM-P implementierte Vorschlag HDBL [22], der stark in SQL3 einging. Einerseits

wurde dadurch für ESCHER nie eine objekt-orientierte Sprachebene erreicht. Andererseits wäre eine von den Sprachstandards der ODMG (ODL und OQL) losgelöste Entwicklung wenig sinnvoll gewesen, auch wenn die Erwartungen, die an die OODBS gestellt wurden, enttäuscht werden.

Alternativ wurde für die Programmierung der für Objekte benötigten Methoden eine kleine, einfache typarme Skriptsprache aufbauend auf Tcl (*Ousterhout* [145]) vorgeschlagen.

Die Implementierung einer solchen aktiven Komponente auf der Basis kleiner, interpretierter Programmstücke existiert für ESCHER seit 1997. Unter dem Namen TclDB wurde *Ousterhouts* Skriptsprache Tcl um Kommandos für die Manipulation von und die Navigation in komplexen Objekten durch die Kopplung an die Programmierschnittstelle von ESCHERs Object-Manager erweitert [18, 227, 189, 190].

Ob die Entscheidung der Wirtssprache heute zugunsten Java ausfallen würde, das immer mehr an Bedeutung zu gewinnen scheint, ist unklar. Vielleicht erweist sich aber die eher konservative Wahl von Tcl wie im Fall der OODBS als langfristig richtig. Eine gewisse Ermüdung über die ständig neuen Merkmale und die Kämpfe um die Normung bei Java läßt das vermuten[2]. Auch Äußerungen wie die von Kernighan, einem der Väter des Betriebssystems UNIX und der Programmiersprache C, daß er momentan viel mit Tcl programmiert (*„Much of my programming in the past few years has been in Tcl/Tk* [...]“ [115, S. 67]), lassen sich als Bestärkung der Skriptidee lesen.

Eine Kooperation mit der SAP AG, Walldorf, hat dazu geführt, daß das Interaktionskonzept von ESCHER für die Objektbehandlung in ABAP übernommen wurde[3]. Das ist ein Indiz dafür, daß Benutzerschnittstellen für komplexe Objekte, die im Hintergrund auf post-relationalen DBMS aufbauen, auch kommerziell an Bedeutung gewinnen.

Zusammengefaßt kann also festgestellt werden, daß sich die eher evolutionäre als revolutionäre Entstehungsgeschichte von ESCHER im nachhinein als recht stabil herausgestellt hat. Sie steht damit im Einklang mit den objekt-relationalen Entwicklungen, die strukturelle Objekt-Orientierung in den Vordergrund stellen und auf eine allzu enge Kopplung an objekt-orientierte Programmiersprachen verzichten. Die derzeitigen Erweiterungen von ESCHER, z. B. die Interaktion über eine WWW-Schnittstelle [193, 19, 79], zeigen, daß die grundlegenden Konzepte erweiterungsfähig sind.

Abbildung 4.3 zeigt eine Beispieltabelle in ESCHER. Die äußere Menge `Abteilungen` enthält zwei atomare Attribute `Abteilungsname` und `Ort` sowie eine Menge `Angestellte`. Der Finger steht auf dem ersten Tupel der Menge `Angestellte`. Ein korrespondierender Schema-Finger steht auf der entsprechenden Stelle in der

[2] In [126], [75] wird z. B. auf dem Titelblatt mit „Java: Half Empty?“ angek¨undigt

[3] ABAP (*Advanced Business Application Programming*) ist eine Programmiersprache; das Interaktionskonzept wurde also genau genommen in den Editor der integrierten Entwicklungsumgebung ¨ubernommen.

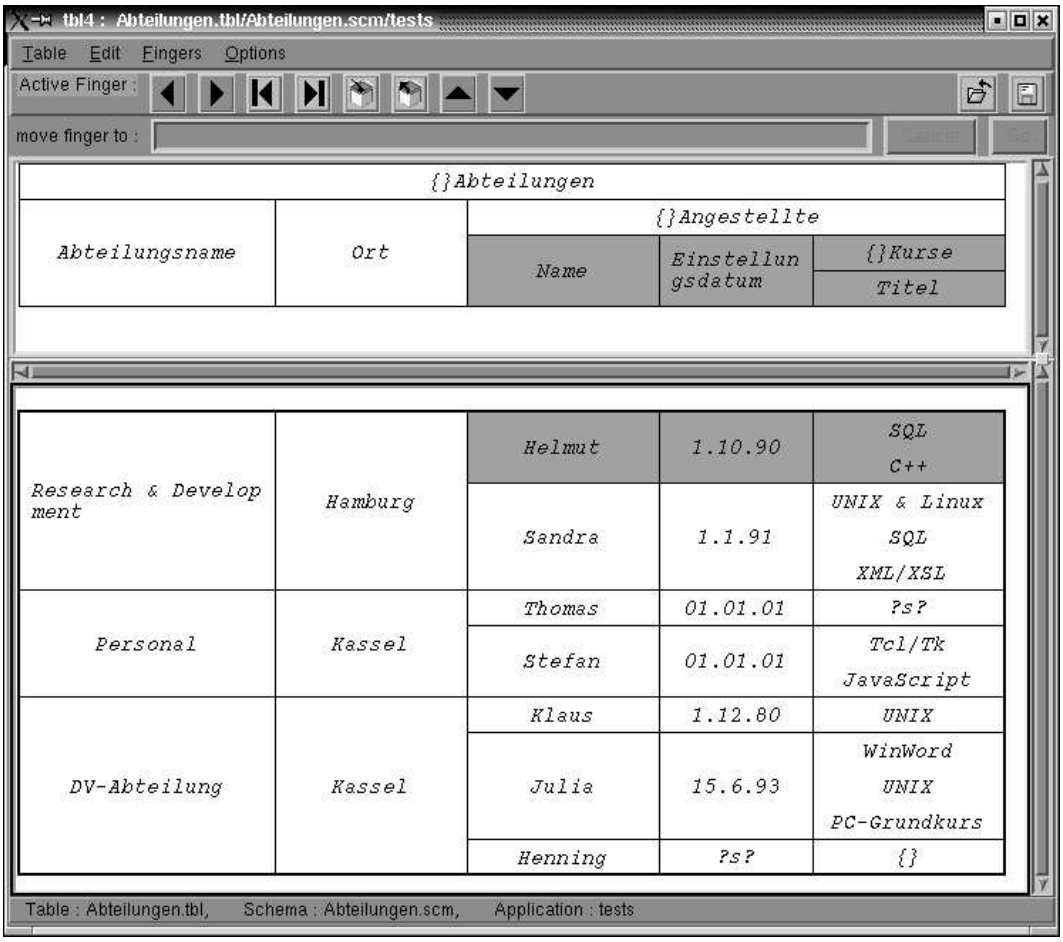


Abbildung 4.3: Beispieltabelle in ESCHER

Schemaanzeige. Null-, und leere Mengen haben eine eindeutige semantische Bedeutung [219]. Nullmengen werden durch `{???}`, leere Mengen durch `{}` und nullwertige atomare Attribute durch `?s?` dargestellt. In der Tabelle hat z. B. *Thomas* sich für einen (unbestimmten) Kurs eingetragen, während *Henning* eine unbestimmte Anzahl von Kursen belegt. Das Einstellungsdatum von *Henning* ist unbekannt.

Als Schnittstelle des Groupware-Servers zu der Datenbank dient die Tcl-Schnittstelle TclDB [144, 18]. Diese greift auf die OM-Funktionen von ESCHER zu und stellt die Funktionalität von ESCHER als eine Sammlung von Tcl-Befehlen zur Verfügung. Anhang A gibt eine Kurzbeschreibung aller Befehle von TclDB.

Als Beispiel eines TclDB-Skripts zeigt Programm 4.1 die Prozedur `iterate`. Diese ist eine generische Routine, die als Argument ein beliebiges Tcl-Skript, `body`, akzeptiert, das für alle Sohnattribute eines komplexen Attributes, auf dem der Finger `finger` steht, ausgeführt wird. Die Prozedur `iterate` wird oft dazu benutzt, bestimmte Operationen auf einem komplexen Objekt und allen seinen Söhnen rekursiv auszuführen. Siehe Program 5.10.

Option	Aufgabenbereich
Parameter	Bemerkung
`push`	*`in`*-Operation
[`-first`]	erstes Sub-Objekt (Default)
`-last`	letztes Sub-Objekt
`-name` *`attribute`*	Komponente namens *attribute*
`-index` *`i`*	i-tes Sub-Objekt
[`-path`] *`path`*	gemäß *path*
`-rid` *`rid`*	Sub-Objekt mit Identifikation *rid*
`pop`	*`out`*-Operation (keine Parameter)
`go`	*`move`*-Operationen
`-first` — `-last`	erstes/letztes Objekt
`-back` — `-next`	vorheriges/nachfolgendes Objekt
`-name` *`attribute`*	Tupelkomponente namens *attribute*
`-index` *`i`*	i-tes Objekt
`get`	Retrieve (atomar)
[*`path`*]	
`set`	Update
[`-value`] *`value`* [*`path`*]	Wert (atomar)
`-tonull` [*`path`*]	zu Null machen (atomar/Kollektionen)
`-toempty` [*`path`*]	nullwertig $\rightarrow$ leer (Kollektionen)
`-tosingleton` [*`path`*]	leer $\rightarrow$ einelementig (Kollektionen)
`delete`	Löschen (Element)
[*`path`*]	
`insert`	Einfügen (Element)
[`-after`] [*`path`*]	nach *fid* (Default)
`-before` [*`path`*]	vor *fid*
`info`	Information über *fid*
[`-all`] [*`path`*]	alle verfügbaren Informationen (Default)
`-card` [*`path`*]	Kardinalität (komplex)
`-first` — `-last` [*`path`*]	auf erstem/letztem Sub-Objekt?
`-index` [*`path`*]	Index als Sub-Objekt
`-type` [*`path`*]	Typbezeichner
`-name` [*`path`*]	Attributname
`is`	Prädikate
`-null` [*`path`*]	nullwertig?
`-empty` [*`path`*]	leer? (Kollektionen)
`-atomic` [*`path`*]	atomarer Typ?
`-complex` [*`path`*]	komplexer Typ?
`-collection` [*`path`*]	Kollektions-Typ?
`-tuple` [*`path`*]	Tupel-Typ?

Fortsetzung auf der nächsten Seite

Option Parameter	Aufgabenbereich Bemerkung
`link`	Operationen mit Links
[`-link`] [*`path`*]	Fingerposition als Link-Ziel (Default)
`-atomic` [*`path`*]	Wert (atomares Link-Ziel)
`-newfinger` [`-fid` *`new_fid`*] [*`path`*]	neuen Finger auf Link-Ziel
`-followfinger` *`other_fid`* [*`path`*]	Finger *other_fid* auf Link-Ziel positionieren

Tabelle 4.1: Fingeroperationen in TclDB

Programm 4.1 *`tcldb::iterate`*

```
proc iterate {finger body} {
    global errorInfo errorCode
    if {![$finger iscomplex]} {
        error "finger \"$finger\" not complex"
    }
    set cplx [$finger rid]
    if {![$finger push -first]} {return 0}
    while {1} {
        set elem [$finger rid]
        set code [catch {uplevel $body} string]
        if {$code == 0} {
            if {![$finger go -next]} break
            continue
        }
        if {$code == 1} {
            $finger push -rid $cplx
            return -code error -errorinfo $errorInfo \
                    -errorcode $errorCode $string
        } elseif {$code == 2} {
            return $string
        } elseif {$code == 3} {
            $finger push -rid $cplx
            return 1
        } elseif {$code == 4} {
            $finger push -rid $elem
            if {![$finger go -next]} break
            continue
        } else {
            $finger push -rid $cplx
            return -code $code $string
        }
    }
    $finger pop
    return 1
}
```

Pfadausdr¨ucke

Pfadausdrücke (engl. *path expression*), also Ausdrücke zur expliziten oder deklarativen Beschreibung der Auswahl eines Pfades in einer baumartigen Datenstruktur, mit der heute üblichen Punktnotation wurden von Carlo Zaniolo für das Datenbankprojekt GEM eingeführt [229] (siehe Abschnitt 2.1.3).

PATH	::=	DEPTH { «. » DEPTH } | «. » DEPTH { «. » DEPTH }
DEPTH	::=	ATTRIBUTE | INDEX | IMP_POP
		| IMP_PUSH | WHERE
WHERE	::=	«?» AUSDRUCK «?»
AUSDRUCK	::=	AUSDRUCK «||» TERM | TERM
TERM	::=	TERM «&&» FACTOR | FACTOR
FACTOR	::=	VALUE OP VALUE | *true* | *false* | BOOL_FUNC
BOOL_FUNC	::=	(*isnull* | *isempty*) ATTRIBUTE
VALUE	::=	ATTRIBUTE | TEXT | NUMBER | CHAR | CARD_FUNC
CARD_FUNC	::=	*card* ATTRIBUTE
INDEX	::=	DIGIT { DIGIT }
ATTRIBUTE	::=	ALPHA { ALPHANUM }
TEXT	::=	«"»ALPHANUM { ALPHANUM }«"»
CHAR	::=	«' »ALPAHNUM«' »
ALPHANUM	::=	ALPHA | NUMBER
NUMBER	::=	DIGIT { DIGIT } | DIGIT { «. » DIGIT }
DIGIT	::=	0 . . . 9
ALPHA	::=	a . . . z | A . . . Z | «_ » | . . .
OP	::=	«==» | «>» | «<» | «>=» | «<=» | «! =»
IMP_POP	::=	«:»
IMP_PUSH	::=	«/»

Tabelle 4.2: Die Grammatik von Pfadangaben in EBNF-Format

Mit Pfadangaben in TclDB kann man sowohl mehrere Positionierungsoperationen zusammenfassen als auch einfache Abfragen formulieren. Auf der Suche nach Elementen (Tupeln), die die Bedingung in der Pfadangabe (Abfrage) erfüllen, wird das erste Element genommen, das diese Bedingung erfüllt (*First-Match-Semantik*). Die Syntax der Pfadangaben wurde mit Tausch von Backslash und Punkt an das UNIX-Dateisystem angelehnt. Pfadangaben bestehen aus Attributnamen, Indizes und Abfragen. Die einzelnen Teile einer Pfadangabe werden durch Punkte getrennt. Abfragen müssen zwischen zwei Fragezeichen «?» eingeklammert werden[4]. Die Angabe eines Attributna-

[4] Trotz unterschiedlicher Syntax ist die große Ähnlichkeit zwischen diesen Pfadausd¨ucken und der neuen Sprache XPath [206] nicht zu ¨übersehen.

mens in einer Abfrage bezeichnet das Element eines komplexen Objekts, auf dem der Finger steht. Die Angabe von Zeichenketten (`string`) oder Zeichen (`char`) muß wie in Tcl durch zwei Hochkommata bei Strings bzw. ein Hochkomma bei einzelnen Zeichen eingeklammert werden. Tabelle 4.2 enthält die Beschreibung der Grammatik von Pfadangaben in der erweiterten Backus-Naur-Form (EBNF).

In der Tabelle aus Abbildung 4.3 liefert z. B. der Ausdruck `f get ''2.Ort''` den Wert von `Ort` im zweiten Tupel der Menge `Abteilungen` (`Kassel`). Der Ausdruck `f get {?Ort =="Kassel"?.Angestellte.?Name=="Julia"?.\ Einstellungsdatum"}` liefert den `Einstellungsdatum` der ersten Mitarbeiterin mit dem Namen *Julia*, in einer `Abteilung` in Kassel (15.6.93).

4.1.2 Der Groupware-Server

Unter dem Begriff Groupware-Server verstehen wir eine Mittelschicht, die Aufgaben der Mehrbenutzerinteraktion und der Awareness-Signalisierung wahrnimmt. Wie in Abbildung 4.1 gezeigt, gibt es innerhalb des Groupware-Servers eine weitere Trennung in den

- Virtuellen Client (VClient)
- Datenserver

Daraus ist ersichtlich, daß die Schichtenarchitektur eine zentrale Datenspeicherung vorsieht, im Gegensatz etwa zu einer verteilten Datenhaltung, wie in [152] als Alternative angedeutet.

Die weiteren Aufgaben, wie in Abbildung 4.1 angedeutet, können (von oben nach unten) wie folgt beschrieben werden. Die Benutzeroberfläche meldet sich bei dem zentralen Server (VClient-Generator) an. Dieser erzeugt eine Instanz des VClients, die einerseits eine (lokale) Verbindung zum Datenserver, andererseits einen Datenkanal zum in der Regel entfernten Client aufbaut. Diese Schicht bekommt von der Anwendung generische Navigations-, und Schreiboperationen, generiert daraus TclDB-Skripte (siehe Abschnitt 4.1.1) und schickt sie an den Datenserver. Aus den Ergebnissen dieser Operationen generiert sie Visualisierungen, die als XML-Dokumente an den Client weitergeleitet werden. Der Datenserver schickt zusätzlich Awareness-Nachrichten an den VClient in Form von Funktionsaufrufen. Der VClient generiert daraus Visualisierungsnachrichten, und leitet sie an den Client weiter.

Der Datenserver

Der Datenserver baut auf TclDB als unterste Schicht auf. Die Zugriffe auf die Datenbank basieren auf dem Fingermechanismus wie in 4.1.1 definiert[5]. TclDB als Schnitt-

stelle zu einer Einbenutzerdatenbank wird um Mechanismen für den Mehrbenutzerbetrieb für die synchrone Gruppenarbeit erweitert. Diese Erweiterung läßt sich mit den folgenden Punkten zusammenfassen:

- Zugriffskontrolle
- Sperrmechanismen zur Nebenläufigkeitskontrolle
- Generieren von Awareness-Nachrichten.

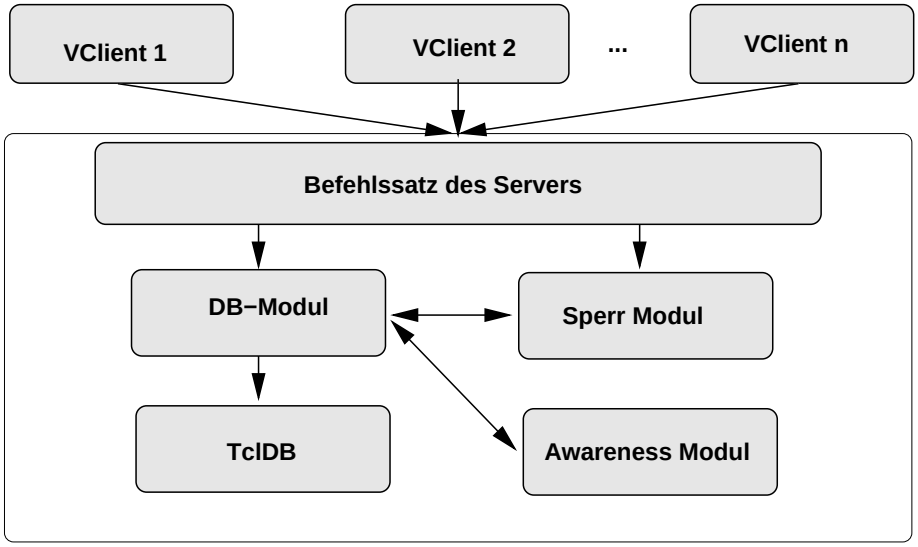


Abbildung 4.4: Der Datenserver

Diese Schicht stellt eine Art *zentrale Einheit* zur Ausführung von kurzen Operationen dar. Die Ausführung einer Operation in einer Anwendung entspricht dem Zutritt in einen kritischen Abschnitt, so daß zu jeder Zeit eine einzige Anwendung eine solche Operation ausführen kann. Die Operationen auf dieser Ebene sind *atomar*, d. h. entweder wird eine Operation zu Ende ausgeführt oder gar nicht.

Für die Bereitstellung von Group-Structural Awareness werden Informationen über angemeldete Benutzer und die von jedem Benutzer benutzten Daten gesammelt. Darüber hinaus werden einige Visualisierungsinformationen, die teilweise vom Benutzer als notwendige *Registrierung* kommen, für diesen Zweck eingesammelt.

Der Datenserver stellt die Basis für die Bereitstellung von Workspace-Awareness bereit. Dabei werden lediglich der Fokus und der Nimbus eines Benutzers vermerkt (siehe Definitionen in Abschnitt 3.5). Visualisierungsnachrichten, die Workspace-Awareness betreffen, werden vom Datenserver aus diesen beiden Informationen generiert und an die entsprechende VClient-Instanz weitergeleitet, wo die eigentliche

[5] Theoretisch kann man TclDB und den Fingermechanismus auf ein anderes Datenbanksystem abbilden. Dies kann in weiteren Arbeiten realisiert werden.

Visualisierung erzeugt und an den Client gesendet wird. Weitere Visualisierungsinformationen sind bei dieser Schicht nicht bekannt. Inhalte mit passenden Formatierungsangaben zur Darstellung eines Dokuments in einem speziellen Viewport werden bei der entsprechenden Instanz der oberliegenden Schicht, dem VClient, generiert.

Abbildung 4.4 zeigt den Aufbau des Datenservers. Zur oberen Schicht, dem VClient, stellt der Datenserver einen ähnlichen Befehlssatz wie TclDB zur Verfügung. Zum Befehlssatz gehören auch Funktionen zum expliziten Abfragen von Awareness-Informationen und zum Setzen und Freigeben von Sperren.

Der Virtuelle Client

Für die Unterstützung von verschiedenen Gerätetypen mit unterschiedlichen Anzeigefähigkeiten, Übertragungsgeschwindigkeit und Rechenkapazität ist auf der Serverseite für jede Anwendung einen Client-Simulator (VClient) zuständig [109, 112].

Der VClient ist der serverseitige Verbindungsstück zum Client. Der Datenaustausch basiert auf XML. Für jede laufende Applikation wird eine VClient-Instanz erzeugt. Sie dient als Übersetzer zwischen einem höheren Sprachsatz, der Schnittstelle zur Client-Applikation, und den technisch aufwendigen Operationen des Datenservers. Vom Client kommen kontextabhängige Schreib- und Navigationsoperationen auf XML-Dokumenten, die durch ein Protokoll festgelegt sind. Diese werden in TclDB-Skripte übersetzt und an den Datenserver weitergeleitet. Mit einem festgelegten Satz von Datenbankoperationen und Awareness-Abfragen kommuniziert der VClient mit dem Datenserver. Auf der anderen Seite schickt der Datenserver Visualisierungsnachrichten, die ebenso aus einem festgelegten Satz von Visualisierungsoperationen bestehen, die der VClient umsetzt. Diese reflektieren die Aktivitäten der anderen Benutzer.

Eine zentrale Aufgabe dieser Schicht ist der Aufbau einer *virtuellen Visualisierung*[6] und die Bereitstellung von generischen intuitiven Navigationsoperationen auf XML-Dokumenten [210]. Dazu werden eNF^2-Tabellen in DOM-Bäume überführt und gehalten. Anhand von Awareness-Nachrichten des Datenservers aktualisiert der VClient die virtuelle Visualisierung und leitet die Veränderungen an den Client in Form von XML-Dokumenten weiter[7], der die reale Visualisierung vornimmt.

---

[6] Der Begriff virtuelle Visualisierung bezeichnet eine interne abstrakte Darstellung, hier als DOM-Baum, die zwar visuelle Effekte festlegt (z. B. Finger zeigt auf Knoten k), deren reale Ausprägung aber dem realen Client überläßt (z. B. Tabellenelemente des Knoten k rot einfärben).

[7] Für Implementierungsdetails und die genaue Struktur dieser Dokumente siehe Abschnitt 5.2.

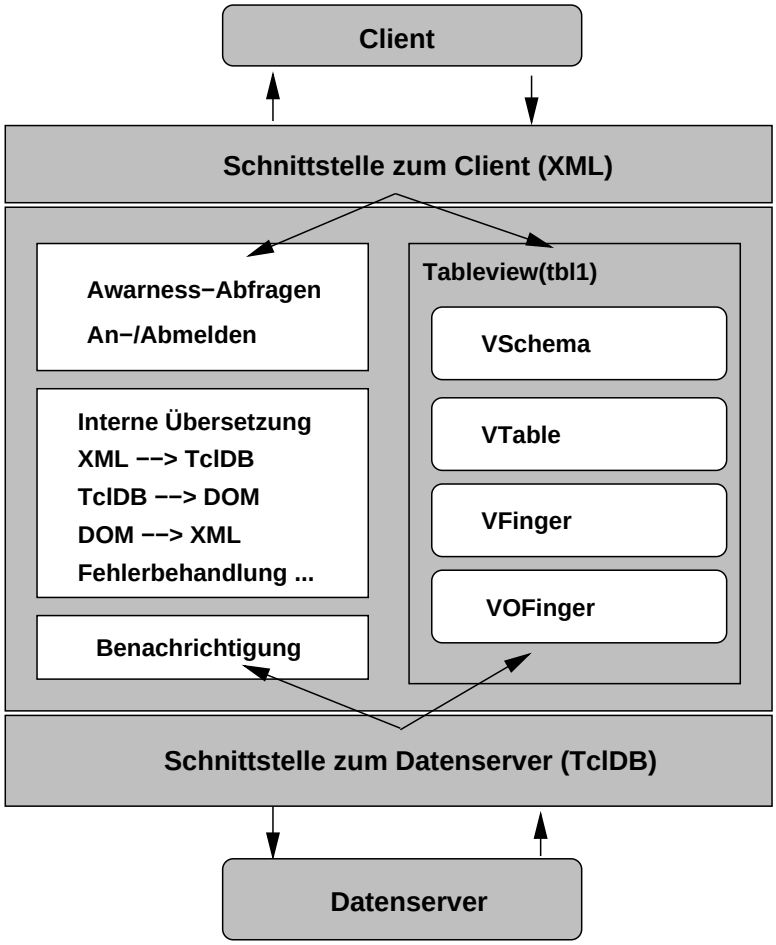


Abbildung 4.5: Aufbau des VClients

4.1.3 Der Client

Für die Entwicklung von Client-Applikationen ist eine Java-Schnittstelle implementiert, die die komplizierten Details der Kommunikation mit dem VClient und die Protokollbehandlung verbergen. Eine Client-Applikation kommuniziert nach außen mit dem Benutzer. Sie fängt seine Eingaben auf und ruft bei Bedarf entsprechende Methoden des VClients auf, um die Aktionen des Benutzers auszuführen.

Auf der anderen Seite bekommt ein Client Visualisierungsnachrichten vom VClient in Form von standardisierten XML-Dokumenten. Für die Anzeige solcher Dokumente können Clients generische Stylesheets [207] verwenden, die eine Standardvisualisierung, etwa eine Tabellendarstellung, erzeugen. Zusätzlich zu diesen generischen Stylesheets kann eine Client-Implementierung eigene Stylesheets für spezielle Visualisierung verwenden. Die Interpretation der XML-Dokumente und Stylesheets wird von Klassen der Clientschnittstelle, oder vom VClient bei leistungsschwachen Geräten,

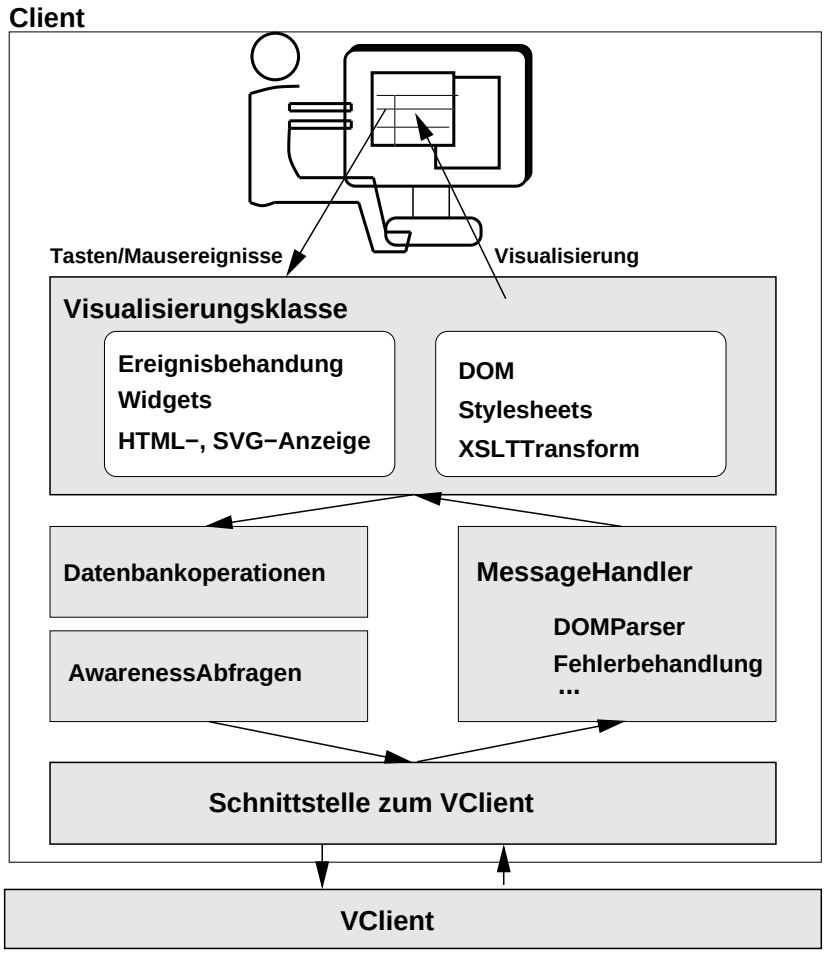


Abbildung 4.6: Aufbau der Clientschnittstelle

durchgeführt. In letztem Fall werden HTML oder SVG verschickt. Dazu werden ein XML-Parser und ein Stylesheet-Prozessor eingesetzt [8].

4.2 Visualisierung

Sehr wichtig für das Visualisierungsmodell ist, daß verschiedene Clienttypen und Anwendungen die Möglichkeit haben, dieselben Daten entsprechend ihrer Anzeigemöglichkeiten darstellen zu können. Aus diesem Grund ist sowohl die Trennung zwischen Inhalt und Visualisierung, als auch die Verwendung eines allgemeinen Sprachstandards zur Beschreibung der Daten wichtig.

[8] In der Implementierung wurden der xerces XML-Parser und der xalan Stylesheet-Prozessor eingesetzt. Siehe `http://xml.apache.org`

Eine eNF^2-Tabelle bildet bei dem VClient einen DOM-Baum. Der bei einem Benutzer sichtbare Teil einer Tabelle wird dem Client als ein XML-Dokument übermittelt. Um verschiedene Visualisierungen zu ermöglichen, werden Attribute im XML-Dokument eingebaut, die die Struktur einer eNF^2-Tabelle wiedergeben [222].

Ein Attribut `nf2type` enthält den Typ des Elements. Vorgesehene Werte sind `set`, `list`, `gtuple`, `ptuple`, usw. Generische Visualisierungen erzeugen daraus z. B. je Tupel eine Zeile über mehrere Spalten hinweg, für Listen und Mengen die übliche vertikale Anordnung als Untertabelle. In [222] wird im Detail die automatische Erzeugung der Werte aus XML-Schema Angaben erläutert. Im interaktiven Betrieb (etwa beim Einfügen eines Knotens) werden die Angaben aber direkt in den DOM-Baum (und damit später in das XML-Dokument) geschrieben. Ferner wird ein zusätzliches Attribut `id` zur eindeutigen Identifizierung einzelner Objekte eingebaut. Zur Darstellung von Fingern im Dokument wird ein weiterer Attribut `bg` vom VClient im DOM-Baum eingebaut.

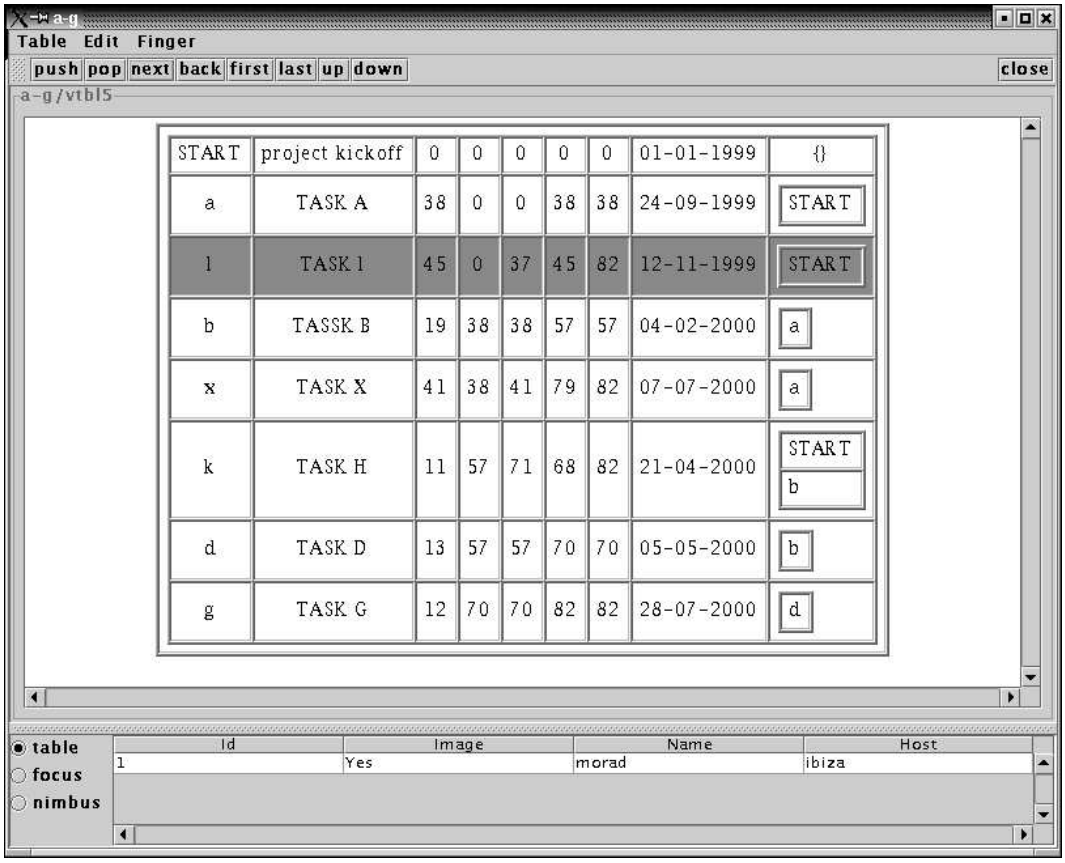


Abbildung 4.7: Die Darstellung des Dokuments durch das HTML-Anzeige-Widget `JTextEditorPane`. Durch Verwenden des Attributes `bg` werden Finger in der Anzeige dargestellt

Für Anwendungen synchroner Gruppenarbeit mit Modifikationen und Navigation auf gemeinsamen Datenräumen, hier die oben genannten eNF^2-Tabellen, ist eine dyna-

mische Visualisierung von zentraler Bedeutung. Durch einen Anwender verursachte Änderungen müssen bei allen betroffenen Clients aktualisiert werden. Dies wiederum muß zuerst bei den zugehörigen VClients in der virtuellen Visualisierung erfolgen, bevor die entsprechenden Ausschnitte aus dem modifizierten DOM-Baum ermittelt und als XML-Dokumente an den realen Client gesandt werden.

Auf dieser Weise erzeugt der VClient eine virtuelle Visualisierung, in der jede Datenbankoperation auf generische Weise ein Visualisierungsgegnstück besitzt. Die Menge der Visualisierungsoperationen bezeichnen wir als Visualisierungssatz. Er ist implementiert als Bibliothek von standardisierten Visualisierungsfunktionen, die Objekte sowie Operationen des Daten- und Interaktionsmodells visualisieren.

Die tatsächliche Darstellung einer Tabelle bzw. von Operationen wird bei einem Client anhand des verwendeten Stylesheets oder sonstiger Visualisierungen realisiert. Dies könnte durch die Verwendung von anderen, eigenentwickelten Stylesheets, oder durch das Parsen des XML-Dokuments und die Visualisierung, z. B. durch den Aufbau einer grafischen Oberfläche etwa in Java oder Tcl/Tk erfolgen.

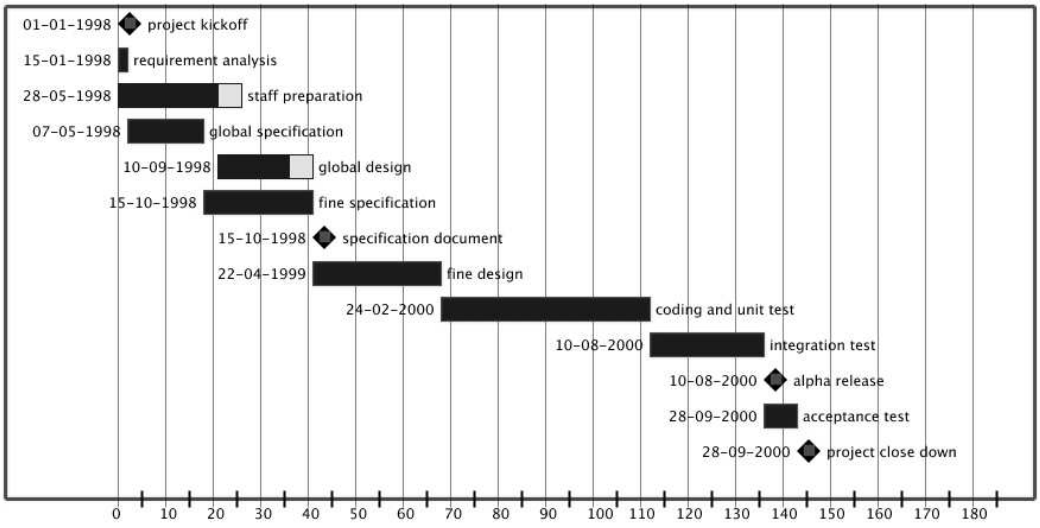


Abbildung 4.8: Die Darstellung des Dokuments durch den **batik** SVG-Viewer

Als Beispiel betrachten wir eine eNF^2-Tabelle für Projektmanagement. Programm B.1 im Anhang zeigt, wie eine solche Tabelle bei dem VClient als XML-Dokument dargestellt wird. Durch die Verwendung von zwei unterschiedlichen Stylesheets wird dieses Dokument auf zwei verschiedene Weisen visualisiert.

Das erste Stylesheet (Programm B.2) erzeugt eine Standarddarstellung eines eNF^2-XML-Dokuments in Form einer geschachtelten HTML-Tabelle (vgl. Abb. 4.7) . Dieses Stylesheet wird für Standardvisualisierung bei dem Client benutzt, falls kein anderes Stylesheet gewählt wird. Das Stylesheet ist generisch und transformiert das Dokument anhand seiner Struktur.

Das zweite Stylesheet (Programm B.3) erzeugt aus dem XML-Dokument eine Darstellung eines Projekts als PERT-Diagramm in SVG[9]. Diese Visualisierung wird deshalb speziell genannt, weil sie eine Interpretation des Inhalts der Elemente vornimmt, hier etwa die Interpretation des DUR-Elements als Zeitdauer eines Projekteschritts. Spezielle Stylesheets können nur auf XML-Dokumente mit demselben Schema angewandt werden.

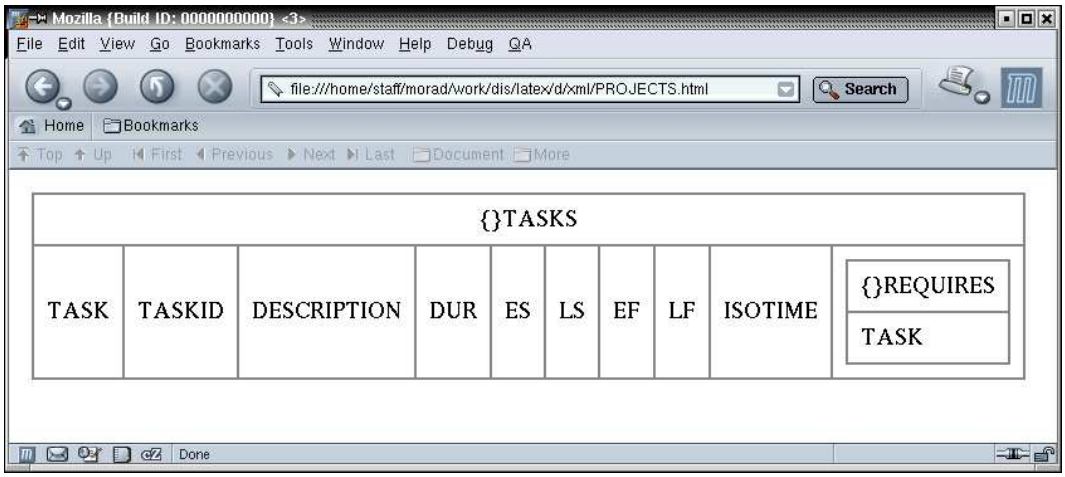


Abbildung 4.9: `Abteilungen.tbl` als HTML-Tabelle dargestellt mit einem Browser

Für die Definition der Struktur eines Dokuments (eNF^2-Schemas) stellt XML zwei Möglichkeiten: DTD, und XML Schema (siehe Abschnitt 2.2.3). Ein großer Vorteil bei dem Einsatz der neueren Sprache, XML Schema, ist, daß XML Schema-Dokumente selbst XML-Dokumente sind, und damit durch ein Stylesheet visualisiert werden können. Der Programmcode in B.4 zeigt die Schemadefiniton einer eNF^2-Tabelle `Projects.scm` als XML Schema Dokument. Durch den Einsatz eines generellen Stylesheets für die Visualisierung von jedem XML Schema kann man die HTML-Darstellung in Abbildung 4.9 erzeugen (siehe Stylesheet B.5).

4.2.1 Generierung von Visualisierung

Für die Generierung von Visualisierungen aus XML-Daten existieren verschiedene Möglichkeiten. Eine allgemeine Lösung ist die Benutzung von XSLT-Stylesheets, um XML-Dokumente in andere XML-basierte Sprachen umzuwandeln. Der Transformationsprozeß kann auf Serverseite oder Clientseite ausgeführt werden.

Die Transformierung auf Serverseite entlastet den Client. Vor allem bei Clienttypen mit geringer Rechenkapazität kann das sogar notwendig sein. Verschiedene VClient-

[9] Das Projekt **batik** der Apache-Gruppe stellt einen SVG-Viewer zur Verf¨ugung, mit dem SVG-Dokumente angezeigt werden k¨onnen (siehe `http://xml.apache.org/batik`).

Typen, die Visualisierungen maßgeschneidert an die Geräte (Speicherkapazität, Prozessor, Grafikfähigkeiten etc.) anpassen, können implementiert werden. Diese Strategie entlastet die Clientgeräte von der Verarbeitung und Transformation der XML-Dokumente in den Anzeigesprachen (HTML, SVG, PDF etc.). Für verschiedene Gruppen von Geräten müssen neue VClients implementiert werden. Geräte mit bestimmten Voraussetzungen können aber in einer Implementierungsklasse eingeschlossen werden. Lediglich die Fähigkeit der Ausführung der jeweiligen Visualisierungsmethoden mit einer sinnvollen Darstellung wird vorausgesetzt. Weitere Softwarebibliotheken zum Parsen und Validieren von XML sowie für die Transformation in anderen Sprachen sind nicht notwendig.

Das Verschicken von XML-Dokumenten hat den Vorteil, daß Client und Anwendungsentwickler die Möglichkeit haben, die Darstellung der Daten unterschiedlich zu realisieren. So können Anwendungen auf leistungsstarken Rechnern grafische Oberflächen aus dem XML-Inhalt erzeugen. Da XML sich als Standard zum Austausch von Informationen immer mehr durchsetzt, ist davon auszugehen, daß verschiedene, auch ,,einfache" Anzeigegeräte, die notwendige Software enthalten werden, um XML-Dokumente zu verarbeiten und in andere Sprachen umzuwandeln.

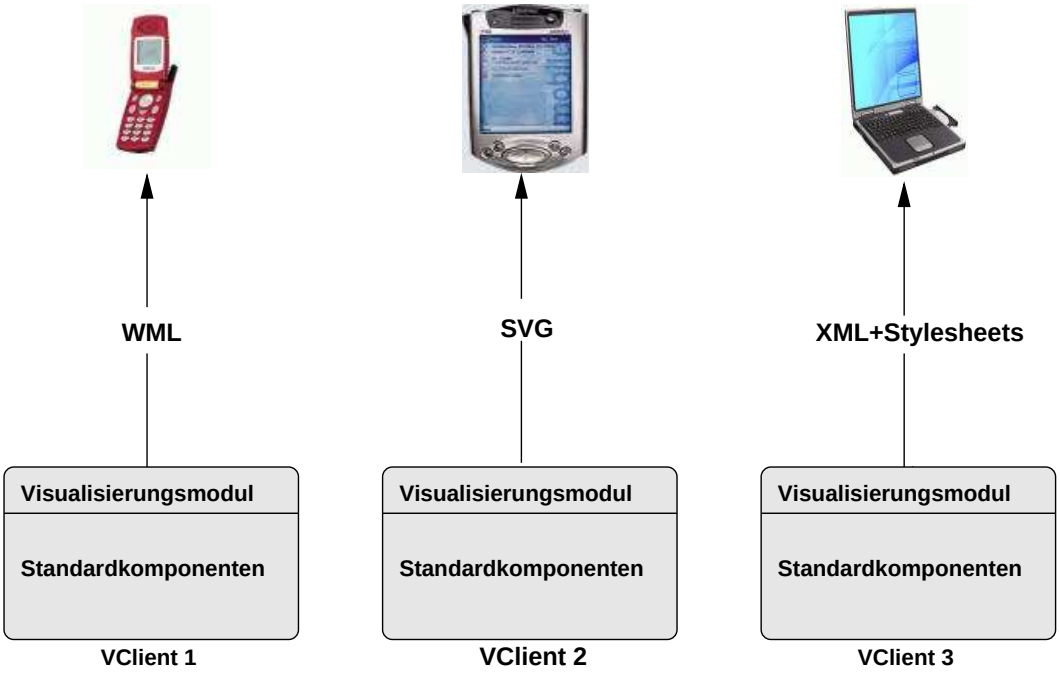


Abbildung 4.10: Verschicken von Visualisierungen an verschiedene Ger¨ate

In der Implementierung dieser Arbeit wird zwischen drei verschiedenen Geräten unterschieden: *pc*, *applet* und *mobile*. An pc-Clients werden XML-Daten übertragen. Bei applet-, und mobile-Geräte werden die Dokumente erst in HTML oder SVG transformiert und dann an dem Client übertragen. Abbildung 4.10 veranschaulicht diesen Ansatz.

4.3 Awareness

Der Datenserver enthält ein Awareness-Modul, das Mechanismen für die Generierung und Anzeige von Workspace-Awareness sowie Group-Structural Awareness (siehe Abschnitt 2.5) zur Verfügung stellt. Obwohl Group-Structural Awareness, also die Information beteiligter Benutzer, ständig vom Datenserver aktualisiert und in der Benutzeranzeige dargestellt wird, ist dies kein zentrales Thema in dieser Arbeit, und wird daher nur kurz behandelt. Workspace-Awaereness wird in Abschnitt 4.3.2 behandelt.

4.3.1 Group-Structural Awareness

Für jeden angemeldeten Benutzer werden spezifische Informationen gesammelt (z. B. sein Name, der Rechner, aus dem er angemeldet ist usw.). Weiterhin werden Informationen über die Tabellen, auf die er zugreift, und die Finger und deren Position auf diesen Tabellen generiert. Mithilfe dieser gesammelten Informationen stellt der Datenserver Funktionen zur Verfügung, mit denen der VClient Group-Structural Awareness abfragen kann.

Das Awareness-Modul *sammelt* Awareness-Informationen und *verwaltet* sie in internen Datenstrukturen für die *Generierung* von Awareness-Nachrichten. Die Gewinnung von Informationen kann entweder *explizit* oder *implizit* sein. Die folgenden Awareness-Informationen werden vom VClient dem Datenserver übermittelt:

- Name, Rechnername usw.
- Für jede Tabelle und jeden Finger einen Objektbezeichner aus dem Namensraum des VClients
- Für jeden Benutzer den Bezeichner des aktiven Fingers auf jeder Tabelle
- Für jeden Benutzer den Fokus auf jeder Tabelle

Abbildung 4.11 zeigt wie man über die Benutzer Id detailliertere Informationen bekommen kann. Einige dieser Informationen sind bei jedem Client repliziert. Andere werden beim Abfragen vom Datenserver geholt. Über die gezeigte Maske kann man den Benutzer kontaktieren oder an ihn Nachrichten verschicken.

Bei der Ausführung von Datenbankoperationen generiert der Datenserver weitere Awareness-Informationen, die wie folgt zusammengefaßt werden können:

- Für jeden Benutzer Objektbezeichner aller Tabellen, auf die zugegriffen wird, und alle Finger des Benutzers auf diesen Tabellen. Die Bezeichner sind aus dem Datenraum des Datenservers.
- Für jeden Finger Besitzer, Zeit usw.
- Alle V-Sperren auf einer Tabelle (siehe Definition 4.1).
- Alle Schreib-Sperren auf einer Tabelle.

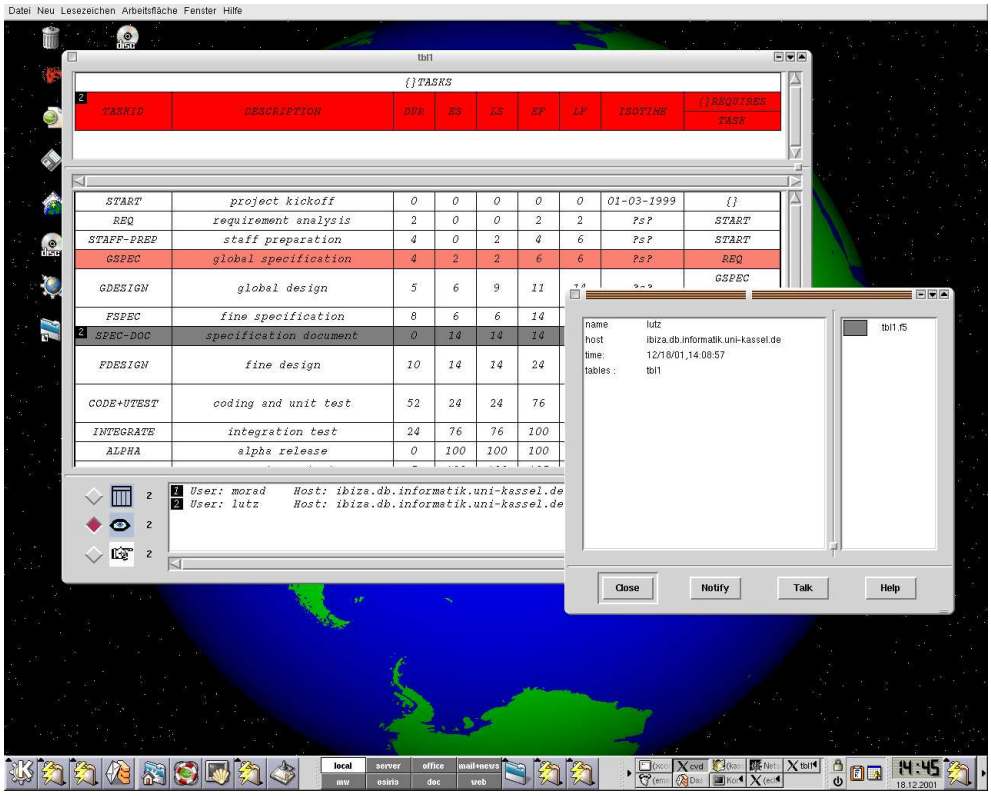


Abbildung 4.11: Social Awareness Anzeige

4.3.2 Workspace-Awareness

Awareness-Informationen werden vom Datenserver zusammengestellt und bei Bedarf für die Benachrichtigung an den VClient in Form von Funktionsaufrufen geschickt. Der VClient trägt mit seinen Funktionen die Veränderungen in der virtuellen Anzeige ein und leitet diese Veränderungen in Form von XML-Dokumenten an den Client weiter. Die Client-Schnittstelle stellt Klassen für die Bearbeitung dieser XML-Dokumente und die Aktualisierung der Anzeige zur Verfügung.

Das Awareness-Modul im Datenserver kann in die folgenden drei Teile unterteilt werden [42] (siehe Abschnitt 2.5):

- Informationen über die Teilnehmer erfassen. Diese können entweder aus den Aktivitäten der Benutzer interpretiert werden oder durch eine explizite Registrierung des Benutzers.
- Aus diesen Informationen konkrete Awareness-Einträge erstellen, aus denen schnell Benachrichtigungen generiert werden können (z. B. welche Benutzer sehen einander, welche arbeiten mit denselben Tabellen usw.)
- Aus den vorhandenen Awareness-Einträgen Awareness-Nachrichten an die interessierten Benutzer generieren und versenden.

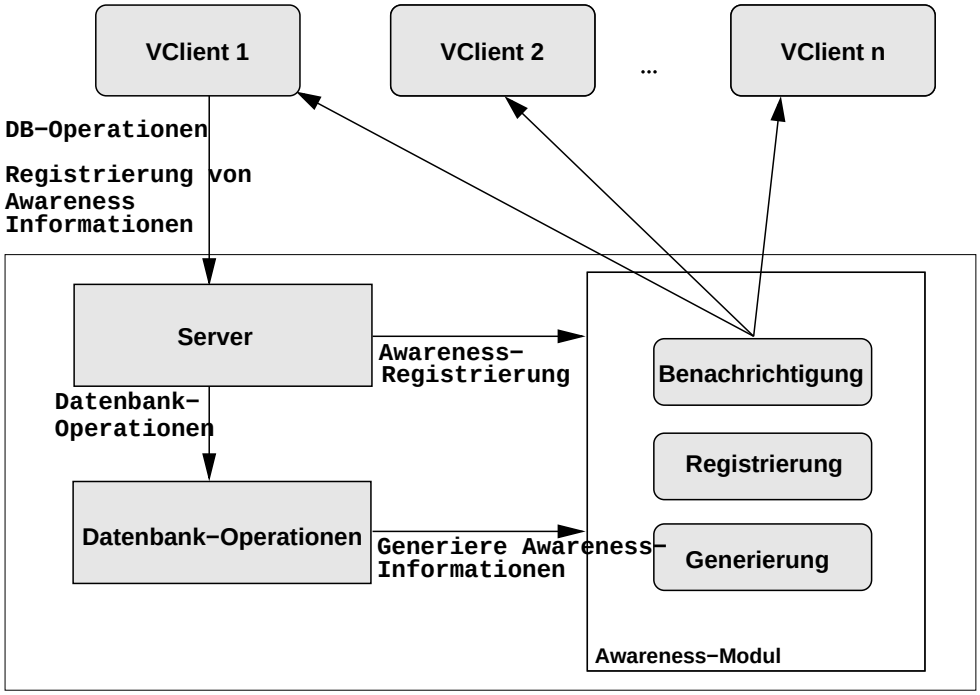


Abbildung 4.12: Struktur des Awareness-Moduls

Objekte im Viewport werden vom Benutzer gesehen und werden daher mit einer Art *Sehsperre*, sogenannte *Notify Locks* oder hier *Visuelle Sperren* (*V-Sperren*), versehen. Diese stellt die Verbindung zwischen der Visualisierung und der Datenbank, d. h. zwischen Awareness und der Kontrolle der Nebenläufigkeit, her.

Definition 4.1 (V-Sperre)
Eine V-Sperre auf ein Objekt bedeutet, daß das Objekt bei dem Benutzer sichtbar ist.

Die Gewinnung von Informationen über Fokus und Nimbus erweist sich durch das zugrunde liegende eNF^2-Datenmodell und den Fingermechanismus als extrem einfach (siehe Abschnitt 3.5).

Bestimmung des Fokus: Beim Generieren der virtuellen Visualisierung werden XML-Elemente mit eindeutigen Id's versehen, die aus der Datenbank generiert werden (RID's, vergl. Abschnitt 4.1.1). Mit Hilfe dieser eindeutigen Bezeichnungen und der Referenzen auf die Position des Benutzers in der virtuellen Anzeige, kann der VClient die sichtbaren Objekte im Viewport eines Benutzers effizient ermitteln und sie dem Datenserver weitergeben.

Bestimmung des Nimbus: Finger sind in der Datenbank bekannt und enthalten Referenzen auf das Objekt, auf das sie zeigen. Für den Datenserver ist der Nimbus durch den aktiven Finger gegeben.

Im Interaktions-Modell ist bisher vorgesehen, daß der Finger typgerecht auf ein Objekt (und damit auch alle seine Unterobjekte) zeigt, also auf ein Tupel, auf eine Menge oder Liste, oder auf ein atomares Element. Damit ist der Nimbus gleichwertig mit der Wurzel des Objektbaums, auf den der Finger zeigt.

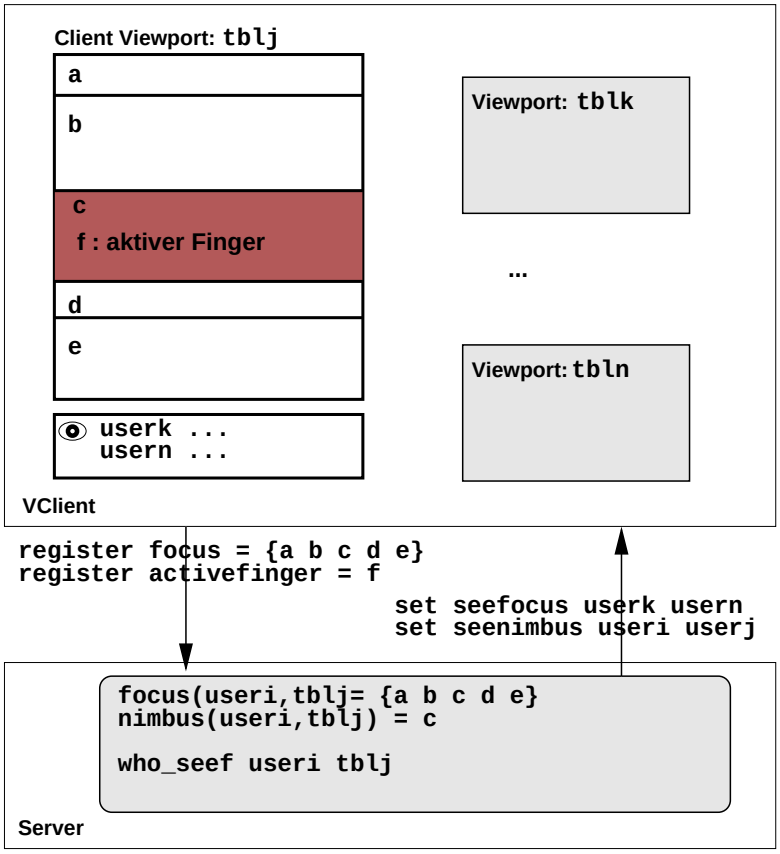


Abbildung 4.13: Generierung von Awareness-Informationen

Allerdings könnte man sich auch vorstellen, daß das Interaktionsmodell mittels der Finger eine Art Selektion und Projektion zuläßt. Damit könnte ein Finger auf eine Auswahl von Attributen eines Tupels oder auf eine Teilmenge, speziell auch eine zusammenhängende Teilliste (Spanne), zeigen, bzw. man würde eine Menge oder Liste von Fingern (eine „Hand") einführen. Daher werden wir im folgenden den Nimbus als Menge von Objekten behandeln, obwohl er in der Regel ein Singleton, nämlich der Bezeichner der Wurzel eines Teilbaums, ist.

Der Datenserver besitzt Funktionen, die mit diesen Informationen Abfragen der Art „*Welche Benutzer sehen meinen Fokus (Nimbus) ?*" oder „*Wieviele Benutzer sind in meinem Fokus aktiv ?*" beantworten können.

Beim Datenserver gesammelte Awareness-Informationen werden automatisch, bei jeder Änderung dieser Informationen, an die betroffenen Benutzer geschickt. Nach

Ausführung einer Fingeroperation erhalten alle die VClients, die den Finger sehen, eine generische Visualisierungsnachricht. Der Datenserver benutzt für solche Benachrichtigungen ein Protokoll, das der VClient versteht. Als Resultat werden die Aktionen der Benutzer bei anderen Teilnehmern auf derselben Stelle der Tabelle angezeigt.

Abbildung 4.13 zeigt wie Informationen bei Überschneidungen von Fokus/Nimbus generiert werden. Ein VClient verwaltet Daten über Viewport und Position von Fingern eines Benutzers. Die im Viewport liegenden Objekte sowie die Objekte, auf die der aktive Finger des Benutzers zeigt, werden dem Datenserver übermittelt. Aus diesen Daten der Benutzern rechnet der Datenserver, welche Benutzer einander sehen und schickt diese Informationen an die relevanten VClients weiter.

Dieses Modell ist an die Anwendungsmöglichkeiten angepaßt und weicht daher stark vom theoretischen Fokus/Nimbus-Modell ab. Schwerpunkt des neuen Modells ist die Schaffung einer Awareness-Architektur, die eine intuitive Arbeitsweise der Benutzer und eine einfach zu verstehende Awareness-Anzeige unterstützt. Die Verwendung des Fingermechanismus für die Ausführung von Datenbankoperationen läßt die Ermittlung des Aktivitätsbereichs eines Benutzers auf einfache und natürliche Art erfolgen. Im folgenden geben wir eine formale Beschreibung des Modells.

Definition 4.2
Sei B die Menge der Benutzer Id's, die zu einem gegebenen Zeitpunkt an der Gruppenarbeit teilnehmen. T sei die Menge der geöffneten Tabellen und O_t die Menge der Objekte in Tabelle $t \in T$. Für eine geöffnete Tabelle bezeichnen wir mit B_t die Menge der Benutzer, die auf die Tabelle zugreifen. Die Funktion $focus : B \times T \rightarrow O_t$ wählt den Fokus eines Benutzers $b \in B$ auf der Tabelle $t \in T$ also, also $focus(b,t) = \{o_1, \ldots o_n\} \subseteq O_t, (n \in \mathbb{N})$. Die Funktion $nimbus : B \times T \rightarrow O_t$ bestimmt den Nimbus von b auf t, also $nimbus(b,t) = \{o_1, \ldots o_m\} \subseteq O_t, (m \in \mathbb{N})$.

Für die Benachrichtigung der Benutzer über die Auswirkungen einer Operation ist die Bestimmung aller Benutzer, die den Nimbus des Ausführenden sehen, notwendig. Ein Benutzer $b \in B_t$ sieht eine Operation *op* vom Benutzer $a \in B_t$ falls gilt: $focus(b,t) \cap nimbus(a,t) \neq \emptyset$. Programm 4.2 zeigt als Beispiel die Funktion `whoSeeNimbus`.

Programm 4.2 *Ermittle alle Benutzer, die den Nimbus von a sehen.*

```
procedure whoSeeNimbus(Benutzer a);
begin
   N := ∅;
   for b ∈ B_t
      for all o ∈ nimbus(a,t) do
         if o ∈ focus(b,t) then
            N := N + b;
      end;
   end;
   return N;
end;
```

4.4 Nebenläufigkeitskontrolle

Für die Kontrolle der Nebenläufigkeit gelten zusammenfassend folgende Punkte:

- Operationen werden ausgehend von den momentan dargestellten Objekten im Viewport ausgeführt. Auswirkungen von Operationen eines Benutzers werden bei anderen Teilnehmern visualisiert.

- Operationen, die von einem Benutzer hintereinander ausgeführt werden, werden von ihm nicht als eine zusammenhängende Transaktion aufgefaßt. Es ist deshalb wünschenswert, jede einzelne Operation unabhängig von vorigen und nachfolgenden Operationen auszuführen. Dabei kann die Ausführung einer einzelnen Operation auf der Visualisierungsebene die Ausführung mehrerer Datenbankoperationen bewirken.

- Die Zusammenfassung von mehreren Operationen zu einer Transaktion muß möglich bleiben. Das kann explizit vom Benutzer bestimmt werden. Parallel laufende Transaktionen müssen untereinander synchronisiert werden.

- Das Blockieren von Transaktionen ist unerwünscht, da der Benutzer an der Ausführung der Transaktion direkt beteiligt ist. Der Benutzer möchte eine Operation oder eine Transaktion ausführen und die Ergebnisse möglichst schnell sehen. Grundsätzlich gilt daher bei Eintritt von Situationen, die die Ausführung einer Transaktion verzögern würden, diese abzubrechen und eine Rückmeldung an den Benutzer zu schicken. Die Entscheidung, die Transaktion nocheinmal zu starten oder sie komplett abzubrechen, bleibt bei dem Benutzer selbst.

- Die Einleitung eines *visuellen Rollbacks* bei Eintritt von Konflikten ist unerwünscht. Trotzdem kann man nicht ausschließen, daß solche eingeleitet werden müssen.

Die einfachste Form zur Sicherung der Korrektheit ist sicher zu stellen, daß zu einem Zeitpunkt ein einziges Gruppenmitglied die Datenbank modifizieren darf und vor der Vergabe dieses Rechts eine Aktualisierung der Visualisierung bei dem entsprechenden Benutzer erfolgt. Dabei spielt es keine Rolle, ob die Vergabe dieses Rechts an jeden Benutzer in einer festgelegten Reihenfolge vergeben wird, oder von einem Benutzer explizit gefordert wird. Ähnlich der Serialisierbarkeit bei Transaktionen in Datenbanksystemen, sichert dieses Vorgehen *per Definitionem* die Korrektheit. Als Lösung der Nebenläufigkeit ist dieses Verfahren jedoch inakzeptabel, da die Parallelität der Arbeit der einzelnen Gruppenmitglieder dadurch gestört wird, daß zu jedem Zeitpunkt ein einziger Benutzer die Datenbank modifizieren kann, wobei andere Gruppenmitglieder passiv zuschauen.

Um ein Verfahren für die Nebenläufigkeitskontrolle zu entwickeln, wird als erstes eine Definition für Korrektheit und Konsistenz angegeben. Im zweiten Schritt werden Fingeroperationen in verschiedene Klassen aufgeteilt und Konflikte zwischen diesen Operationen bestimmt. Aus diesen beiden Hauptschritten wird durch die Einführung von Sperrmechanismen ein Verfahren für die Lösung von nebenläufigen Operationen/Transaktionen entwickelt.

4.4.1 Gruppenkommunikationssysteme

Für die Behandlung der Nebenläufigkeit müssen einige Zusicherungen für die zugrundegelegten Kommunikationssysteme vorausgesetzt werden [118]. Gruppenkommunikationssysteme (*Group Communication Services*) ermöglichen die Kommunikation einer Gruppe von Prozessen mit gewissen Zusicherungen [48, 68, 37, 118].

Für unsere Betrachtungen ist der Spezialfall von Interesse, bei dem Anwendungen mit einem zentralen Server über die Zwischenschicht (einem VClient) kommunizieren. Direkte paarweise Kommunikation der Anwendungen untereinander ist dagegen nicht vorgesehen.

Der wichtigste Teil eines Gruppenkommunikationssystems ist der Mitgliedschaftsdienst (*Group Membership Service*) [68, S. 173]. Er bestimmt, welche Anwendungen zu einer Gruppe gehören, d. h. er eine gemeinsame Sicht (engl. *view*) des Systemzustands benötigt. Diese Sicht wird mittels Botschaften hergestellt, d. h. die gruppenmitglieder werden über Operationen benachrichtigt und führen diese lokal aus. Im Gegensatz dazu führt in unserem Modell der Datenserver alle Operationen zentral aus. Mitglieder einer Gruppe sind alle Teilnehmer, die einen nicht-leeren Durchschnitt der Foki bekommen damit die Auswirkung einer Operation sehen.

Die Komponente des Gruppenkommunikationssystems, die für den Übertragungsdienst zuständig ist, heißt *Multicast Service* [48]. Hierzu werden Zusicherungen bzgl. Übertragung und richtige Reihenfolge der Nachrichten gemacht. Diese Zusicherungen sind vom jeweiligen Anwendungsbereich abhängig und variieren daher vom System zu System. In unserem Modell stellen wir die folgenden Voraussetzungen [37, 48, 68]:

1. Nachrichten kommen in der Reihenfolge an, in der sie verschickt werden (*FIFO Delivery* [48, S. 447]).
2. Eine Nachricht, die vom Server an mehreren Clients verschickt wird, enthält einen eindeutigen Zeitstempel. Dieser wird vom Datenserver erzeugt, und hat bei derselben Nachricht den selben Wert bei allen Clients, die die Nachricht bekommen [48, S. 448].
3. Eine Nachricht eines Clients an den Datenserver trägt den Zeitstempel der letzten Visualisierungsnachricht vom Datenserver an diesem Client.

Weiterhin verlangen wir keine sichere Übertragung. Diese kann nur dadurch gesichert werden, daß der Sender auf Rückmeldung vom Empfänger für eine maximale Zeitperiode wartet (*Acknowledge Messages*). Der VClient ermittelt stattdessen, wenn ein zugehöriger Client nicht mehr erreichbar ist (*failure detector* [48, S. 458]), und beendet in diesem Fall die Verbindung. Clients gelten als erreichbar solange ihr korrespondierender VClient erreichbar ist.

4.4.2 Kooperationsformen

Für die Bearbeitung des gemeinsamen Informationsraumes können Benutzer unterschiedlich vorgehen. Intuitiverweise würden sie mit einem Finger navigieren und ihre Operationen hintereinander ausführen [200]. Mit Hilfe von zusätzlichen Awareness-Informationen würden die Gruppenmitglieder auf die Aktivitäten der anderen reagieren und so eine Koordinierung der Gruppenarbeit erreichen [63].

Bei der Ausführung solcher Operationen soll jedem Gruppenmitglied das Gefühl gegeben werden, daß seine Operationen vom Server sofort entgegen genommen werden, wobei mögliche Konflikte auf der Implementierungsebene entdeckt und vermieden werden müssen. Weiterhin muß zugesichert werden, daß jede Operation vom Benutzer mit voller Absicht ausgelöst wurde. D. h. zusichern, daß der Benutzer die aktuellsten Informationen bzgl. seiner Operation zum Zeitpunkt der Auslösung sieht. Solche Zusicherung hat sowohl einen juristischen Hintergrund (Beweisführung der Art ,,Benutzer hat Vertragsbedingung gesehen"), als auch Korrektheitsgründe (verhindern, daß Benutzer eine Operation ausführt mit einem von ihm nicht beabsichtigten Ergebnis [88]). Da diese Operationen aus der Visualisierung ausgelöst werden, werden sie hier *visuelle Operationen* genannt.

Besteht der Wunsch, mehrere Operationen miteinander zu verbinden (,,trage mich in XML-Kurs nur dann ein, wenn Eintrag in UNIX-Kurs auch möglich ist"), so muß dem Benutzer oder dem Anwendungsentwickler die Möglichkeit gegeben werden, Transaktionen auszuführen. Solche Art von Transaktionen können vom Benutzer gestartet und von ihm während der gesamten Ausführung gesteuert werden. In einer Gruppenumgebung müssen die Auswirkungen dieser Transaktionen natürlich von den anderen Gruppenmitgliedern beobachtet werden können. Sie werden hier deshalb *visuelle Transaktionen* genannt.

4.4.3 Visuelle Operationen

Ausgangspunkt für die Definition der Korrektheit und Konsistenz ist die im Abschnitt 3.6.3 für die synchrone Gruppenarbeit auf gemeinsamen Datenräumen gemachte Grundaussage: Unerwünscht sind Auswirkungen des parallelen Betriebs, die

den Einzelnen dazu veranlassen, nicht reversible Aktionen zu unternehmen, die er so nicht ausgeführt hätte, wenn er den korrekten aktuellen Zustand des Systems gekannt hätte. Intuitiv läßt sich diese Aussage in dem gebräuchlichen Satz zusammenfassen: „Ich hätte das nicht gemacht, wenn mir die aktuelle Situation bewußt bewesen wäre (*I would not have acted like this, if I had been aware of the actual situation.*)"

Wie in 3.6.4 beschrieben, muß man wegen der fehlenden Isolation zwischen dem Zustand der gespeicherten Datenbasis und dem einem Benutzer auf der Benutzeroberfläche angebotenen Ausschnitt der Datendarstellung, hier Visualisierung genannt, unterscheiden. Ähnlichkeiten zur Cache-Konsistenz in verteilten Anwendungen (siehe etwa [177, 74, 151, 34]) und zur View-Materialisierung sind dabei offensichtlich.

Auf der Benutzeroberfläche sehen die Benutzer eine Visualisierung eines Ausschnitts der physischen Datenbank, die sie als Ausgangspunkt für jede weitere Operation auf die Datenbank betrachten.

Definition 4.3 (Visualisierung)
Die Visualisierung eines Objekts O bei dem Benutzer B wird mit $V(B, O)$ bezeichnet. Die Visualisierung von allen sichtbaren Objekten eines Dokuments T am Bildschrim eines Benutzers B, $Vp(B, T)$ wird der Viewport von B auf T genannt. Die Menge der Datenbankobjekte, die zur Darstellung im Viewport ausgewählt sind, wird mit $fokus(B, T)$ bezeichnet. Die Abbildung, die Datenbankobjekte in Visualisierungsobjekte überführt, wird $visual$ genannt.

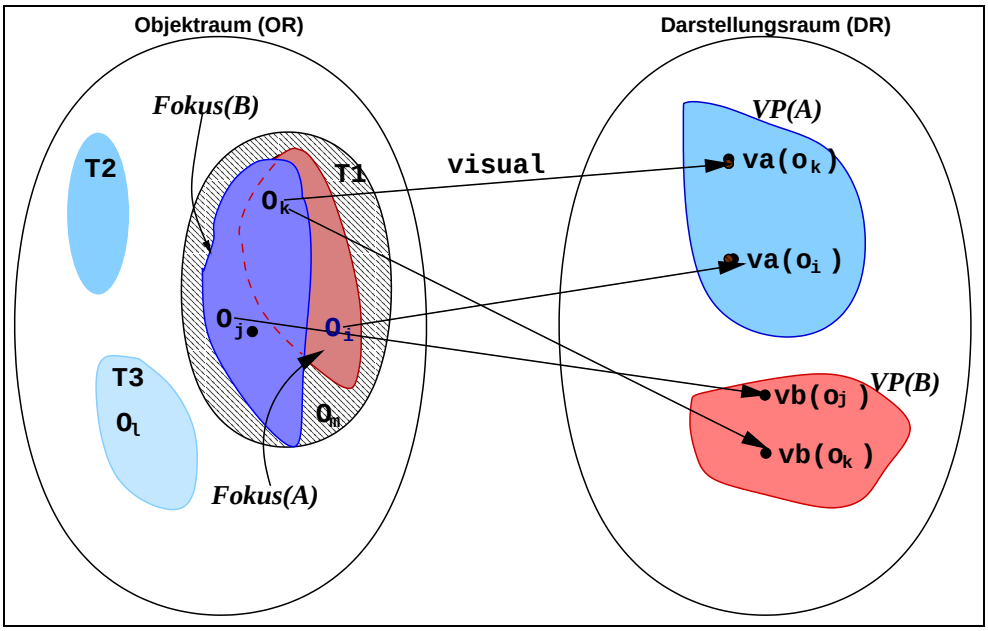


Abbildung 4.14: Visualisierungsabbildung. O_k liegt in Viewport von A und B. O_m wird von beiden Benutzern nicht gesehen. O_i wird nur von A, O_j nur von B gesehen

Die Ausführung von Operationen auf der Benutzeroberfläche wird hier mit *Benutzerabsicht* (*user intention*) bezeichnet, und von der tatsächlichen Operation auf der Datenbankebene unterschieden [179, S. 435]. Benutzer erwarten, daß ihre Absichten tatsächlich auf der Datenbankebene realisiert werden [179, 88]. Repräsentiert die Visualisierung der sichtbaren Objekte bei einem Benutzer den aktuellen Inhalt dieser Objekte in der Datenbank, so spricht man von einer *aktuellen Visualisierung*. Wichtig ist dabei, zwischen Objekten, die der Benutzer tatsächlich sieht und dem ganzen Dokument zu unterscheiden.

Eine Visualisierung stellt immer einen früheren Datenbankzustand dar. Dementsprechend ist eine Visualisierung aktuell, wenn sich dieser Zustand noch nicht geändert hat. Den Viewport eines Benutzers B zu einem gegebenem Zeitpunkt t bezeichnen wir mit $V_t(B,T)$ (*last view* [48, S. 449]). Den Tabellenzustand zum Zeitpunkt t bezeichnen wir mit T_t. Für jeden Zeitpunkt t_1 stellt $V_{t_1}(B,T)$ den Zustand des Fokus von B zu einem früheren Zeitpunkt t_0 dar. Also $V_{t_1}(B,T) = visual(fokus(B,T_{t_0}))$, mit $t_0 < t_1$.

Definition 4.4 (Aktuelle Visualisierung)
Ein Viewport $V(B,T_{t_n}) = visual(fokus(B,T_{t_m}))$ *ist aktuell, wenn es keinen Tabellenzustand* $T_{t_{m'}}$, $t_m < t_{m'} < t_n$ *gibt, mit* $V(B,T_{t_n}) \neq visual(fokus(B,T_{t_{m'}}))$.

Ähnlich ist die Visualisierung eines einzelnen Objekts aktuell, wenn sie den letzten Zustand des Objekts darstellt. Die Bewahrung der Konsistenz aus der Visualisierungssicht beschränkt sich darauf, dafür zu sorgen, daß die Visualisierung der Daten mit dem Zustand der Daten im gemeinsamen Informationsraum übereinstimmt. Die Konsistenz der Daten im Datenbanksinne darf aber nicht verletzt werden. Die Konsistenz bei der synchronen Gruppenarbeit setzt sich daher zusammen aus der Konsistenz der Visualisierung und der des gemeinsamen Informationsraumes. Aus den beiden Definitionen leiten wir die Konsistenz eines Viewports ab:

Definition 4.5 (Konsistenz)
Das Viewport eines Benutzers ist konsistent, falls der zugehörige gemeinsame Informationsraum konsistent und die Visualisierung aktuell ist.

Verlangt man, daß der ganze Viewport bei der Abgabe einer Schreiboperation aktuell ist, so stellt man sicher, daß der Benutzer den Zusatand der Objekte, die er modifizieren will kennt und daher diese Operationen auch tatsächlich so beabsichtigt. In der Praxis ist es aber nicht erforderlich zu verlangen, daß der gesamte Viewport aktuell ist. Es wurde reichen, wenn das Objekt (die Objekte), das durch die Operation modifiziert wird, zur Zeit der Abgabe der Operation aktuell ist. Dazu muß bei jeder Fingeroperation festgestellt werden, welches das betroffene Objekt ist.

Sehr wichtig ist die Unterscheidung zwischen dem *Sehen* von Datenbankobjekten im Viewport und dem *Lesen* dieser Objekte aus der Datenbank. Lesen bedeutet *die*

Ausführung einer Leseoperation auf der Datenbankebene. Dabei entspricht der gelesene Wert genau dem Zustand des Objekts zum Zeitpunkt der Ausführung. Während das Lesen eines Objekts gleichzeitige Modifikationen von anderen Benutzern ausschließt, können Objekte, die von anderen Benutzern gesehen werden, von einem anderen Teilnehmer modifiziert werden, sofern die Aktualisierung der Anzeige entsprechend dieser Modifikationen ein Teil der Operation ist.

Definition 4.6 (Sehen)
Ein Benutzer sieht ein Objekt, falls das Objekt im Viewport des Benutzers liegt.

Das Sehen eines Objekts im Viewport bedeutet genau zweierlei:

- Operationen eines Benutzers auf Objekten, können vom gezeigten Inhalt des Objekts im Viewport ausgehen, allerdings ohne Zusicherung, daß die Objekte nicht parallel zur eigenen Operation modifiziert werden.
- Änderungen an diesem Objekt durch andere Benutzer werden aktualisiert.

Als Beispiel denke man an eine Anmeldeliste für Vorlesungen und Übungen. Ein Teilnehmer A (Student) trägt den eigenen Namen in eine Übungsgruppe ein und sieht dabei den angekündigten Tag und die Uhrzeit der Gruppe. Es wäre nicht verboten, wenn Teilnehmer B (der Gruppenleiter) parallel zu dieser Operation den Tag ändert. Das CSCW-System ist verpflichtet, die Tagesänderung bei A sofort zu visualisieren (im übrigen auch den neuen Teilnehmereintrag bei B). In der Regel sehen beide, der eintragende Student und der Gruppenleiter, diese Änderungen sofort.

Will A oder B vor einer Änderung während der Eintragung sicher sein, müßte er das ganze Übungstupel (Termin und Liste der Teilnehmer) sperren, z. B. A den Termin zum Lesen und die Liste zum Schreiben sperren, B umgekehrt.

Hier ist klar, daß es eine Unterscheidung zwischen *bewußtem Sehen*, bei dem der Benutzer dem System explizit mitteilt „ich sehe dieses Objekt und führe Operationen ausgehend von dessen Inhalt aus" und *unbewußtem Sehen*, bei dem der Benutzer zwar das Objekt sieht, aber dies nicht explizit sagt, gibt. Bewußtes Sehen entspricht dem Lesen innerhalb einer Transaktion im Datenbanksinn und soll Modifikationen anderer Teilnehmer ausschließen. Unbewußtes Sehen dagegen veranlaßt das System nur zur Aktualisierung der gesehenen Objekte, verhindert aber keine weiteren Modifikationen an dem Objekt.

Die Ausführung einer Fingeroperation auf der Visualisierungsebene wird als *visuelle Operation* bezeichnet. Man kann die gesamte Ausführung einer visuellen Operation in fünf Schritte aufteilen: *Auslösen*, *Senden*, *Ausführen*, *Empfangen* und *Visualisieren* [63, 88, 179]. Benutzer assoziieren das Erzeugen einer Operation mit ihrer Ausführung.

Definition 4.7
Den Zeitpunkt der Auslösung einer Operation O durch einen Benutzer B (im Viewport von B) bezeichnen wir mit $t_g(O_B)$, den Zeitpunkt der Beendigung der Operationsausführung beim Datenserver mit $t_e(O_B)$, den Zeitpunkt der Visualisierung der Operation O_B im Viewport eines Benutzers A mit $t_v^A(O_B)$.

Der Datenserver führt visuelle Operationen verschiedener Benutzer sequentiell aus. Obwohl einzelne Operationen scheinbar unabhängig voneinander sind, besteht zwischen zwei visuellen Operationen ein Konfliktpotential. Die Fehlerquelle liegt darin, daß der Benutzer sie aus einer nicht aktuellen Visualisierung ausführt (es entstehen unbeabsichtigte Ergebnisse [88, 179]).

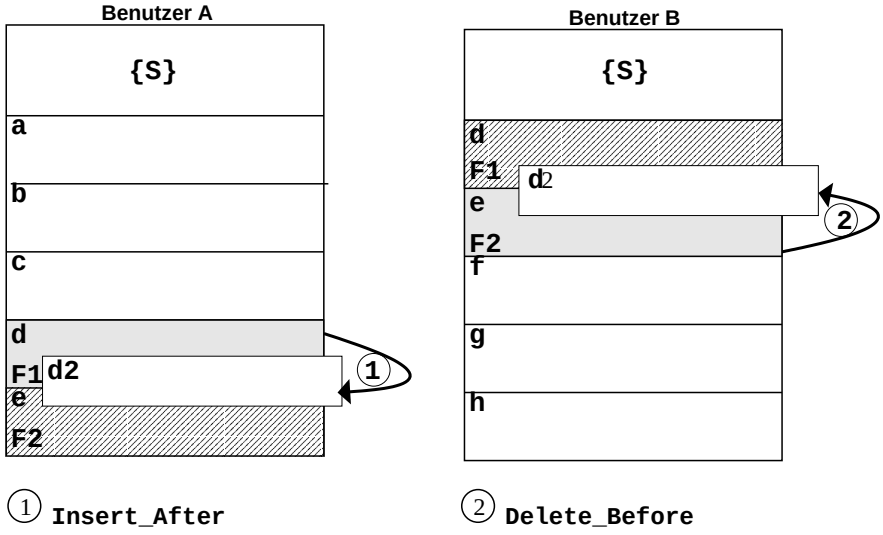


Abbildung 4.15: Beispiel f¨ur unbeabsichtigte Auswirkung

Als Beispiel, wie solche Konflikte zu nicht erwarteten Ergebnissen führen, betrachten wir Abbildung 4.15. Der Finger vom Benutzer A steht auf dem Tupel d der Menge S. Der Finger von B steht auf dem Tupel e. A erzeugt zum Zeitpunkt 1 die Operation $o_1 =$ `Insert_After`. Diese wird zum Zeitpunkt 3 bei dem Server ausgeführt und ein neues Tupel d_2 in der Menge S eingefügt. Zu diesem Zeitpunkt erzeugt Benutzer B die Operation $o_2 =$ `Delete_Before`. Die Absicht von B ist das Tupel d zu löschen. Stattdessen wird aber das neu eingefügte Tupel d_2 gelöscht.

Zwei visuelle Operationen stehen im Konflikt zueinander, falls sie dasselbe Objekt modifizieren wollen, wobei die Auswirkungen der ersten Operation erst nach der Erzeugung der zweiten Operation sichtbar werden, d. h. wenn eine dieser Operationen erzeugt wird, bevor die zweite visualisiert wurde.

Konflikte bei Fingeroperationen

Zur Lösung der Nebenläufigkeit müssen Konflikte zwischen Fingeroperationen bestimmt werden (siehe Tabelle 4.1). Dazu müssen Fingeroperationen auf eNF^2-Relationen auf den klassischen `read`-, `write`-Operationen abgebildet werden und in Abhängigkeit der Fingerposition die zugehörige Granularität der Objekte, auf die sich eine Operation auswirkt, bestimmt werden [101].

Operationen auf der Visualisierungsebene bewirken in manchen Fällen die Ausführung mehrerer Operationen auf der Datenbankebene. Auf der Visualisierungsebene führen Benutzer ausschließlich zwei Arten von Operationen auf dem gemeinsamen Informationsraum aus: Navigations- und Schreiboperationen. Leseoperationen werden nicht explizit vom Benutzer aufgerufen, sondern entstehen implizit z. B. aus der Navigation eines Fingers zu einer, bis zu diesem Zeitpunkt unsichtbaren Stelle im Dokument oder durch Scrollen. Fingeroperationen können in den folgenden drei Kategorien unterteilt werden:

Navigationsoperationen: Positionieren einen Finger auf ein neues Datenobjekt. Navigationsoperationen sind `go`, `push` und `pop` mit all ihren Optionen `back`, `next` etc.

Schreiboperationen: Sind Operationen, die den Zustand eines Objekts, etwa den Wert eines atomaren Attributs oder die Kardinalität einer Menge, modifizieren. Schreiboperationen sind `insert`, `delete` und `set` mit den dazugehörigen Optionen.

Leseoperationen: Leseoperationen sind Operationen, die eine Information über ein Datenobjekt liefern. Dies kann der Wert eines atomaren Attributs, Kardinalität einer Menge, oder Abfragen über den Zustand eines Objekts sein. Leseoperationen sind `get`, `info` und `is` mit den dazugehörigen Optionen.

Es ist klar, daß Finger-Leseoperationen einem `read`und Finger-Schreiboperationen einem `write` entsprechen. Die zentrale Frage bleibt : *Welche Bedeutung hat Navigation?* Um diese Frage zu beantworten, müssen die folgenden Punkte geklärt werden:

Wird jede Navigationsoperation bewußt ausgeführt?
In der Regel navigieren Benutzer, um von einer Position zur nächsten zu gelangen. Dabei wandern sie über mehrere Objekte. Die Frage kann man also klar mit „*nein*" beantworten. Die Positionierung eines Fingers auf einem Objekt kann vorübergehend erfolgen, ohne eine weitere Absicht damit zu verbinden.

Will man mit der Positionierung eines Fingers immer etwas schreiben?
Auch diese Frage kann klar mit „*nein*" beantwortet werden. Finger werden sowohl zur Navigation als auch zur Ausführung von Schreiboperationen benutzt. Die Navigation kann allgemein sowohl zum „Stöbern" (Lesen) als auch zum Editieren dienen.

Lesen Benutzer Objekte auf denen sie navigieren?
Natürlich sehen (in der Regel) Benutzer wohin sie navigieren. Sie beabsichtigen aber nicht, mit jeder Navigationsoperation ein Objekt für sich zu „reservieren" und andere Teilnehmer daran zu hindern, dieses Objekt zu modifizieren. Die Positionierung eines Fingers auf einem Objekt bedeutet daher auch keine „bewußte" Leseoperation.

Die Benutzer können also verschiedene Absichten mit der Navigation haben. Da diese allein aus der Navigation nicht zu bestimmen sind und die Zuordnung von Navigationsoperationen zu Lese- Schreiboperationen die Parallelität der Arbeit enorm stören würde, wird *der Positionierung eines Fingers keine feste semantische Bedeutung zugeordnet* [151, S 160]. Vielmehr gilt die Grundregel, Benutzer geben ihre Absichten explizit an.

Weiterhin muß die Granularität einer Operation bestimmt werden [101, 88]. Genauer: *Auf welches Objekt bezieht sich eine Schreib-, Leseoperation?*

Anders als bei einer Relation im relationalen Datenmodell, kann auf einem komplexen Element beim eNF^2-Datenmodell eine *Leseoperation* ausgeführt werden. Z. B. liefert die Anweisung `fid card` die Anzahl der Tupel in einer Menge, was offenbar vom Zustand der Menge abhängig ist. Dementsprechend sind Schreiboperationen auf mengenwertige Elemente definiert. Wir fassen daher das Einfügen/Löschen von Tupeln in Mengen, anders als einem `INSERT` [10] in eine Relation, als eine Schreiboperation **auf diese Menge** auf.

Die Bestimmung von Konflikten zwischen Fingeroperationen bei dem vorliegenden eNF^2-Datenmodell hängt von der Art der Operation und der Struktur des betroffenen Objekts ab.

Definition 4.8 (Konflikäre Fingeroperationen)
Die folgenden Regeln bestimmen, welche Fingeroperationen beim eNF^2-Datenmodell in Konflikt zueinander stehen:

- *Löschen und Hinzufügen in mengenwertigen Objekten (`set`, `list`, `mset`) durch zwei Anwender.*
- *Löschen/Hinzufügen in mengenwertigen Objekten und gleichzeitiges Lesen von Informationen über dasselbe Objekt.*
- *Löschen eines komplexen Objekts und Modifizieren eines seiner Unterobjekte.*
- *Zwei Schreiboperationen auf einem atomaren Objekt.*
- *Eine Schreib- und eine Leseoperation auf einem atomaren Objekt.*

[10] Ein INSERT ist nicht eindeutig als eine Schreiboperation aufgefaßt. Siehe das sogenannte *Phantom Phenomenon* [170, 103]

Definition 4.9 (Unbeabsichtigte Operation)
Für zwei Operationen O_1 und O_2 von zwei verschiedenen Benutzern B_1 und B_2 auf ein Objekt X ist O_2 bezüglich X unbeabsichtigt, *falls O_1 und O_2 konfliktäre Fingeroperationen sind und $t_g(O_2) < t_v(O_1)$ gilt.*

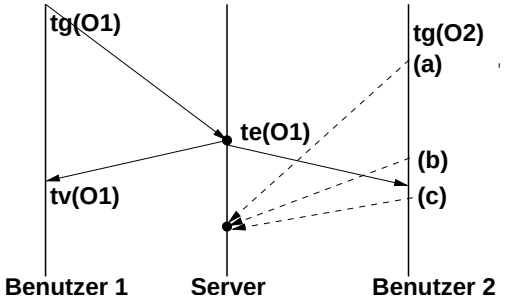


Abbildung 4.16: Ablauf von visuellen Operationen

Abbildung 4.16 zeigt die verschiedenen Möglichkeiten, wann eine Operation, relativ zur Ausführung einer anderen, erzeugt wird. In den Fällen (a) und (b) gilt O_2 als unbeabsichtigt. Im Fall (c) darf O_2 ausgeführt werden, da der Benutzer die Visualisierung von O_1 gesehen hat, bevor er die Ausführung von O_2 auslöste.

In der Regel will man zusichern, daß eine Operation genau das bewirkt, was damit beabsichtigt wird. Eine Operation ist *beabsichtigt*, wenn das entsprechende Objekt zum Zeitpunkt ihrer Auslösung aktuell ist.

Definition 4.10 (Korrekte Visuelle Operation)
Eine Visuelle Operation ist korrekt, wenn sie beabsichtigt ist.

Awareness-Zusicherung mit Zeitstempelverfahren

Zur Erfüllung des Korrektheitskriteriums für visuelle Operationen, wonach diese nur in Kenntnis des neusten, verfügbaren Systemzustands erfolgen sollen, führen wir ein einfaches Prüfverfahren ein, das kompatibel mit der Definition der Lamport-Uhr [124] ist.

Dazu führt der Datenserver eine Uhr (Zählregister) mit der sich Zeitstempel erzeugen lassen. Wurde eine Operation ausgeführt und wird deren Visualisierung durch die relevanten VClients verschickt, so enthalten alle diese Botschaften (Dokumente) den gleichen, aktuellen Zeitstempel des Servers (siehe Abschnitt 4.4.1). Danach wird der Zähler beim Server um 1 hochgezählt.

Clients registrieren jeweils nur den letzten empfangenen Zeitstempel. Das Überholen von Botschaften des Servers, d. h. der Empfang einer Botschaft mit einem Zählerwert $<$ der letzten Botschaft, wird hier ausgeschlossen (siehe Abschnitt 4.4.1).

Jede vom Client an den Server geschickte Botschaft enthält nun zusätzlich den gerade gültigen Zeitstempel beim Client und damit eine Information über die zuletzt empfangene Visualisierung.

Beim Empfang im Server prüft dieser, ob die Operationsauslösung auf einer nicht aktuellen Visualisierung beruht, d. h. ob es eine konfliktäre Operation auf einem Datenobjekt X gibt, deren Visualisierung beim Client noch nicht zum Auslösungszeitpunkt der gerade zu prüfenden Operation eingetroffen war.

Zur Prüfungsmethode gibt es zwei Verfahrensmöglichkeiten.

Zeitorientiert: Es gibt ein Visualisierungslog beim Server, das in die Vergangenheit hinein für jede Botschaft ein Paar (Zeitstempelwert, betroffenes Objekt) speichert[11]. Jetzt geht man solange zurück bis man entweder das relevante Datenobjekt X findet oder der Zeitstempel vor dem empfangenen liegt. Findet man ein X mit Zeitstempel größer dem empfangenen, dann war dessen Visualisierung noch nicht aktualisiert worden, die Operation also in Unkenntnis der neueren Operation ausgeführt worden. Man sollte sie daher abweisen. Sonst war die Visualisierung aktuell. Das Log kann abgeschnitten werden für alle Zeitstempel die kleiner sind als das Minimum der empfangenen Zeitstempel über alle Clients. Das ist allerdings problematisch, wenn Clients sich nicht melden (inaktiv, abgehangen, sehr langsam).

Datenorientiert: Bei jedem Objekt X (entweder direkt im Datenraum, oder in einer geeigneten Datenstruktur, z. B. Hash-Tabelle) wird der Zeitstempel der letzten verschickten Visualisierung gespeichert. Bei Eingang einer Operation auf X prüft man bei X, ob der dort eingetragene Zeitstempel größer ist als der empfangene Zeitstempel. Wenn ja, sollte man die Operation abweisen, weil der Client diese letzte Visualisierung nicht berücksichtigt hat. Anderenfalls wird die Operation zugelassen und bei Versand der Visualisierung wird der aktuelle Zeitstempel des Servers bei X eingetragen. Im Falle einer seperaten Datenstruktur kann das Paar (X, Zeitstempel) gelöscht werden wie oben, d.h. wenn der Zeitstempel bei X kleiner ist als das Minimum aller empfangenen Zeitstempel über allen Clients. Ein fehlender Eintrag signalisiert dann ebenfalls, daß die Visualisierung von X bei allen Clients bereits vor Auslösung der gegenwärtigen Operation erfolgte.

Der Aufwand dieses Verfahrens ist proportional zur Anzahl der beim Server ausgeführten Operationen in der Zeitspanne, die der langsamste Client braucht zwischen einem Visualisierungsversand beim Server und der nächsten Botschaft des Clients nach Eingang dieser Visualisierungsbotschaft.

Satz 4.1 *Das Zeitstempelverfahren sichert zu, daß Operationsauslösung stets auf aktuellen Visualisierungen beruhen. Andere, auf nicht aktuelen Sichten beruhende Auslösungen, werden abgewiesen.*

[11] Der Zeitstempelwert w¨are entbehrlich, wenn die Uhr sequentiell immer um 1 hochgez¨ahlt wird.

Visuelle Überraschungen

Zur Ausführung von Schreiboperationen benutzt man die dafür vorgesehene Taste, wodurch die Ausführung der Operation beginnt. In der Regel erwartet der Benutzer, daß die Auswirkung der Operation im eigenen Viewport nach kurzer Zeit dargestellt wird. Entsprechend erwartet man eine Fehlermeldung bei Eintritt von Fehlern. Kommt zwischen der Auslösung einer Operation und derer Visualisierung die Darstellung einer Operation eines anderen Benutzers, so ist man erstmal *überrascht*, da man normalerweise die Darstellung der eigenen Operation erwartet. Noch überraschender ist es, wenn die eigene Operation mit einer (späteren) Fehlermeldung erklärt wird, die durch die zuletzt dargestellte Operation des anderen Benutzers verursacht wurde.

Visuelle Überraschungen führen nicht zu Unkonsistenzen, können aber Verwirrungen hervorbringen. Um solche Verwirrungen zu verhindern, *werden Modifikationen anderer Teilnehmer zwischen der Auslösung einer Operation und deren Visualisierung nicht angezeigt*. Das wird realisiert, indem der Client Visualisierungsnachrichten in dieser Zeitspanne bei sich sammelt, bis die Visualisierung der eigenen Schreiboperation eintrifft.

Editieren von atomaren Attributen

Atomare Attribute bilden die kleinste Einheit in einer eNF^2-Relation. Das Editieren eines solchen Objekts kann daher die Parallelität der Gruppenarbeit in geringem Maße stören, auf der anderen Seite aber eine lange Zeit in Anspruch nehmen. Zum Editieren von atomaren Objekten wird daher *eine Schreibsperre gesetzt*, die Modifikationen anderer Benutzer an diesem Objekt während des Editiervorgangs verhindert.

Der Benutzer editiert eine *lokale Kopie* des atomaren Objekts. Am Ende des Editiervorgangs wird der Wert in die Datenbank geschrieben und die Sperre freigegeben. Der Benutzer kann aber seine Modifikationen explizit zwischenspeichern. Nach der Speicherung des neuen Werts in der Datenbank wird die Visualisierung an andere Benutzer propagiert.

4.4.4 Transaktionen

Zusätzlich zu der Möglichkeit der sequentiellen Ausführung von visuellen Operationen, können Transaktionen eine wichtige Rolle in der Gruppenarbeit spielen. Benutzer werden damit in die Lage versetzt, mehrere Operationen zu einer Transaktion zusammen zu fassen. Mögliche Szenerien dafür sind z. B.„buche Hotelzimmer nur, wenn Flug abgebucht werden kann", „kaufe Auto, wenn die Bank Kredit gibt" usw. Dabei verstehen wir unter dem Begriff Transaktion die logische Zusammenfassung mehrerer Operationen zu einer Einheit (Atomarität). In der Datenbankwelt wird dagegen

der Begriff einer Transaktion immer in Verbindung mit den sog. ACID-Eigenschaften gebracht [103, S. 392]. Die Bewahrung der Atomarität, Konsistenz sowie die Dauerhaftigkeit sind nach wievor unverzichtbare Eigenschaften einer Transaktion. Lediglich die Isolation wird hier nicht als Voraussetzung gestellt, sondern als zusätzliche Eigenschaft, nach der Transaktionen kategorisiert werden (isolierte, nicht isolierte Transaktionen).

Es gibt insgesamt drei Fälle, bei denen eine Folge von Operationen zu einer Transaktion zusammengefaßt wird:

Visualisierungsskripte: Eine Folge von Navigations- und Leseoperationen, die eine Visualisierung aus einer Datenbanktabelle erzeugt.

Visuelle Transaktionen (VTA): Ein Benutzer faßt eine Folge von visuellen Operationen zu einer Transaktion zusammen, um ein atomares Verhalten zu erreichen.

Isolierte Transaktionen: Der Benutzer führt eine visuelle Operation aus, die die Ausführung mehrerer Operationen auf die Datenbank erfordert.

Visuelle Transaktionen

Visuelle Transaktionen sind Folgen von Operationen, die vom Benutzer explizit zu einer Einheit zusammengefaßt werden. Sie befinden sich während ihrer gesamten Ausführung unter Kontrolle des ausführenden Benutzers und stehen unter Beobachtung der Gruppenmitglieder.

Definition 4.11 (Visuelle Transaktion)
Eine visuelle Transaktion ist eine Folge von visuellen Operationen mitsamt ihrer Visualisierungen bei allen Benutzern, die eine Teilmenge der von der Transaktion bearbeiteten Objekte sehen. Der ausführende Benutzer wird der **Besitzer** *der Transaktion genannt. Die Menge der Objekte, die von der Transaktion bearbeitet wird, wird* **Transaktionsmenge** *genannt. Benutzer, die eine Untermenge der Transaktionsmenge sehen, werden* **Beobachter** *der Transaktion genannt. Objekte und Benutzer, die in dem Viewport des Besitzers sichtbar sind, werden als* **Kontext** *der Transaktion bezeichnet.*

Visuelle Transaktionen werden während ihrer Ausführungen von anderen beobachtet. Auf der einen Seite beeinflußt der ausführende Benutzer die Beobachter der Transaktion und somit ihre Entscheidungen (z. B. Erhöhen eines Preisangebots in einer Versteigerung). Auf der anderen Seite hängen die eigenen Entscheidungen des ausführenden Benutzers von den Aktivitäten ab, die er in seinem Viewport beobachtet (z. B. Kauftransaktion abbrechen, weil ein günstigeres Angebot erscheint). Der Ablauf einer visuellen Transaktion ist somit nicht vorhersehbar (nicht vorprogrammiert), sondern wird während ihrer Ausführung vom ausführenden Benutzer entschieden [88].

Eine (theoretisch ausgedachte) serielle Ablauffolge von Transaktionen würde diesen gegenseitigen Einfluß nicht berücksichtigen können, da in diesem Fall die Transaktionen hintereinander, also isoliert, ablaufen würden. Visuelle Transaktionen laufen dagegen ohne Isolation ab. Das impliziert die Möglichkeit Objekte, die von anderen Transaktionen bearbeitet werden zu lesen. Demzufolge bedeutet dies der Verzicht auf der Serialisierbarkeit, da in einer seriellen Ablauffolge Transaktionen einzeln ablaufen und daher Objekte anderer Transaktionen nicht lesen können.

Im Viewport liegende Objekte des ausführenden Benutzers einer visuellen Transaktion, die nicht zur Transaktionsmenge gehören, dürfen weiterhin von anderen Teilnehmern modifiziert werden. Man vermutet zwar bei der Ausführung einer visuellen Transaktion, daß der Benutzer die Ausführung der Transaktion von dem Inhalt einiger Objekte in seinem Viewport abhängig machen könnte, also davon ausgehen könnte, daß sich die Inhalte dieser Objekte während der Ausführung der gesammten Transaktion nicht ändern werden. Das entspricht der Interpretation des *Sehens der Objekte* von der Seite des ausführenden Benutzers als Leseoperationen innerhalb seiner Transaktion und kann die Parallelität der Gruppenarbeit enorm stören. Stattdessen kann der Benutzer *pessimistisch vorgehen* und diese Objekte explizit sperren, d. h. sie zur Transaktionsmenge addieren (bewußtes Sehen), oder *optimistisch vorgehen*, indem er die Transaktion abbricht, falls sich Objekte in seinem Viewport entgegen seiner Erwartung ändern.

Der Abblauf einer visuellen Transaktion aus Benutzersicht beginnt mit einer *begin transaction*-Anweisung. Dannach können die Objekte der Transaktion bestimmt werden. Dazu positioniert der Benutzer seinen Finger auf ein Objekt und verlangt nach einer Sperre (Für den Benutzer sind das die Objekte, die er editieren will oder die von anderen Benutzern in der Zwischenzeit nicht verändert werden sollen). Gelingt es alle Sperren zu setzen, so kann die Ausführung von Operationen beginnen. Nachdem alle Operationen ausgeführt sind, beendet er die Transaktion mit einer *end transaction*-Anweisung.

Isolierte Transaktionen

Isolierte Transaktionen sind Transaktionen im klassischen Sinne, wie sie bei Datenbanksystemen zu finden sind [34, 84]. Führt ein Benutzer eine Operation aus, die weitere Lese- bzw. Schreiboperationen hervorbringt, so müssen diese innerhalb einer Transaktion ausgeführt werden. Die Auswirkungen der einzelnen Operationen sind irrelevant. Vielmehr interessiert man sich für das Gesamtergebnis der Transaktion. Die Visualisierung wird daher erst nach der Ausführung der gesamten Transaktion generiert.

Solche Transaktionen werden nicht vom Benutzer selbst ausgeführt, sondern von speziellen Anwendungen wie Mehrbenutzereditoren, Zeichenprogrammen, Projektmanagement etc. Operationen einer Anwendung für Projektmanagement könnten z. B. aus

mehreren einzelnen Datenbankoperationen zusammengesetzt sein. Die Zusammenfassung solcher Operationen zu einer Transaktion, was in der einfachsten Form der Anwender selbst vornimmt, übernimmt hier der Anwendungsentwickler .

Nehmen wir an, ein Benutzer möchte in einer Projektmanagement-Anwendung die Länge einer Task (Dauerzeit) erhöhen. Dies erreicht er in der Darstellung aus Abbildung 4.8, indem er mit der Maus die rechte Ecke des Balkens nach rechts zieht. Hier ist vorstellebar, daß die Anwendung in der lokalen Anzeige des Benutzers eine Art „*Absichtsanzeige*" erzeugt, z. B. indem sie die neue Position des Balkens über das vorhandene Bild blinken läßt und damit andeutet, daß die Operation noch nicht zu Ende ausgeführt wurde. Die Erhöhung der Dauerzeit kann aus mehreren Datenbankoperationen zusammengesetzt werden[12]. Die Anwendung würde also eine isolierte Transaktion beginnen, die notwendigen Objekte sperren, und die entsprechenden Operationen ausführen. Nach Beendigung der gesammten Transaktion würde die „Absichtsanzeige" gelöscht.

Wie der Name schon sagt, müssen diese Transaktionen im Gegensatz zu visuellen Transaktionen Isolation gewähren. Schreiboperationen sind daher bei diesen Transaktionen übliche `write`-Operationen und erfordern entsprechende Sperre auf das betroffene Objekt (X-Sperre). Da das Lesen eines Objekts nicht für die Visualisierung, sondern eventuell für Berechnungszwecke verwendet wird, muß sichergestellt werden, daß sich dieser Wert für die Dauer der Transaktion nicht ändert. Eine `read`-Operation erfordert daher das Setzen einer Lesesperre („*shared lock*", S-Sperre).

Synchronisierungsverfahren f¨ur Transaktionen

Es gilt nun, die verschiedenen Transaktionen untereinander zu synchronisieren. Dafür wird ein Sperrverfahren eingesetzt, daß durch die Einführung von neuen Sperrarten auf der einen Seite isolierten Transaktionen einen exklusiven Schreibzugriff gewährleistet, auf der anderen Seite visuellen Transaktionen einen Schreibzugriff ermöglicht, mit dem Erlaubnis des Lesens zum Visualisieren.

Sperren

Anders als bei DB-Transaktionen können Objekte, die von einer visuellen Transaktion geschrieben werden (Transaktionsmenge) von anderen Teilnehmern beobachtet werden. D. h., es muß erlaubt werden, daß Visualisierungsskripte die Werte dieser Objekte lesen, bevor die Transaktion mit *commit* oder *rollback* beendet wird. Für die Darstellung der Auswirkungen der einzelnen Schreiboperationen innerhalb einer visuellen Transaktion werden also Leseoperationen ausgeführt. Hier handelt es sich um eine Art „*Lesen, um zu visualisieren*", die nicht sicher stellen kann, daß ein gelesenes Objekt nach dem Versand an einen Client seinen Wert in der Datenbank schon geändert hat.

[12] In einer Projektmanagementanwendung w¨urde dies eine neue Sortierung aller Tasks in dem Projekt hervorbringen.

Definition 4.12 (`readV`: Lesen, um zu visualisieren)
Eine `readV`-Operation ist eine Leseoperation für den Zweck der Visualisierung.

Schreiben innerhalb einer visuellen Transaktion bedeutet „*Schreiben und Visualisieren*", d. h. Schreiboperationen sind immer mit einer unmittelbaren Visualisierung verbunden.

Definition 4.13 (`writeV`: Schreiben und Visualisieren)
Eine `writeV`-Operation ist die Zusammenfassung einer Schreiboperation und ihrer Visualisierung bei Benutzern, die Teile ihrer Auswirkungen sehen.

	`write`	`writeV`	`read`	`readV`
`write`	+	+	+	-
`writeV`	+	+	+	-
`read`	+	+	-	-
`readV`	-	-	-	-

Tabelle 4.3: Konflikte zwischen Operationen

Innerhalb einer visuellen Transaktion sind alle Schreiboperationen `writeV`-Operationen. Entsprechend sind alle Leseoperationen für die Visualisierung (innerhalb von Visualisierungsskripten) `readV`-Operationen. Neben den üblichen Konfliktsituationen - zwei Schreiboperationen, sowie eine Schreib- und eine Leseoperation auf dem selben Objekt - sind weitere Konflikte wegen der neu eingeführten Operationen zu beachten (Tabelle 4.3).

Zur Ausführung von `readV` und Navigationsoperationen ist keine Sperre erforderlich [151, S 160]. Zu den üblichen Lese-, Schreibsperren (S-, X-Sperren) wird daher eine weitere Sperre für die Ausführung einer `writeV`-Operation eingeführt.

Definition 4.14 (W-Sperre)
Eine W-Sperre ist eine Schreibsperre, bei der zusätzlich eine Visualisierung erfolgen muß.

Eine W-Sperre verhindert, daß zwei Schreiber gleichzeitig ein Objekt modifizieren, erlaubt aber das Lesen der Objekte zum Zweck der Visualisierung. Eine Leseoperation auf ein mit W-Sperre versehenes Objekt kann aber nicht sicherstellen, daß das Objekt für die Dauer einer Transaktion unverändert bleibt. Daher sind nur `readV`-Operationen auf solche Objekte erlaubt, nicht aber `read`-Operationen.

X- und W-Sperren sind unterschiedlich, obwohl in der Tabelle kein Unterschied zu sehen ist. Eine Schreiboperation auf ein mit W-Sperre versehenes Objekt ist eine `writeV`-Operation. Die Visualisierung ist somit ein Bestandteil der Operation. Weiterhin darf eine `readV`-Operation den Inhalt eines mit W-Sperre versehenem Objekts

	X	W	S
X	-	-	-
W	-	-	-
S	-	-	+

Tabelle 4.4: Verh¨altnisse zwischen den verschiedenen Sperren

lesen. Eine Schreiboperation auf einem mit X-Sperre versehenem Objekt ist dagegen eine übliche `write`-Operation. Daher dürfen Objekte, die mit einer X-Sperre versehen sind von keiner Leseoperation gelesen werden.

Erlangung der Sperren

In der Literatur existieren mehrere Verfahren für die Bestimmung der Reihenfolge in der Sperren erlangt werden, und für die Bestimmung der Granularität einer Sperre. Das *Zwei-Phasen Sperrprotokoll* (*Two-Phase Locking*, 2PL [67]) schreibt vor, daß eine Transaktion zunächst in einer Wachstumsphase alle Sperren anfordern muß, bevor in der Schrumpfphase die Freigabe der Sperren erfolgt [103]. Das 2PL-Protkoll garantiert die Serialisierbarkeit, verhindert aber Deadlocks nicht [170]. „Dirty read" kann auch eintreten, falls eine Transaktion abgebrochen wird. Das kann durch das sog. *strikte Zwei-Phasen-Sperrprotokoll* verhindert werden [103, S. 418].

Andere Sperr-Protokolle setzen eine Art partielle Ordnung der von der Menge der Transaktionen bearbeiteten Daten voraus. Diese Ordnung kann z. B. aus der logischen Struktur der Daten (hierarchische Datenbanken) abgeleitet werden. Das sog. *tree protocol* garantiert ebenfalls die Serialisierbarkeit und verhindert darüberhinaus Deadlocks [171, 170, 114]. Allerdings hat dieses Verfahren große Nachteile, da in manchen Fällen Objekte gesperrt werden, obwohl sie von der Transaktion nicht bearbeitet werden [170, S. 603].

Mit dem hier vorgestellten Verfahren wird versucht, das aus der Literatur bekannte 2PL-Protokoll zu verwenden [67]. Zur Verhinderung von Deadlocks werden Bedingungen aufgestellt, die vom *tree protocol* stammen [171]. Dabei wird die oben erwähnte partielle Ordnung aus der Baumstruktur einer eNF^2-Tabelle hergeleitet[13]. Für die Erlangung von Sperren innerhalb einer Transaktion (visuelle und isolierte Transaktionen) stellen wir die folgenden Regeln auf:

1. Jedes beliebige Objekt darf als erstes gesperrt werden.
2. Es kann keine Sperre auf ein Objekt mit einem gesperrten Nachfolger gesetzt werden.
3. Sperren, die innerhalb einer Transaktion freigegeben werden dürfen nicht wieder erlangt werden.

[13] Genauso kann diese Ordnung aus der Baumstruktur eines DOM-Baumes hergestellt werden.

Bei visuellen Transaktionen verlangen Benutzer die Sperren selbst, um Atomarität zu erzwingen. Um die Möglichkeiten eines Rollbacks zu minimieren müssen alle Sperren vor Beginn einer Transaktion erlangt werden. Mit der ersten Schreiboperation gelangt man in der Ausführungsphase, in der man keine weiteren Sperren erlangen kann. Mit Ende der Transaktion werden alle Sperren freigegeben. Sperren können aber vom Benutzer explizit vor Ende der Transaktion freigegeben werden, wenn sie nicht mehr gebraucht werden. Das kann z. B. der Fall sein, wenn andere Benutzer Interesse an das Objekt melden. Bei visuellen Transaktionen sehen die Benutzer, wenn eine Sperre nicht erlangt werden kann, von welchem Teilnehmer das Objekt gesperrt ist. In solchen Fällen werden Konflikte durch Verständigung der Benutzer unter einander gelöst.

Bei isolierten Transaktionen werden Sperren ebenfalls nach dem strikten 2-Phasen Sperr-Protokoll erlangt. Eine isolierte Transaktion blockiert, wenn eine Sperre nicht erlangt werden kann, behält aber die schon erlangten Sperren weiter. In dieser Zeit kann die Transaktion jeder Zeit vom Benutzer abgebrochen werden.

Sperrgranularität

Die Bestimmung der Granularität einer Sperre spielt eine wichtige Rolle bei der erstrebten Nebenläufigkeit. Für hierarchische Datenbanken existieren verschiedene Verfahren für die Bestimmung der Sperrgranularität (siehe z. B. das *tree protocol* [171], und das *queue protocol* [114]). Für die Bestimmung der Sperrgranularität bei dem vorliegenden eNF^2-Datenmodell stellen wir die folgenden Regeln auf (siehe Definition 4.8):

1. Einfügen/Löschen in einer Menge verlangt eine X-Sperre auf der Menge.
2. Beim Einfügen in einer Menge wird das neue Tupel mit einer Schreibsperre versehen.
3. Löschen eines Tupels aus einer Menge verlangt eine Schreibsperre auf das zu löschende Tupel.
4. Eine Sperre auf einem Tupel versieht alle seiner Söhne mit derselben Sperre. Ist einer der Söhne gesperrt, so kann keine Sperre auf das Tupel gewährt werden.
5. Zum Editieren eines atomaren Attributes muß dieses gesperrt werden.

Algorithmus

Das Verfahren für die Synchronisierung der verschiedenen Transaktionsarten untereinander wird durch die folgenden Punkte zusammengefaßt:

- Die verschiedenen Sperren sind gemäß Tabelle 4.4 miteinander verträglich.
- Schreiboperationen in einer visuellen Transaktion sind `writeV`-Operationen.
- Leseoperationen zur Visualisierung sind `readV`-Operationen.
- Schreiboperationen in einer isolierten Transaktion sind `write`-Operationen.
- Leseoperationen in einer isolierten Transaktion sind `read`-Operationen.

- Die Ausführung einer `readV`-Operation benötigt keine Sperre.
- Für die Ausführung der folgenden Operationen müssen die entsprechenden Sperren gesetzt werden:
  `write`: X-Sperre
  `read`: S-Sperre
  `writeV`: W-Sperre
- Eine `readV`-Operation auf einem mit X-Sperre versehenem Objekt ist undefiniert.
- Eine `readV`-Operation auf einem mit W-Sperre versehenem Objekt liefert seinen momentanen Wert zurück.
- Visualisierungsskripte setzen keine Sperren, da sie ausschließlich `readV` und Navigationsoperationen ausführen.

Programm 4.3 *Ausführung einer visuellen Transaktion*

1. **Sperr-Phase***: Benutzer fordert explizit die Sperren nacheinander. Das Fordern der ersten Sperre bedeutet den Beginn der Transaktion.*
   (a) *initialisiere Transaktion und die Logdatei.*
   (b) *sperre geforderte Objekte (W-Sperre). Falls eine Sperre nicht gesetzt werden kann, so kann der Benutzer entweder warten oder die Transaktion abbrechen.*
2. **Ausf¨uhrungs-Phase***: Die Ausführung der ersten Operation bedeutet den Übergang in diese Phase. Das Sperren weiterer Objekte ist nicht erlaubt.*
   (a) *solange kein COMMIT oder ROLLBACK*
      i. *wenn visuelle Operation eintrifft, dann*
         A. *Operation ausführen.*
         B. *Visualisierung der Operation an alle Beobachter senden.*
3. **End-Phase***: Benutzer benachrichtigt den Server über das Ende der Transaktion (führt end transaction aus).*
   (a) *warte auf die Beendigung der Visualisierung bei allen Clients.*
   (b) *für jede gesetzte Sperre :*
      i. *gib Sperre frei und lösche Eintrag aus der Logdatei.*
      ii. *benachrichtige Beobachter über die Freigabe der Sperre.*
   (c) *beende Transaktion.*
   (d) *benachrichtige ausführenden Client über die Beendigung der Transaktion.*

Konsistenzstufen

Für die Ausführung von Schreiboperationen wurden bei visuellen Transaktionen W-Sperren eingesetzt, wodurch Modifikationen anderer Teilnehmer für die gesamte Dauer der Transaktion ausgeschlossen wurde. Für Leseoperationen innerhalb von Visualisierungsskripten war keine Sperre notwendig. Die Erlangung der Sperren basierte auf dem 2PL-Protokoll. Für eine visuelle Transaktion können also die folgenden Zusicherungen erreicht werden:

- Schreiboperationen gelten für die gesamte Dauer der Transaktion.
- Visuelle Operationen innerhalb der Transaktion sind beabsichtigt.
- Modifikationen an sichtbaren Objekten sind nach kurzer Zeit sichtbar.

Das obige Verfahren garantiert also für visuelle Transaktionen die sog. Konsistenzstufe 2 [85, 84] (verhindert write/write Konflikte, read/write-Konflikte sind erlaubt), mit der zusätzlichen Sicherung, daß jede Operation innerhalb der Transaktion aus einer aktuellen Visualisierung entsteht (beabsichtigt). Diese Zusicherung reicht in der Praxis aus, da dirty read den Teilnehmern bekannt ist. Die Benutzer beobachten den Ablauf anderer visuellen Transaktionen, sodaß ein dirty read nur für ganz kurze Zeitabstände eintritt. Verlaufen Transaktionen anderer Teilnehmer entgegen den eigenen Erwartungen, so wurde ein Benutzer seine Transaktion abbrechen und sie rückgängig machen.

Innerhalb einer isolierten Transaktionen wurde für die Ausführung einer Schreiboperation eine X-Sperre gesetzt, und eine S-Sperre für Leseoperationen. Dadurch wurde innerhlab einer isolierten Transaktion die folgenden Zusicherungen erreicht:

- Schreiboperationen gelten für die gesamte Dauer der Transaktion.
- Leseoperationen gelten für die gesamte Dauer der Transaktion.

Für isolierte Transaktionen garantiert das obige Verfahren also die sog. Konsistenzstufe 3 [85, 84]. D. h. es werden sowohl write/write Konflikte als auch read/write Konflikte verhindert. Damit werden alle 3 bekannten Anomalien innerhalb einer isolierten Transaktion verhindert. Die Serialisierbarkeit von isolierten Transaktionen unter einander wird durch das obige Verfahren garantiert (Siehe „*Fundamentalsatz des Sperrens*" [103, S. 417][67]).

Deadlocks werden dadurch verhindert, daß für die Erlangung der Sperren, ähnlich dem *tree protocol*, eine partielle Ordnung durch die Baumstruktur einer eNF^2-Relation gegeben wird. Durch Regel 2 zur Erlangung der Sperren wird versichert, daß Objekte in einer partiellen Ordnung gesperrt werden, und damit einen Deadlock verhindert [171, 170].

4.5 Der Groupware-Server aus Benutzersicht

Wie aus der Schichtenarchitektur zu erkennen ist, stellt der VClient aus Benutzersicht die Schnittstelle des Groupware-Servers zum Client dar. Das Protokoll, das diese Kommunikation beschreibt wird durch zwei Schnittstellen festgelegt. Die „*Schnittstelle zwischen dem VClient und dem Client*" beschreibt die Befehle, die eine Anwendung an den Groupware-Server verschicken kann, um Dokumente zu lesen, Finger zu erzeugen, zu navigieren und Schreiboperationen auszuführen. Die „*Schnittstelle zwischen dem Client und dem VClient*" beschreibt den Aufbau der Dokumente, die der Groupware-Server an eine Anwendung verschickt. Diese beschreiben einfache Antworten, den Inhalt vom Viewport, Fehlermeldungen, Warnungen oder Benachrichtigungen. Aus Benutzersicht beschreibt dieses Protokoll die Schnittstelle einer Anwendung zum Groupware-Server[14].

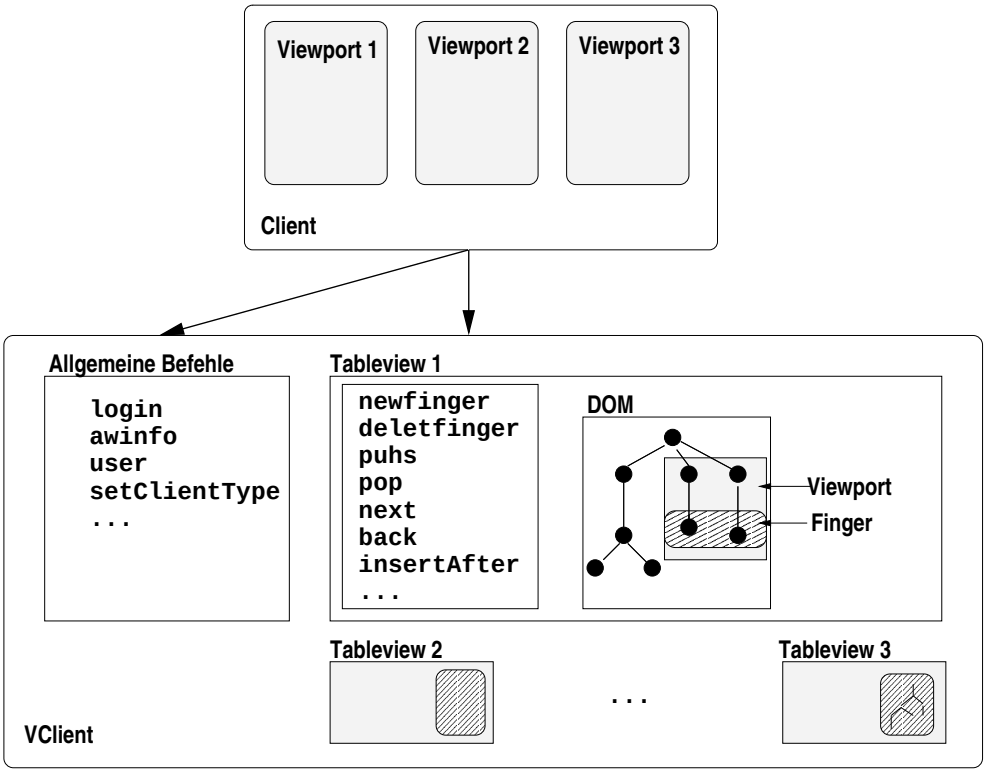


Abbildung 4.17: Der Groupware-Server aus Benutzersicht

Die Kommunikation zwischen einem Client und dem Server basiert auf einer einfachen Socket-Verbindung. Die Verbindung zu dem Server wird durch seinen Namen und einen festen Port aufgebaut. Nachdem die Verbindung hergestellt wird, muß sich ein

[14] Im Rest dieses Kapitels verwende ich das Wort Server statt Groupware-Server.

Benutzer mit einem Passwort authentifizieren, anschließend kann eine Anwendung Befehle an dem Server senden.

`login` *`name password`* : Anmelden. Die Anmeldedaten werden an den Datenserver weitergeleitet.
`application list` : Liefert eine Liste der vorhandenen Applikationen.
`application select` *`application`* : Macht *`application`* zur aktiven Applikation.
`table (list | open | close | save)` : Angepaßtes `table` Kommando. Bietet dieselben Funktionen wie der TclDB-Kommando `table` . (Öffnen, Schließen und Speichern von Tabellen). Das Öffnen einer Tabelle erzeugt eine Instanz der Klasse `Tableview`, die die Ausf¨uhrung aller weiteren Operationen auf die Tabelle erm¨oglicht.
`tableview` `newfinger	deleteFinger` : Erzeugen (L¨oschen) eines neuen Fingers auf der Tabelle.
`tableview` `push	pop	next	back	up	down` : F¨uhrt entsprechende Navigationsoperationen auf der Tabelle mit dem aktiven Finger des Benutzers aus.
`tableview` `insertBefore	insertAfter	delete	deleteBefore` : F¨uhrt die entsprechenden Editieroperationen auf der Tabelle mit dem aktiven Finger des Benutzers aus.
`tableview` `draw` *`height`* : Schickt den Inhalt eines Dokuments an dem Client weiter. Es werden nur die entsprechenden Teile des Dokuments geschickt, die in den Viewport des Benutzers passen.
`lock (info | set | free | islocked | who | rid)` : Behandlung von Sperren. Eine Anwendung kann explizit Sperren setzen und freigeben. F¨ur die Identifizierung eines Objekts wird seine Objekt Id, und die Id des Dokuments ben¨otigt.
`transaction (begin | end | lock | abort)` : Behandlung von visuellen Transaktionen. Mit `begin` wird eine TA eingeleitet. Mit `lock` k¨onnen innerhalb der TA Objekte gesperrt werden. Mit `end` kann die TA abgeschlossen werden.
`user info` *`user`* : Liefert ein XML-Dokument mit detaillierten Informationen ¨uber den Benutzer `user`. Die Struktur des XML-Dokuments ist festgeschrieben.
`awinfo` *`op`* : Liefert Awareness-Informationen entsprechend dem Unterbefehl *`op`*.
`setClientType` *`op`* : Legt den Typ eines Clients fest. Zul¨assige Typen sind momentan `pc`, `applet` und `mobile`.

Tabelle 4.5: Die Schnittstelle des VClients zum Client in abgek¨urzter Form. *`tableview`* bezeichnet eine Instanz der Klasse `Tableview` zu einer bestimmten Tabelle.

Der Informationsraum ist grundsätzlich aus sog. Applikationen aufgebaut. Jede Applikation enthält logisch zusammenhängende Dokumente. Der Befehl `application` dient zur Auflistung und Selektion von Applikationen. Innerhalb einer Applikation

können Dokumente mit dem Befehl `table list` aufgelistet werden. Mit dem Befehl `table open` kann ein Dokument geöffnet werden. Für jedes geöffnete XML-Dokument wird eine Instanz der Klasse `Tableview` erzeugt, die Navigationsoperationen in Abhängigkeit zur Position des aktiven Fingers ausführt.

Finger-Navigationsoperationen werden wie in Abschnitt 4.1.1 beschrieben ausgeführt. Die Ausführung einer solchen Operation bewirkt eine Neupositionierung des aktiven Fingers. Diese kann allerdings mehrere Auswirkungen auf die Anzeige haben. Scrollt man z. B. zu einer Stelle außerhalb des sichtbaren Bereichs, so müssen neue Ausschnitte des Dokuments gelesen werden; möglicherweise ändern sich dadurch einige Awareness-Informationen, z. B. sehen neue Benutzer den eigenen Finger oder umgekehrt. Nach jeder Navigationsoperation werden all diese Möglichkeiten im Server behandelt und der aktuelle Zustand des Viewports in Form eines XML-Dokuments an den Client geschickt. Ändert sich nur die Position des Fingers, so wird lediglich diese Veränderung der Anzeige ermittelt.

Zur Ausführung von Schreiboperationen dient auch dieselbe Klasse `Tableview`. Fingerschreiboperationen werden wie in Abschnitt 4.1.1 ausgeführt. Nach der Ausführung von Schreiboperationen wird der Inhalt eines Viewports dem schreibenden Client und allen Benutzern, die seinen Viewport sehen, neu verschickt. Die Synchronisierung der Fingeroperationen bleibt in solchen Fällen für den Benutzer unsichtbar. Das Editieren von Inhalten atomarer Objekte muß durch eine Sperre geschützt werden. Eine Anwendung kann eine solche Sperre explizit mit dem Befehl `lock` (mit dem entsprechenden Unterbefehl/Option) setzen.

Für die Ausführung von Transaktionen dient der Befehl `transaction` mit den Unterbefehlen `begin`, `end`, `abort` und `lock`. Die Befehle `user` und `awinfo` dienen zum expliziten Abfragen von Awareness-Informationen. Tabelle 4.5 enthält eine Zusammenfassung der Befehele der VClient-Schnittstelle.

Der VClient verschickt seine Daten an den Client in Form von XML-Dokumenten, die nach einem einfachen Protokoll aufgebaut sind. Dannach besteht jede Nachricht aus zwei XML-Dokumenten, einem *Header* und einem *Body*. Die Nachricht selbst wird in kleine Abschnitte (4 KB) unterteilt, deren erste zwei Bytes die Größe angeben. Der Header selbst ist der erste Teil einer Nachricht. Das Ende einer Nachricht wird mit einem Abschnitt der Größe 0 abgeschlossen. Durch die Bereitstellung von entsprechenden Methoden in der Client-Schnittstelle (siehe Abschnitt 5.3) wird es möglich, daß verschiedene Nachrichten getrennt voneinander bearbeitet werden. Dadurch kann ein XML-Parser direkt aus der Socket wie aus einer üblichen Datei lesen.

Es gibt sechs verschiedene Arten von Nachrichten:

View: Aktualisiert die Anzeige in einer Tabelle, nachdem eine Operation auf der Tabelle die Anzeige im Viewport des Benutzers verändert hat. View-Nachrichten bestehen aus einem XML-Dokument, das den ganzen Viewport des Benutzers beschreibt und damit den alten ersetzt. Die Datenbankobjekte werden nach dem angegebenen Schema für eNF^2-Tabellen als ein Unterelement dargestellt.

Das folgende Beispiel zeigt eine View-Nachricht. Der Header besteht aus einer eindeutigen `messageid`, `type` (Typ der Nachricht) und `id` (eindeutige Identifizierung des Objekts, an das die Nachricht gerichtet ist). Die Angabe in `objekt` dient einer weiteren Qualifizierung. Es gibt Nachrichten, die an ein eindeutiges Objekt gerichtet sind und solche, die für die gesamte Anwendung von Interesse sind (`objekt = system`). Der zweite Teil der Nachricht enthält die Datenbankobjekte selbst und eine Liste der Benutzer bestimmt nach dem Awareness-Mode.

Programm 4.4 *Eine View-Nachricht*

```
[länge]<?xml version='1.0'  encoding="ISO-8859-1" standalone="yes"?>
<head>
<messageid>12</messageid>
<type>view</type>
<object>table</object>
<id>vtbl1</id>
</head>[länge]
<?xml version='1.0' encoding="ISO-8859-1" standalone="no" ?>
<?xml:stylesheet type="text/xsl" href="viewport.xsl" ?>
<view name="Software">
    <!-- Teil der Tabelle, der im viewport sichtbar ist. -->
    <ETable>
      <TASKS nf2type="sett" id="0x5" bg="none">
       <TASKS id="0x6" nf2type="ptuplet" bg="none">
          <TASKID id="0x7" nf2type="textt" bg="none">START</TASKID>
          <DESCRIPTION id="0x8" nf2type="textt" bg="none">
               project kickoff
          </DESCRIPTION>
          <DUR id="0x9" nf2type="intt" bg="none">0</DUR>
          <ES id="0xa" nf2type="intt" bg="none">0</ES>
          <LS id="0xb" nf2type="intt" bg="none">0</LS>
          <EF id="0xc" nf2type="intt" bg="none">0</EF>
          <LF id="0xd" nf2type="intt" bg="none">0</LF>
          <ISOTIME id="0xe" nf2type="textt" bg="none">
               01-01-1998
          </ISOTIME>
          <REQUIRES id="0xf" nf2type="sett" bg="none" state="empty"/>
      </TASKS>
      ...
     </TASKS>
   </ETable>
  <awareness>
    <users>
      <user>
        <name>morad</name>
        <host>ibiza.db.informatik.uni-kassel.de</host>
      </user>
    </users>
  </awareness>
</view>[länge = 0: Ende der Nachricht][nächste Nachricht]...
```

Awareness: Bezieht sich auf die Veränderung der Awareness-Anzeige. Dies ist z. B. wenn ein neuer Benutzer auf die Tabelle zugreift, oder seinen Finger in dem Viewport des Benutzers positioniert hat. Der Benutzer kann dazu zwischen drei Awareness-Modi wählen mit denen er bestimmen kann, welche Benutzer er in seiner Awareness-Anzeige sehen will:

Table: Benutzer, die auf die Tabelle zugreifen.
Fokus: Benutzer, die seinen Fokus sehen.
Nimbus: Benutzer, die seinen Nimbus sehen.

Das XML-Dokument enthält eine Liste von den Benutzern, die nach dem Awareness-Modus in der Anzeige aufgelistet werden.

Programm 4.5 *Beispiel einer Awareness-Nachricht*

```
<?xml version='1.0'  encoding="ISO-8859-1" standalone="yes"?>
<head>
<messageid>7</messageid>
<type>awareness</type>
<object>system</object>
<id>0</id>
</head>
<?xml version='1.0' encoding="ISO-8859-1" standalone="yes" ?>
<awareness>
   <users>
      <user id="2">
         <name>morad</name>
         <host>ibiza</host>
      </user>
      <user id="3">
         <name>wegner</name>
         <host>elba</host>
      </user>
   </users>
</awareness>
```

Merge: Eine Merge-Nachricht bezieht sich nur auf Navigationsoperationen. Bei der Änderung der Position eines Fingers ändert sich in der Anzeige nur das Attribut `bg`. Es wird nicht wie bei View-Nachrichten das ganze XML-Dokument zum Client übertragen, sondern nur die Veränderung, die die neue Position des Fingers darstellt.

Das folgende Beispiel zeigt eine Merge-Nachricht. Hier wird nur die Veränderung des Attributes `bg` für das Objekt mit dem `id 0x6` auf den Wert `#FF6611` gesetzt. Bevor die Veränderungen eingetragen werden, werden alle `bg` Attribute auf `none` gesetzt[15].

Programm 4.6 *Eine Merge-Nachricht*

```
[...]<?xml version='1.0' encoding="ISO-8859-1" standalone="yes"?>
<head>
```

[15] `bg = none` bedeutet f¨ur ein Element, daß es keine Hintergrundfarbe besitzt.

```
   <messageid>19</messageid>
   <type>merge</type>
   <object>table</object>
   <id>vtbl3</id>
</head>[...]
<?xml version='1.0' encoding="ISO-8859-1" standalone="yes" ?>
<changes attr='bg'>
   <change id='0x6'>
      <newvalue>#FF6611</newvalue>
   </change>
</changes>[0][...]
```

Responde: Einige Operationen sind Abfragen, die einen Wert zurückliefern, wie z. B. `awinfo user list`, um eine Liste aller momentanen Benutzer zu bekommen. Hier wird ein XML-Dokument geschickt, das die Rückgabe der Operation enthält. Die Client-Schnittstelle kann aus dem XML-Dokument diese Daten herauslesen und sie an die Anwendung weitergeben.

Programm 4.7 *Beispiel einer Responde-Nachricht*

```
<?xml version='1.0'  encoding="ISO-8859-1" standalone="yes"?>
<head>
<messageid>3</messageid>
<type>responde</type>
<object>system</object>
<id></id>
</head>
<?xml version='1.0' encoding="ISO-8859-1" standalone="yes"?>
<body>
   <responde>.system demo Projekts tests books users lectures</responde>
</body>
```

Unsuccessop: Benachrichtigt den Client, daß eine Operation nicht ausgeführt werden kann. Dies ist z. B. der Fall, wenn der Benutzer eine `pop`-Operation ausführt, wobei der Finger auf der Wurzel der Tabelle steht.

Programm 4.8 *Beispiel einer Unsuccessop-Nachricht*

```
<?xmlversion='1.0'  encoding="ISO-8859-1" standalone="yes"?>
<head>
<messageid>12</messageid>
<type>unsuccessop</type>
<object>table</object>
<id>vtbl11</id>
</head>
<?xml version='1.0' encoding="ISO-8859-1" standalone="yes"?>
<body>
   <unsuccessop>
      <code>6</code>
      <message>operation unsuccessful</message>
   </unsuccessop>
</body>
```

Error: Fehler werden gemeldet, falls syntaktisch nicht korrekte Operationen geschickt werden, oder serverinterne Fehler entstehen. Fehlermeldungen sind im Protokoll mit eindeutigen Fehler-Codes versehen und werden von der Client-Schnittstelle behandelt.

Programm 4.9 *Beispiel einer Error-Nachricht*

```
<?xml version='1.0'  encoding="ISO-8859-1" standalone="yes"?>
<head>
<messageid>27</messageid>
<type>error</type>
<object>none</object>
<id>0</id>
</head>
<?xml version='1.0' encoding="ISO-8859-1" standalone="yes" ?>
<error>
   <message>
      invalid command name "tb12.f13"
   </message>
   <code>0</code>
</error>
```

4.6 Zusammenfassung

In diesem Kapitel wurde die Systemarchitektur vorgestellt. Nachdem das Interaktionsmodell und das Fingerparadigma vorgestellt wurden, wurden Lösungen für die Visualisierung, Awareness und die Nebenläufigkeitskontrolle vorgestellt.

Die Visualisierung basiert auf der Darstellung von eNF^2-Tabellen durch eine XML-Sprache, die die eNF^2-Struktur auch im XML-Dokument beibehält und auf der Bereitstellung von generischen Stylesheets für die Darstellung solcher Dokumente. Weiterhin wurde die Möglichkeit offen gelassen, daß Anwendungsentwickler ihre eigenen, generischen oder speziellen Stylesheets entwickeln.

Durch das Fingerparadigma kann die örtliche Präsenz eines Benutzers einfach bestimmt werden, woraus Awareness-Nachrichten erzeugt werden können. Der Viewport gibt den Interessenbereich eines Benutzers an, während sein aktiver Finger seinen Aktivitätsbereich angibt. Die Verwandtschaft dieses Modells mit dem Fokus/Nimbus-Modell wurde hervorgehoben.

Für die Lösung der Nebenläufigkeit wurden Fingeroperationen qualifiziert und Konflikte zwischen ihnen bestimmt. Dabei wurde das Sehen von Objekten durch eine sogenannte V-Sperre vermerkt und zwei neue Operationen eingeführt: `writeV`und `readV`. Weiterhin wurden verschiedene Möglichkeiten der Ausführung von Operationen auf dem gemeinsamen Informationsraum qualifiziert und in verschiedene Klassen unterteilt. Wichtig war dabei, zwischen Operationen/Transaktionen zu unterscheiden, die unmittelbar nach ihrer Ausführung visualisiert werden müssen und solchen, die

erst nach der Beendigung der gesamten Transaktion, zu der sie gehören, visualisiert werden.

Für die Bearbeitung des gemeinsamen Informationsraumes gibt es zwei mögliche Vorgehensweisen, die von den Benutzern miteinander kombiniert werden können. Auf der einen Seite können Benutzer einzelne visuelle Operationen hintereinander ausführen und sich auf diese Weise mit Hilfe der zusätzlichen Awareness-Informationen gegenseitig koordinieren. Bei visuellen Operationen ist es notwendig sicher stellen zu können, daß sie aus einem aktuellen visuellen Zustand ausgelöst werden, damit sichergestellt werden kann, daß ein Benutzer mit voller Absicht eine bestimmte Operation ausführt. Diese Sicherstellung konnte mit einem Zeitstempelverfahren erreicht werden.

Auf der anderen Seite können für bestimmte Aufgaben Transaktionen gestartet werden, die während ihrer Ausführung unter Kontrolle des ausführenden Benutzers bleiben und unter Beobachtung der Gruppenmitglieder stehen (visuelle Transaktionen). Transaktionen, die bestimmte schnelle Operationen ausführen, laufen dagegen im Hintergrund und stellen erst nach ihrer Beendigung ihr Endergebnis den Beobachtern zur Verfügung (isolierte Transaktionen). Die Unterschiede zwischen diesen Transaktionen wurden dargestellt. Die Synchronisierung von verschiedenen, parallel laufenden Transaktionen untereinander wurde durch ein sperrbasiertes Verfahren erreicht.

Kapitel 5

Implementierung

Nachdem die Systemarchitektur in Kapitel 4 beschrieben wurde, wird in diesem Kapitel die Implementierung des Prototyps vorgestellt. Da diese ein zentraler Bestandteil dieser Arbeit ist, werden verschiedene Bestandteile des gesamten Systems im Detail behandelt. Insbesondere werden die Entwicklung von Exportmechanismen von eNF^2-Tabellen in XML, die Entwicklung von generischen Stylesheets für die Visualisierung von XML-Dokumenten und Schemata sowie die Implementierung des Fokus/Nimbus-Modells behandelt.

Die Realisierung der Dreischichtenarchitektur wird zuerst vorgestellt; sowohl die Implementierung von jeder Schicht sowie die Schnittstellen zwischen den verschiedenen Schichten werden besprochen. Die Implementierung einer Java-Anwendung und die Entwicklung von Spezialanwendungen mit Hilfe der Clientschnittstelle werden am Ende dieses Kapitels diskutiert[1].

5.1 Der Datenserver

Der Server hat einen uneingeschränkten Zugriff auf die Datenbank und regelt den Zugriff durch eine Hülle um TclDB, die Mechanismen für die Nebenläufigkeitskontrolle und Zugriffskontrolle implementiert. Die Schnittstelle zum VClient baut auf Funktionen der vier Hauptmodule des Datenservers auf (siehe Abb. 4.4):

- Datenbankmodul (`db`)
- Nebenläufigkeitskontrolle (`lock`)
- Zugriffs- und Benutzerverwaltung (`users`)
- Awareness (`awareness`)

[1] L¨angere Codesegmente werden im Anhang aufgelistet.

Datenbankmodul

Für die Ausführung von Operationen auf den gemeinsamen Informationsraum (eNF^2-Datenbank) werden die Standard TclDB-Kommandos neu implementiert. Sie werden um die folgenden vier Funktionalitäten erweitert:

- Finger haben Besitzer, der als einziger Operationen mit ihnen ausführen darf[2].
- Benutzerausführungsrechte werden überprüft.
- Überprüfen, ob richtige Sperre für die Ausführung der Operation gesetzt ist.
- Awareness-Informationen werden aus den Datenbankoperationen generiert. Benachrichtigungen werden an die interessierten Benutzer nach Ausführung von Operationen gesendet.

Weiterhin werden im Datenserver zwei Arten von Fingern unterschieden: *Skript-Finger* werden vom Datenserver zum Ausführen von Operationen in der Datenbank durch einen Skript benutzt und werden in der Tabellenanzeige nicht angezeigt. Für die Generierung von Awareness-Informationen sind sie daher uninteressant. *Visuelle Finger* werden vom Benutzer für die Navigation auf einer Tabelle erzeugt und in der Benutzeranzeige dargestellt.

Erzeugen von Fingern

Nachdem ein Finger erzeugt wurde, werden die Client-Daten in dem Tcl-assoziativen Array `clientInfo` aktualisiert. Der Finger wird in die Fingerliste des Benutzers eingetragen. Der Besitzer eines Fingers wird ebenfalls in `fingerInfo($f,owner)` eingetragen. Für die Ausführung von Fingeroperationen beim Datenserver dient die Funktion `fingerObjCmd`. Als Argumente bekommt sie den `fid` und den Besitzer des Fingers, sowie die Fingeroperation selbst. Beim VClient wird eine Prozedur mit demselben Namen wie der `fid` erzeugt, deren Aufruf die Ausführung von `fingerObjCmd` mit dem Argument `fid` bei dem Server bewirkt. Fingeroperationen werden auf diese Weise bei dem VClient wie in TclDB aufgerufen.

Programm 5.1 *`finger_fork`: Erzeugen von Fingern beim Server*

```
proc finger_fork {fid} {
    global clientInfo  cid2id dp_rpcFile
    set client  $cid2id($dp_rpcFile)
    set f [uplevel #0 tcldb::finger fork $fid]
    set tid [tcldb::$f tid]
    if {![has_focus $client $tid] || \
            [lsearch $clientInfo($client,tables) $tid] < 0 } {
        tcldb::finger free $f
```

[2] Finger, die der Server für die Ausführung von eigenen Funktionen erzeugt, haben den Besitzer 0 (Der Server).

```
        error "table is not open, or focus is not defined"
    }
    init_finger $f $client "visuall"
    incr clientInfo($client,fingerCount)  1
    lappend clientInfo($client,fingers)   $f
    if {$clientInfo($client,$tid,fingerCount) == 0} {
        set clientInfo($client,$tid,fingerCount)  1
        set clientInfo($client,$tid,fingers) $f
     } else {
        incr clientInfo($client,$tid,fingerCount) 1
        lappend clientInfo($client,$tid,fingers) $f
    }
    # Der Finger soll bei einem VClient "normal" aufgerufen werden.
    set script "proc $f {args} {
    global ESERVER
    return \[uplevel #0 dp_RPC  \$ESERVER fingerObjCmd \$f \$args\]\}}"
     dp_RDO $dp_rpcFile uplevel #0 $script
    return $f
}

proc init_finger {f owner mode} {
    global fingerInfo
     if {[lsearch "work visuall" $mode] < 0} {
         error "bad finger mode \"$mode\"should be work or visuall"
     }
    set fingerInfo($f,owner)    $owner
    set fingerInfo($f,mode)     $mode
    set fingerInfo($f,vfinger)  "none"; # bis der Client einen registriert
    set fingerInfo($f,time)     [clock seconds]
}
```

Ausf¨uhren von Fingeroperationen

Die Methode `fingerObjCmd` führt Fingeroperationen bei dem Server aus. Diese werden in Navigations-, Schreib- und Leseoperationen aufgeteilt:

Programm 5.2 *`fingerObjCmd`: Behandeln von Fingeroperationen*

```
proc fingerObjCmd {f args} {
    global dp_rpcFile clientInfo cid2id
    set user $dp_rpcFile
    if ![isregistered $user] {
        error "can't execute command, client isn't registred!"
    }
    if ![fingerBelongsTo $f $user] {
        error "$user doesn't own finger!"
    }
    set readCS {get type iscomplex isatomic istype isnull \
                isempty card on attr info rid link\
                tid table application schema }
```

```
    set writeCS   {set insert delete swap}
    set moveCS    {push pop top go}
    set cmd [lindex $args 0]
    if {[lsearch $writeCS $cmd] >= 0} {
        return [uplevel finger_write $f $args]
    } elseif {[lsearch $readCS $cmd] >= 0} {
        return [uplevel finger_read $f $args]
    } elseif {[lsearch $moveCS $cmd] >= 0} {
        return [uplevel finger_navigate $f $args]
    } else {
        error "bad option \"$cmd\" should be fingerObjCmd fid \n \
              $readCS\n $writeCS \n $moveCS"
    }
}
```

Für jede dieser Befehlsklassen gibt es eine Methode, die entsprechende Rechte vor der Ausführung prüft und die entsprechenden Aktionen nach der Ausführung in Gange setzt. Als Beispiel dient die Methode `finger_write` zur Ausführung von Schreiboperationen (Programm C.2). Dazu werden erst die Ausführungsrechte kontrolliert:

Programm 5.3 *`finger_write`: kontrollieren von Ausführungsrechte*

```
proc finger_write {tf args} {
  # Objekt ist nicht gesperrt! Kann daher nicht modifiziert werden.
  if {[set lRid [Xlock_rid $tid $rid]] == ""} {
      error "object not writable, no lock seted on objekt!"
  }
  # Überprüfen der Sperrgranularität.
  if {[check_granularity $tf $lRid $cmd] == 0} {
      error "parent attribute must be locked"
  }
  # Benutzer ist nicht der Besitzer der Sperre.
  if {$client != $XLInfo($table,$lRid,client)} {
      puts "$client != $XLInfo($table,$lRid,client)"
      error "The specified action isn't allowed"
  }
  # Ok! Operation ausführen ...
}
```

Nach der Ausführung einer Operation wird die Methode `table_notify` aufgerufen, um Auswirkungen der Operation erst zu registrieren und dann an entsprechende Benutzer weiterzuleiten. Dabei benachrichtigt `table_notify` Benutzer, die den Ort der Operation sehen. In dem Beispiel wird ein `delete` behandelt:

Programm 5.4 *`finger_write`: Ausführen der Operation und benachrichtigen der Benutzer*

```
proc finger_write {tf args} {
  ...
  uplevel tcldb::$tf $args;# Ausführen der Operation.
```

```
    set rid2 [tcldb::$tf rid]
    switch -- $cmd {
        "delete" {
            set last [tcldb::$tf on -last]
            table_notify $tid $client \
                    delete o_$tf $rid $rid2 $last
        }
    ...
  }
```

Sperrmodul

Eine Sperre wird auf einer Tabelle (gegeben durch ihre *tid*) und den *rid* des Objekts gesetzt. Alle Unterobjekte werden mit derselben Sperre versehen. Für den VClient können die Sperrmechanismen mit der Funktion `lock` mit entsprechenden Unterbefehlen und Optionen aufgerufen werden. Für jede Art von Sperre gibt es die entsprechenden Funktionen zum Setzen und Freigeben sowie zum Abfragen von Informationen über die Sperre und deren Besitzer.

Informationen über eine Sperre werden in einem Tcl-assoziativen Array gespeichert. Für exklusive Schreibsperren z. B. enthält `XLInfo($table,$rid,$client,-infovar)` die entsprechende Information über die Schreibsperre, die der Client `client` auf das Objekt `rid` besitzt. In einem assoziativen Array, z. B. `XLTable($table)` bei Schreibsperren, werden alle Objekte der Tabelle eingetragen, die mit einer visuellen bzw. Schreibsperre versehen sind. Das folgende Beispiel zeigt die interne Funktion zum Setzen einer Schreibsperre:

Programm 5.5 *`Xlock_set` Setzen von Schreibsperren*

```
proc Xlock_set {tid rid client} {
    global clientInfo tableInfo
    global XLTable XLInfo
    set table $tableInfo($tid,catalog)
    # Objekt ist gesperrt
    if {[lsearch $XLTable($table) $rid] >= 0} {
        return 0
    }
    if {[lsearch $RLTable($table) $rid] >= 0} {
        return 0
    }
    # Sperre kann gesetzt werden
    # Nur ein Client kann die Schreibsperre besitzen
    lappend XLTable($table) $rid
    set XLInfo($table,$rid,stime) \
            [set XLInfo($table,$rid,etime)  [clock seconds]]
    set XLInfo($table,$rid,client) $client
    lappend clientInfo($client,wlocks) "$table $rid"
    lappend clientInfo($client,$table,wlocks) "$rid"
```

```
    incr clientInfo($client,$table,wlockC)  1
    incr clientInfo($client,wlockcount)  1
    ntf_newlock $client $tid $rid
    return 1
}
```

Awareness-Modul

Das Awareness-Modul stellt die Basis für die Bereitstellung von Group-Structural-Awareness und Workspace-Awareness. Für die Bereitstellung von Group-Structural-Awareness müssen einige Informationen vom Benutzer bzw. die entsprechende VClient-Instanz registriert werden. Für die Registrierung von Benutzerinformationen implementiert das Awareness-Modul die Funktion `register`.

Programm 5.6 *`register` kennt vier Unterbefehle `vtable` (`vfinger`): Virtuelle Tabelle (Finger) bei dem VClient, `activefinger`: Der aktive Finger eines Benutzers, `newclient`: Informationen über einen neuen Benutzer*

```
proc register {what args} {
    global  dp_rpcFile
    if {($what != "newclient") && (![isregistered $dp_rpcFile])} {
        error "can't execute command, client isn't registred!"
    }
    global cid2id clientInfo clients fingerInfo tableInfo
    set length [llength $args]
    switch -- $what {
        "vtable" {...}
        "vfinger" {...}
        "activefinger" {...}
        "newclient" {
            if {$length != 3} {
                error "wrong # args : should be register
                                      newclient user password host"
            }
            set other $clients;# client ist noch nicht in der Liste
            set id [register_client $dp_rpcFile [lindex $args 0] \
                                    [lindex $args 1] [lindex $args 2]]
            set newC $clientInfo($id,cid)
            # Client selbst
            eval dp_RPC $newC add_client $id \
                    $clientInfo($id,user) $clientInfo($id,host)\
                    $clientInfo($id,addr)
            foreach clt $other {
                set _id $cid2id($clt)
                eval dp_RPC $clt add_client $id $clientInfo($id,user)
                          $clientInfo($id,host) $clientInfo($id,addr)
                eval dp_RPC $newC add_client $_id $clientInfo($_id,user)
                        $clientInfo($_id,host) $clientInfo($_id,addr)
            }
```

```
            return $id
        }
       default {error "bad option : should be register args"}
    }
}
```

Aus den explizit durch Registrierung und den aus den Operationen des Benutzers implizit gesammelten Informationen werden Awareness-Nachrichten generiert. Solche Informationen können explizit vom Benutzer mittels den Funktionen `awinfo` und `user` abgefragt werden. Sie werden aber auch implizit nach dem Auftritt bestimmter Ereignisse generiert. Für die implizite Generierung und das Verschicken von Awareness-Nachrichten dienen beide Funktionen `snotify` und `table_notify`.

Programm 5.7 *`snotify`: Benachrichtigen von allgemeinen Awareness. Nach dem Öffnen einer Tabelle werden alle Benutzer, die ebenfalls auf die Tabelle zugreifen, benachrichtigt*

```
proc snotify {cmd args} {
    global clientInfo clients cid2id tableInfo fingerInfo
    set length [llength  $args]
    switch --  $cmd {
        "opentable" {
            if { $length != 2} {
                error "wrong  # args :
                        should be snotify opentable client tid"
             }
            set client [lindex $args 0]
            set tid    [lindex $args 1]
            set cindex [lsearch $tableInfo( $tid,clients)  $client]
            set other  [lreplace
                         $tableInfo( $tid,clients) $cindex $cindex]
            foreach Id  $other {
                set oc  $clientInfo( $Id,cid)
                set vp  $clientInfo( $Id, $tid,visProc)
                if { $vp != "none"} {
                    dp_RPC  $oc   $vp awareview addto table  $client
                }
            }
        }
        "newfocus" {...}
        "closetable" {...}
        "newfinger"  {...}
        "movefinger" {...}
        "deleteclient" {...}
    }
   ...
}
```

Fokus/Nimbus-Modell

Die Implementierung des Fokus/Nimbus-Modells basiert darauf, daß die Benutzer (die zugehörige Instanz des VClients) ihren Fokus als eine Liste von Rid's bei dem Server melden und ihren aktiven Finger angeben (siehe Abschnitt 4.3.2). Daraus können alle weiteren Informationen über Überschneidungen von Fokus und Nimbus generiert werden:

Programm 5.8 *`who_seeFocus`: Ermittelt die Benutzer, die einen Teil des Fokus eines Benutzers sehen. In `$clientInfo($client,$table,nlocks)` sind alle V-Sperren des Benutzers auf dieser Tabelle gespeichert, also der Fokus. In `$NLInfo($table,$rid,clients)` sind alle Clients eingetragen, die eine V-Sperre auf dieses Objekt besitzen. Hat ein Benutzer eine V-Sperre auf ein einziges Objekt aus dem Fokus, so sieht er einen Ausschnitt des Fokus*

```
proc who_seeFocus {tid client {what ""}} {
    global clientInfo tableInfo
    global NLInfo
    # client greift nicht auf die Tabelle, return 0
    if {[lsearch $tableInfo($tid,clients) $client] < 0} {
        return 0
    }
    set table $tableInfo($tid,catalog)
    set who $client
    set ccount 1
    foreach rid  $clientInfo($client,$table,nlocks) {
        # alle anderen, die <rid> auch sehen.
        foreach c [ldelete  $NLInfo($table,$rid,clients) $client] {
            if {[lsearch  $who  $c] < 0} {
                lappend who  $c ;# Falls noch nicht eingetragen
                incr ccount 1
            }
        }
    }
    if { $what == "count"} {
        return  $ccount;# Möglicherweise 1
    }
    return  $who;# Möglicherweise die leere Menge
}
```

Bei der Ausführung von Datenbankoperationen überprüft der Server, welche Benutzer den aktiven Finger, der die Operation ausgeführt hat, sehen. Die zugehörigen Instanzen des VClients werden mit den Auswirkungen dieser Operation benachrichtigt. Der VClient aktualisiert die Visualisierung anhand dieser Veränderungen und leitet sie an den Client weiter. Als Beispiel für die Generierung von Awareness-Informationen betrachten wir die Behandlung von Schreiboperationen (Programm C.2).

Die Prozedur `finger_write` führt Finger-Schreiboperationen aus. Sie kontrolliert Benutzerberichtigung, Sperre usw. vor der Ausführung der Operation. Kann die Ope-

`escher (application | list)` : Angepaßtes `escher` Kommando. Bietet dieselben Funktionen wie das urspr¨ungliche `escher`-Kommando, außer den Unterbefehlen `boot`, `shutdown` und `home`.
`table (open | list | new | close | delete | save)` : Angepaßtes `table` Kommando. Bietet dieselben Funktionen wie das TclDB-Kommando `table` . (Öffnen, Schließen, Erzeugen, L¨oschen und Speichern von Tabellen)
`finger (fork | free | list)` : Angepaßtes `finger` Kommando. (Erzeugen, L¨oschen und Listen von Fingern einer Tabelle). `finger list -table tid` listet alle Finger einer Tabelle, nicht nur f¨ur einen bestimmten Benutzer. Die Finger eines Benutzers auf einer Tabelle bekommt man mit `user info fingers -table tid`
`fid (push | pop ...insert | delete ...get | info ...)` : Angepaßtes `fid` Kommando. Stellt alle bekannten `fid` Funktionen zur Verf¨ugung (Navigieren, Schreiben, Lesen). Nur der Besitzer eines Fingers kann Operationen damit ausf¨uhren.
`register (vtable | vfinger | activefinger)` Registrierung von Informationen vom Benutzer bei dem Server. Registrierungen sind notwendig, um Awareness-Informationen generieren zu k¨onnen. Sie werden automatisch vom VClient erzeugt und bei dem Server gemeldet. Mit `vtable`, `vfinger` registriert eine VClient-Instanz f¨ur eine Tabelle bzw. einen Finger das zust¨andige Objekt. Diese Objekte stellen dem Awareness-Server ¨offentliche Methoden zur Verf¨ugung, die der Server z. B. bei Benachrichtigung verschickt. Mit `activefinger` wird der aktive Finger eines Benutzers auf einer Tabelle bestimmt.
`register newclient` *`client password`* Registriert einen neuen Benutzer. Dazu werden Benutzername und Passwort kontrolliert. Bei einem erfolgreichen Anmelden werden entsprechende Meldungen an die anderen Clients verschickt.
`awinfo (who | count)(-table | -active | -nimbus | -fokus)` : Explizites Abfragen von Awareness-Informationen. Diese betreffen sowohl Workspace-Awareness als auch Social-, und Informal-Awareness. Explizite Awareness-Abfragen k¨onnen vom VClient f¨ur die (generische) Darstellung von Awareness aufgerufen, aber auch direkt vom Benutzer angesteuert werden. Mit `who` wird das Ergebnis als eine Liste mit Client-Id's geliefert. Mit `count` wird nur der Anzahl der entsprechenden Clients geliefert. Mit der Option `-table` beschr¨ankt man die Abfrage auf Benutzer, die auf dieselben Tabelle zugreifen. Mit `-active` auf die, die im sichtbaren Bereich des Benutzers agieren. Mit `-nimbus` auf Benutzer, die den Nimbus des Benutzers sehen, und mit `-fokus` auf die, die den Fokus sehen.
`user (id | wlocks | rlocks | nlocks | info)` : Explizites Abfragen von Benutzerinformationen (Rechnername, Zeit usw.). Diese Abfragen k¨onnen auch als Awareness-Abfragen betrachtet werden. Dieser Befehl behandelt aber separat Awareness-Abfragen bezogen auf die Benutzer, also Social-Awareness Abfragen. Diese Informationen k¨onnen von jedem Benutzer auch ¨uber andere abgefragt werden. `info` liefert mit weiteren Optionen Informationen ¨uber den Benutzer. Ganz ohne Argument wird eine Liste mit allen gespeicherten Informationen ¨uber einen Benutzter geliefert.
`lock (info | set | free | islocked | who | rid)` : Behandlung von Sperren. Schreib-Sperren werden vom VClient f¨ur die Ausf¨uhrung von Schreiboperationen benutzt. F¨ur die Festlegung des Viewports eines Benutzers werden V-Sperren gesetzt. F¨ur visuelle Transaktionen k¨onnen Sperren ¨uber Methoden des VClients vom Benutzer explizit gesetzt werden.

Tabelle 5.1: Der Befehlssatz des Servers. Diese Befehle defi nieren mitsamt ihrer Unterbefehle die Schnittstelle zwischen dem Awareness-Server und dem VClient. Ausf¨uhrliche Beschreibung der Datenbankbefehle fi ndet man in Abschnitt **??**.

ration ausgeführt werden, so wird nach der Ausführung `table_notify` aufgerufen und damit werden alle Benutzer, die den aktiven Finger sehen, über die Ausführung der Operation benachrichtigt.

Die Kommunikation zwischen dem Datenserver und dem VClient basiert auf RPC (*Remote Procedure Calls*)[3]. Die Schnittstelle des Datenservers besteht aus Tcl-Kommandos, die mitsamt ihren Unterbefehlen logisch zusammenhängende Gruppen von Funktionen implementieren. Die Rückgabewerte werden als Funktionsergebnisse an den VClient geliefert. TclDB-Kommandos werden im Datenbankmodul des Servers neu implementiert, und werden vom VClient in der üblichen Syntax aufgerufen.

Tabelle 5.1 enthält eine kurze Übersicht über den Befehlssatz des Servers. Dabei beschränkt sich die Beschreibung auf die Hauptbefehle, ohne auf jeden Unterbefehl und jede Option einzugehen[4].

5.2 Der Virtuelle Client

Wie schon im vorigen Kapitel erwähnt, dient diese Schicht als Zwischenschicht zwischen dem Client und dem Datenserver. Die Hauptaufgabe dieser Schicht ist die Simulierung einer XML-basierten Datenbank und der Aufbau einer virtuellen Visualisierung in Form eines DOM-Baumes.

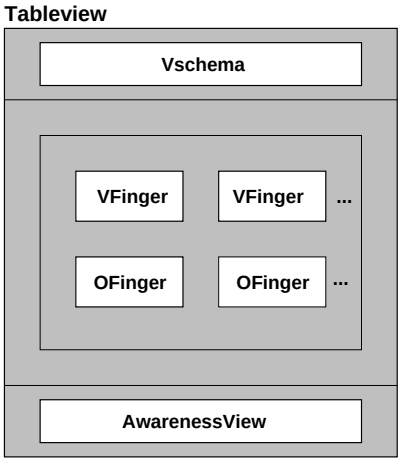


Abbildung 5.1: Aufbau von `Tableview`

[3] F¨ur die Kommunikation zwischen dem Server und dem VClient wurde TclDP eingesetzt. Siehe `http://simon.cs.cornell.edu/Info/Projects/zeno/Projects/Tcl-DP.html`.

[4] Eine detailliertere Beschreibung des Protokolls befi ndet sich unter `http://www.db.-informatik.uni-kassel.de/~morad/eclient`

Für jede Tabelle wird eine Instanz der Klasse `Tableview` erzeugt, die alle Operationen auf einer Tabelle sowohl vom Client (Datenbankoperationen) als auch vom Datenserver (Awareness-Nachrichten) ausführt. `Tableview` repräsentiert die gesamte Visualisierung einer eNF^2-Tabelle, deren Schema und die Awareness-Anzeige. Sie merkt sich die Fingerposition des Benutzers sowie die Objekte, die in seinem Viewport liegen. Außerdem merkt sie die Position der Finger aller anderen Benutzer, die einen Teil des Viewports des Benutzers sehen. Für die Darstellung der Tabelle selbst ist die Klasse `Vtable` implementiert. `Tableview` enthält weiterhin Referenzen auf ein `Awareness`-Objekt, das für die Awareness-Anzeige zuständig ist, und auf ein `Vschema`-Objekt, das die Darstellung des Schemas der Tabelle implementiert.

Programm 5.9 *`Tableview`: simuliert die Visualisierung einer Tabelle*

```
itcl::class Tableview {
    constructor {fid sfid args} {}
    destructor {
        foreach vf  $finger_list {
            catch {itcl::delete object  $vf}
        }
        foreach of $ofingers {
            catch {itcl::delete object  $of}
        }
    }
    private {
        variable tid           ""    ; # Id der Tabelle
        # sid, vschema, vtable, awareness, finger ...
        # finger_list, fingercount, activefinger, ofingers, ofingercount, ...
        method generateView {}
        # generateFingersColorSchema, updateAwareness, colorizeFingers
    }
    public {
         # Anweisungen vom Client:
        method draw {viewportH}
        method get {{what "table"}}
        method deletefinger  {{vf ""}}
        method setAwarenessMode {mode}
        method newfinger {}
        method push {}
        # pop, next, back, up, down
        method insertBefore {}
        # insertAfter, delete, deleteBefore
        # editAtomic, setAtomicValue, updateAtomicValue, endAtomicEditing
         # Anweisungen vom Server (Awareness).
        # newofinger, deleteofinger, replaceofinger, moveofinger,
        method visual {args}
        method awareview {method option value}
    }
}
```

Die Visualisierung einer eNF^2-Tabelle wird von `Vtable` durch einen DOM-Baum

dargestellt. Dazu wird eine generische Darstellung für solche Tabellen entwickelt, die die Struktur der Tabelle behält. Die Übersetzung einer eNF^2-Tabelle in einem DOM-Baum implementiert die Methode `Vtable::fingertodom`.

Programm 5.10 *`fingertodom`: erzeugt aus einer eNF^2-Tabelle einen isomorphen DOM-Baum*

```
itcl::body Vtable::fingertodom {} {
    set attr [$finger attr]
    set type [$finger type]
    set newNode [$domDoc createElement $attr]
    $newNode setAttribute id "[$finger rid]"
    $newNode setAttribute nf2type "$type"
    $newNode setAttribute bg "none"
    if {[$finger isatomic]} {
        if {[$finger isnull]} {
            $newNode setAttribute state "null"
            return $newNode
        } else {
            $newNode appendChild [$domDoc createTextNode "[$finger get]"]
            return $newNode
        }
    }
    if {[$finger istype -tuple]} {
        iterate $finger {$newNode appendChild [fingertodom]}
        return $newNode
    }
    if {[$finger isnull]} {
        $newNode setAttribute state "null"
        return $newNode
    }
    if {[$finger isempty]} {
        $newNode setAttribute state "empty"
        return $newNode
    }
    # none empty collection
    iterate $finger {$newNode appendChild [fingertodom]}
    return $newNode
}
```

Die virtuelle Visualisierung benutzt die Höhe eines atomaren Attributes als Maßeinheit. Es werden nur Teile der Tabelle gelesen, die auch tatsächlich vom Client gefordert werden. Der Client muß angeben, wieviele atomare Attribute in seinem Viewport passen oder wieviele gezeichnet werden sollen[5]. `Tableview` behält alle schon gelesenen Teile der Tabelle in dem lokalen DOM-Baum, sendet aber an den Client nur die Objekte, die in seinem Viewport liegen. Dafür merkt sich `Tableview` die Objekte, die im Viewport liegen und aktualisiert diese bei jeder Veränderung, die z. B. nach Scrollen oder Navigieren entstehen können.

[5] Das kann z. B. f¨ur jede Klasse von Ger¨aten in Konfi gurationsdateien festgehalten werden.

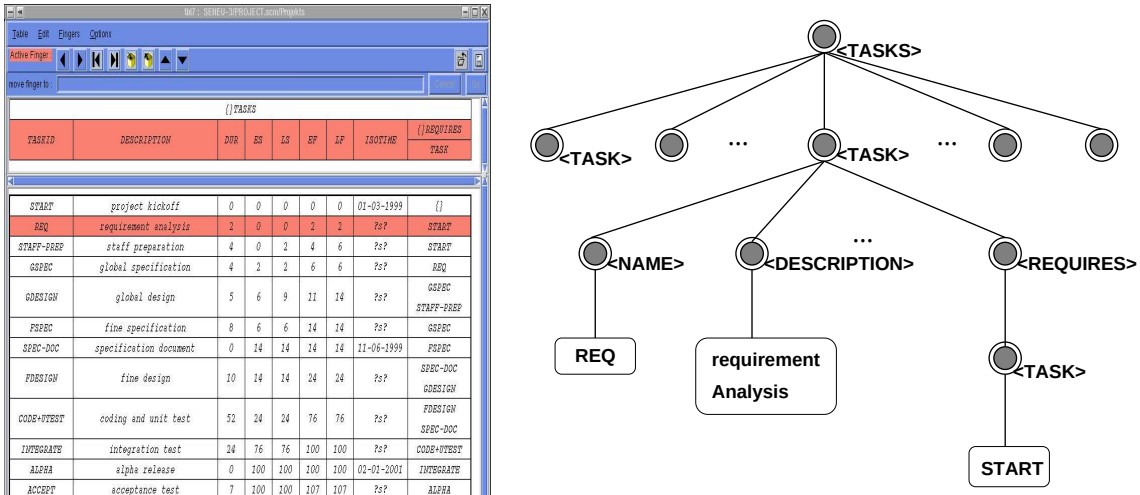


Abbildung 5.2: Darstellung von eNF^2-Tabelle als DOM-Baum

Visualisierungsfunktionen werden vom Datenserver aufgerufen, nachdem ein Benutzer eine Operation ausführt, die die Anzeige verändert. Sie sind als Unterbefehle der Methode `Tableview.visual` implementiert (Programm C.3). Diese ruft ihrerseits interne Methoden anderer Klassen z. B. `Vtable` auf. Für jede Datenbankoperation wird in der Klasse `Vtable` eine korrespondierende Methode implementiert, die die Veränderung im DOM-Baum durchführt. Als Beispiel für eine solche Methode ist die Methode `insertAfter`, die die Datenbankoperation `fid insert -after` visualisiert, im Anhang aufgelistet (siehe Programm C.4).

Nachdem die Visualisierung im DOM-Baum ausgeführt wird, wird die Visualisieung an dem Client übertragen. In Abhängigkeit vom Gerätetyp überträgt der VClient die Visualisierung in verschiedenen Formaten. Ist der Client ein üblicher Rechner, so wird die Vosualisierung in XML-Format übertragen. Der Client transformiert hier die Daten mit Hilfe einer Stylesheet in die Zielsprache. Bei kleinen mobilen Geräten wird das XML-Dokument vom VClient in die Zielsprache übersetzt und dann an dem Client übertragen. Die Zielsprache (HTML, SVG etc.) kann vom Client für jedes einzelne Dokument festgelegt werden.

Die Methode `generateView` stellt das ganze XML-Dokument zusammen und schickt es mittels des *MessageHandlers* an den Client.

Programm 5.11 *`Tableview.generateView`: Dabei gibt die Methode `Vtable.asXML` Teil des DOM-Baumes, der den Inhalt des Viewports darstellt, als XML-Dokument aus*

```
itcl::body Tableview::generateView {} {
    global CLIENT_TYPE
```

```
    switch -- $CLIENT_TYPE {
        "mobile" {
            set rootElement [$vtable rootDocument]
            set outRep [xsltTransform $rootElement [string trim $this :] ]
            tcldbMessager_view $outRep table [string trim $this :]
        }
        "pc" {
            set xmlRep ""
            append xmlRep "[$vtable asXML]\n"
            tcldbMessager_viewXML $xmlRep table [string trim $this :]
        }
        "applet" {
            set rootElement [$vtable rootDocument]
            set outRep [xsltTransform $rootElement [string trim $this :] ]
            tcldbMessager_view $outRep table [string trim $this :]
        }
        default {
            error "unknown client type"
        }
    }
}
```

Datenbankoperationen werden auf dieser Ebene untereinander synchronisiert. Dazu werden alle Fingeroperationen auf dieser Ebene neu implementiert. Die Ausführung einer Fingerschreiboperation besteht z. B. aus dem Verlangen einer Sperre, das Fordern einiger Informationen über Position des Fingers und Granularität und Typ des Objekts usw., die Ausführung bei dem Datenbankserver (atomar), die Generierung von Visualisierung und die Ermittlung dieser an der Anwendung und die Freigabe der Sperre. Programm 5.12 zeigt die Implementierung der Operation `insertBefore`.

Programm 5.12 *`insertBefore`:*

```
itcl::body Svfinger::insertBefore {} {
    if ![setlock] {return 0}
    $this hide
    if {[catch {$finger insert -before}]} {
        if {![$finger isatomic] && ([$finger isnull] || [$finger isempty])} {
            if {[catch {$finger insert}]} {
                unlock
                return 0
            } else {
                set rid [$finger rid]
                $tableview visual 0 insert $this $rid
            }
        } else {
            unlock
            return 0
        }
    } else {
        set rid [$finger rid]
```

```
        $tableview visual 0 insert -before $this $rid
    }
    unlock
    return 1
}
```

Bei dem VClient gibt es zwei Klassen von Fingern. `VFinger` repräsentieren Finger des zugehörigen Benutzers, `OFinger` stellen Finger anderer Benutzer dar. In Abhängigkeit davon, ob Finger sich überdecken oder überschneiden werden sie unterschiedlich dargestellt (z. B. werden Finger rot dargestellt, wenn zwei Finger auf dasselbe Objekt zeigen). Nach der Ausführung von Navigations- und Schreiboperationen wird das Farbschema für die ganze Tabelle in Abhängigkeit des bewegten Fingers neu berechnet, und im DOM-Baum durch das Setzen der Werte für den Attribut `bg` eingetragen. Programm C.5 zeigt die Methode `colorizeFingers`, die das Färben der Finger in der Tabellenanzeige implementiert.

5.3 Die Client-Schnittstelle

Für den Anwendungsentwickler steht eine Java-Schnittstelle zur Verfügung. Diese sorgt für die Kommunikation mit dem VClient über eine Socket-Verbindung, implementiert Methoden für die Behandlung von XML-Dokumenten und das Verschicken von Befehlen an den VClient und stellt Standard-Visualisierungsklassen zur Verfügung.

5.3.1 Kommunikation mit dem VClient

Für die Kommunikation mit dem VClient dient die Klasse `TclDBSocket`. Diese stellt die Verbindung zum VClient her und stellt Methoden zum Verschicken von Befehlen an den VClient und zum Lesen von Nachrichten bereit. `TclDBSocket` arbeitet mit reinem Text und verwendet daher andere Klassen zum Bearbeiten der Nachrichten. Eine spezielle Stream-Klasse (`WrappedInputStream`) sorgt für das Lesen aus der Socket, und simuliert damit eine übliche Stream-Klasse. Für das Verschicken von Befehlen an den VClient stellt `TclDBSocket` vier öffentliche Methoden zur Verfügung (siehe Programm C.6)

`exec`: schickt einen Befehl an den VClient und kehrt gleich zurück, ohne auf eine Antwort zu warten.

`execR`: schickt einen Befehl an den VClient und wartet auf die Antwort. Kommt für eine bestimmte Zeit keine Antwort zurück, so wird eine `TclDBEmptyMessageException` geworfen. Der Returnwert wird aus dem XML-Dokument extrahiert und als Rückgabewert zurückgeliefert.

`execReturnDom`: Ähnliche Funkion wie `execR`, liefert aber als Rückgabewert das ganze XML-Dokument als DOM-Baum zurück.

`execV`: wie `execR`, die Antwort wird aber an den `TclDBMessageHandler` weitergeleitet. `TclDBMessageHandler` bearbeitet das (XML)-Dokument in der Nachricht und ruft die Visualisierungsmethoden der aufrufenden Klasse auf, um die Nachricht zu visualisieren. Das aufrufende Objekt wird an `execV` übergeben und an `TclDBMessageHandler` weitergegeben. Dies ist auch aus dem Header der Nachricht ermittelbar. Stimmen die beiden nicht miteinander überein, so wird eine Fehlermeldung erzeugt[6].

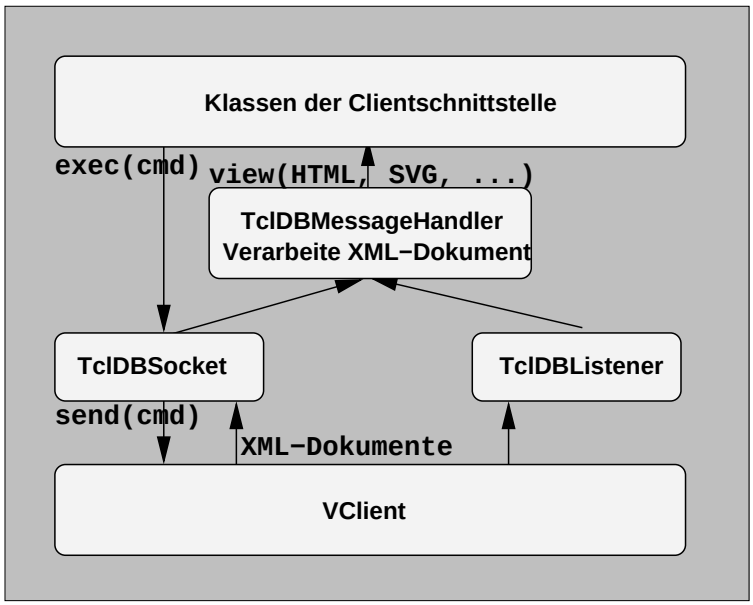


Abbildung 5.3: Kommunikation zwischen dem VClient und der Client-Schnittstelle

Nachrichten, die nicht als Antworten auf Befehlen von dem VClient kommen, werden von einer parallel laufenden *Thread* über eine Instanz der Klasse `TclDBListener` gefangen und an den `TclDBMessageHandler` übergeben. Dieser ermittelt aus dem Header der Nachricht das Objekt, für das die Nachricht gedacht ist, verarbeitet und leitet sie an das Zielobjekt weiter.

Nachdem die Verbindung zum VClient hergestellt wird, erzeugt der Konstruktor von `TclDBSocket` eine Instanz der Klasse `TclDBListener`, die auf Benachrichtigungen vom VClient wartet. Anschließend werden Objekte erzeugt (`escher` und `table`), die die Ausführung von Datenbankoperationen auf Clientseite ermöglichen.

[6] Im Java-Sprachgebrauch `Exception`.

5.3.2 Ausf¨uhrung von Datenbankoperationen

Für die Ausführung von Operationen auf den gemeinsamen Informationsraum sind die drei Klassen `EscherCmd`, `TableCmd` und `Vtable` implementiert. Diese sind zwar an die TclDB-Kommandos angelehnt, jedoch abstrahieren sie die Ausführung von Operationen auf die Datenbank enorm. Weiterhin sind Datenbankoperationen meistens für den internen Gebrauch innerhalb der Clientschnittstelle und werden daher selten von dem Anwendungsentwickler direkt aufgerufen.

Die Klasse `EscherCmd` implementiert Methoden für die Auswahl einer Applikation sowie zum Auflisten vorhandener Applikationen oder Tabellen und Schemata einer bestimmten Applikation. `TableCmd` stellt Methoden zum Öffnen, Schließen, Anlegen und Löschen von Tabellen zur Verfügung. Nachdem eine Tabelle geöffnet wird, operiert die Clientschnittstelle auf geschachtelte Dokumente mit den generischen Methoden der Klasse `Vtable`.

Für jede geöffnete Tabelle wird eine Instanz der Klasse `Vtable` (siehe Programm C.8) erzeugt, die automatisch einen Finger auf der Tabelle erzeugt. `Vtable` stellt Methoden zur Verfügung, mit denen alle Operationen auf der Tabelle ausgeführt werden können, dabei werden diese Operationen bei dem VClient mit dem aktiven Finger des Benutzers ausgeführt.

5.3.3 Verarbeiten von Nachrichten

Nachrichten vom VClient werden als XML-Dokumente geschickt. Zur Verarbeitung dieser Dokumente ist die Klasse `TclDBMessageHandler` zuständig. Nachdem eine Nachricht ankommt, sperrt `TclDBSocket` den `InputStream` der Socket und ruft die Methode `TclDBMessageHandler.handleMessage` auf. Diese liest erst den Header der Nachricht und entscheidet anhand der Headerinformationen, wie die Nachricht zu behandeln ist (siehe Abschnitt 4.5).

Programm 5.13 *`TclDBMessageHandler.handleMessage`*

```
public class TclDBMessageHandler {
    public synchronized void handleMessage (Tableview viewer,
                                            InputStream from)
        throws ParserConfigurationException, IOException {
        WrappedInputStream wrappedReader = new WrappedInputStream(from);
        TclDBHeader header=readMessageHeader(wrappedReader);
        Tableview v = getXMLView(header);
        if (v != viewer) {
            System.out.println("unknown viewer objekt");
        }
        viewMessage(viewer, header, wrappedReader);
        wrappedReader.close();
    }
}
```

Die Methode `viewMessage` ruft anhand des Nachrichtentyps weitere Methoden auf, um die Nachricht zu verarbeiten und übergibt das verarbeitete Dokument an die entsprechenden Methoden des `Tableview` Objekts (siehe Programm C.7).

5.3.4 Visualisierung

Eine Visualisierungsklasse ist eine Klasse, die ein eNF^2-XML-Dokument darstellt und die generische Navigationsoperationen auf dieses Dokument implementiert. Sie kann die Struktur (Schema) des Dokuments visualisieren, und enthält grafische Objekte für die Darstellung von Awareness. Diese generischen Operationen auf ein XML-Dokument sind in dem Interface `Tableview` zusammengestellt. Die abstrakte Klasse `AbstractTableview` (siehe Programm C.9) implementiert den generischen Teil dieser Methoden und soll daher von jeder Visualisierungsklasse geerbt werden. Die Clientschnittstelle implementiert zwei Visualisierungsklassen: `TclDBHTMLTabelview`, und `TclDBSVGTableview`. Weitere Visualisierungsklassen können vom Anwendungsentwickler implementiert werden.

Die Visualisierung der XML-Dokumente basiert auf der Benutzung von Stylesheets, die diese XML-Dokumente in andere Formate (HTML, SVG) umwandeln[7], und diese Dokumente in den entsprechenden Anzeige-Widgets darstellen. Für die Transformierung eines XML-Dokuments registriert jede Visualisierungsklasse durch die Methode `TclDBMessageHandler.setTransformer` eine Referenz auf ein `TclDBTransformer`-Objekt (siehe Programm C.10). Dabei erzeugt `TclDBMessage-Handler` einen neuen Transformer für jedes verwendete Stylesheet[8].

Das Viewport wird von einer Visualisierungsklasse durch einen DOM-Baum dargestellt und verwendet dazu eine Referenz auf ein `DomNavigation`-Objekt (siehe Programm C.11). Jede Visualisierungsklasse implementiert zwei Methoden für die Aktualisierung der Anzeige: `setView(Document dom)` ersetzt das aktuelle Viewport mit dem neuen DOM-Baum, und `mergeView(Document dom)` aktualisiert den Viewport anhand der Änderungen im DOM-Baum. Beide Methoden rufen die Methode `TclDBTransformer.transform(Document newdom)` auf, um das XML-Dokument in das gewünschte Format umzuwandeln und dann die Methode `NF2Viewer.view(Document visualNode)` um das Dokument im Viewport darzustellen.

---

[7] Vorstellbar ist auch die Verarbeitung des XML-Dokuments durch die DOM-Schnittstelle und die Erzeugung einer grafischen Oberfläche mit den Java-Swing-Klassen

[8] Mehrere Visualisierungsklassen können daher denselben Transformer verwenden.

5.3.5 Awareness-Nachrichten

Workspace-Awareness wird durch die Aktualisierung der Anzeige und die Darstellung der Finger im Viewport visualisiert. Zusätzlich kann jede Visualisierungsklasse eine Awareness-Anzeige für Group-Structural-Awareness enthalten. Diese zeigt eine Liste der Benutzer, die anhand des *Awareness-Modes* bestimmt werden können. Awareness-Nachrichten kommen als XML-Dokumente an. Sie werden vom `TclDBMessageHandler` in einem DOM-Baum transformiert und an die Methode `viewAwarenessMessage` des Visualisierungsobjekts übergeben. Diese übergibt den DOM-Baum an das zugehörige `TableAwarenessInfo`-Objekt (siehe Programm C.12) weiter und ruft die `draw`-Methode des zugehörigen `AwarenessPanel`-Objekts auf, um die Anzeige zu aktualisieren.

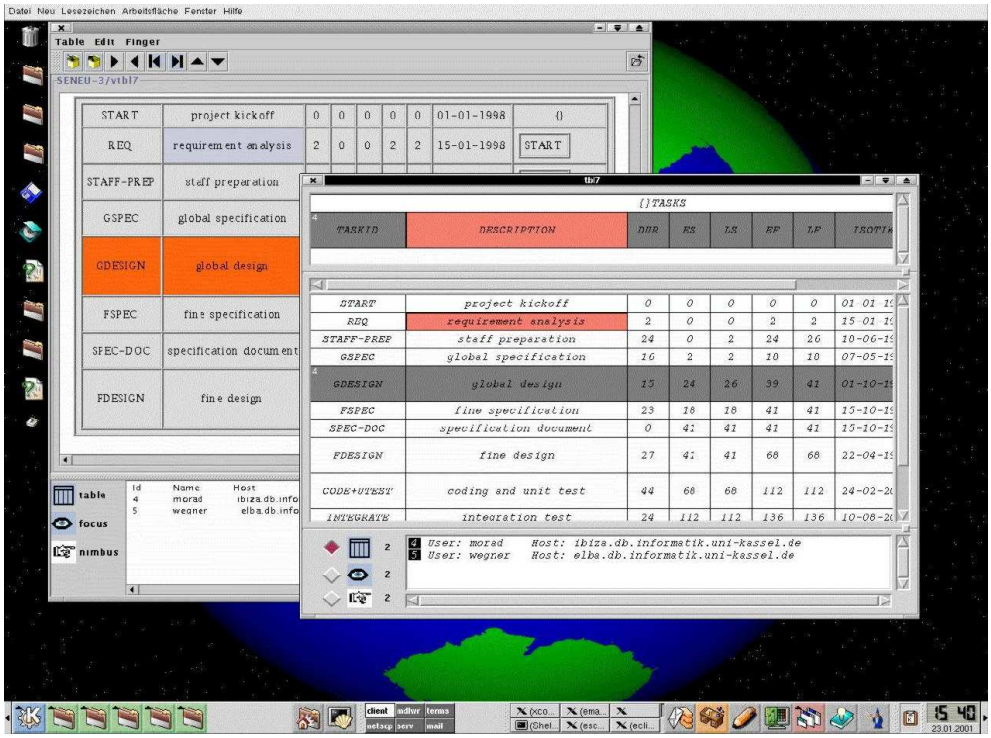


Abbildung 5.4: Tabellenanzeige bei zwei verschiedenen Benutzern

5.3.6 Beispiel

Als Beispiel dafür, wie die verschiedenen Schichten miteinander kommunizieren, dient die explizite Abfrage von Informationen über einen bestimmten Benutzer. Die Klasse `AwarenessPanel` zeigt in einer Anwendung eine Liste von Benutzern an. Informationen über einen bestimmten Benutzer können aus dieser Liste aufgefordert werden. Mit `tcldbSocket.execReturnDom("user info " + id)` wird

die Anweisung an dem VClient geschickt. Das als XML-Dokument zurückgelieferte Ergebnis wird vom `TclDBMessageHandler` als DOM-Baum an das aufrufende Objekt geliefert. Die Methode `drawUserInfo(Document doc)` der Klasse `TclDBStandardUserViewer` bekommt einen DOM-Baum mit den Benutzerinformationen und erzeugt daraus eine Anzeige in einem Textelement.

Programm 5.14 *drawUserInfo*

```
private static void drawUserInfo(Document doc, JTextPane textArea) {
    String nickName, vollName, hostName, dscrText, loginTime, imageSrc;
    Node tmpNode, fNode, attrNode;;
    NodeList children;
    NamedNodeMap attributes;
    clear(textArea);
    Node userinfo = (Node) doc.getDocumentElement();
    tmpNode = TclDBDOMUtils.getNodebyName (userinfo, "login");
    nickName = tmpNode.getFirstChild().getNodeValue();
    appendText(textArea, "loginname: ", "head");
    appendText(textArea, nickName + "\n", "value");
    int i = 0, j = 0;
    String id = "", name="", application="", fId="", fActive="0";
    NodeList fingerChildren;
    Node tables = TclDBDOMUtils.getNodebyName (userinfo, "tables");
    children = tables.getChildNodes();
    if(children != null) {
        appendText(textArea,"tables:\n", "head");
        for (i = 0; i < children.getLength(); i++) {
            Node  tableNode = (Node) children.item(i);
            if (tableNode.getNodeType() == Node.ELEMENT_NODE) {
                attributes = tableNode.getAttributes();
                appendText(textArea, rowTab + name + ":\n", "value");
                // Zeige entsprechende Daten über eine Tabelle im Textelement.
                tmpNode = TclDBDOMUtils.getNodebyName(tableNode, "fingers");
                fingerChildren = tmpNode.getChildNodes();
                if (fingerChildren != null) {
                   // Verarbeite die Finger der Tabelle.
                   for (j = 0; j < fingerChildren.getLength(); j++) {
                       fNode = (Node) fingerChildren.item(j);
                       if (fNode.getNodeType() == Node.ELEMENT_NODE) {
                          attributes = fNode.getAttributes();
                          if(attributes != null) {
                              attributes = fNode.getAttributes();
                              attrNode = attributes.getNamedItem("id");
                              fId = attrNode.getNodeValue();
                              ...
                          }
                       }
                    }
                }
        ...
}
```

Der VClient führt dazu die Methode `getUserInfo`. Dabei ist `user` eine VClient-Methode die Benutzerinformationen bei dem Server abfragt.

Programm 5.15 *`getUserInfo`*

```
proc getUserInfo {user} {
    global userInfo
    set xmlRep "<userinfo user=\"$user\">"
    # Lese alle Informationen vom Server.
    set userInfoStr [user info -id $user]
    foreach v $userInfoStr {
        # erstes element als name, der Rest als Wert.
        set serverInfoArray([lindex $v 0]) [lrange $v 1 end]
    }
    append xmlRep "<login>$serverInfoArray(user)</login>"
    append xmlRep "<vollname>$serverInfoArray(vollname)</vollname>"
    append xmlRep "<description>$serverInfoArray(description)</description>"
    append xmlRep "<image src=\"$serverInfoArray(image)\"/>"
    append xmlRep "<host>$serverInfoArray(host)</host>"
    append xmlRep "<logintime>[timeFormat $serverInfoArray(c_time)]\
                  </logintime>"
    set tables $serverInfoArray(tables)
    append xmlRep "<tables>"
    foreach t $tables {
        array set tableInfoArray [table info $t all]
        set tName $tableInfoArray($t,name)
        set tApp $tableInfoArray($t,application)
        append xmlRep "<table id=\"$t\" name=\"$tName\"\
                                        application=\"$tApp\">"
        append xmlRep "<fingers>"
        set fingers $serverInfoArray($t,fingers)
        set activeFinger $serverInfoArray($t,activefinger)
        foreach f $fingers {
            if {$f == $activeFinger} {
                append xmlRep "<finger id=\"$f\" active=\"1\">"
            } else {
                append xmlRep "<finger id=\"$f\">"
            }
            append xmlRep "</finger>"
        }
        append xmlRep "</fingers>"
        append xmlRep "</table>"
    }
    append xmlRep "</tables>"
    append xmlRep "</userinfo>"
    tcldbMessager_xmlResponde $xmlRep "system"
}
```

Die vom Datenerver gelieferten Daten sind die Rückgabewerte der Tcl-Prozeduren. Die Abfrage `user info` *`user`* liefert eine Liste von Paaren bestehend aus Schlüsselwörtern und Werten. Als konkretes Beispiel betrachten wir die Abfrage über den Benutzer

`wegner`. Das gelieferte Ergebnis vom Datenserver sieht wie folgt aus:

```
{cid tcp2} {addr 141.51.180.21}
{user wegner} {password H^m;hJa}
{host ibiza} {registred 1}
{image images/wegner.gif}
{vollname Prof. Dr. Lutz Wegner} {type 3}
{description Prof. Dr. Lutz Wegner. Leiter
der Gruppe Praktische Informatik.}
{application tests} {tables tbl5 tbl7}
{fingers tbl5.f3 tbl7.f3} {fingerCount 2}
{c_time 1027585081} {w_time 1027585081}
{tbl5,fingers tbl5.f3} {tbl5,fingerCount 1}
{tbl5,visProc ::vtbl5} {tbl5,schemaFinger tbl6.f1}
{tbl5,visFinger tbl5.f2} {tbl5,activefinger tbl5.f3}
{tbl7,fingers tbl7.f3} {tbl7,fingerCount 1}
{tbl7,visProc ::vtbl7} {tbl7,schemaFinger tbl8.f1}
{tbl7,visFinger tbl7.f2} {tbl7,activefinger tbl7.f3}...
```

Der VClient baut daraus folgendes XML-Dokument:

```
<?xml version='1.0' encoding="ISO-8859-1" standalone="yes"?>
<userinfo  user="4">
   <login>wegner</login>
   <vollname>Prof. Dr. Lutz Wegner</vollname>
   <description>
       Prof. Dr. Lutz Wegner. Leiter der Gruppe
       Praktische Informatik.
   </description>
   <image src="images/wegner.gif"/>
   <host>ibiza</host>
   <logintime>24/07/2002:17:20:23</logintime>
   <tables>
      <table id="tbl13" name="a-g" application="Projekts">
         <fingers><finger id="tbl13.f3" active="1"></finger></fingers>
      </table>
      <table id="tbl15" name="Abteilungen.tbl" application="tests">
         <fingers><finger id="tbl15.f3" active="1"></finger></fingers>
      </table>
   </tables>
</userinfo>
```

5.3.7 Anwendung

Die Entwicklung einer synchronen Groupware-Applikation mit Hilfe der Client-Schnittstelle wird sehr stark vereinfacht und ist sogar vergleichbar mit der Implementierung von Einbenutzer-Applikationen. Zusätzlich zu den in diesem Abschnitt beschriebenen Klassen sind weitere Hilfsklassen für oft verwendete grafische Komponenten (Widgets) implementiert.

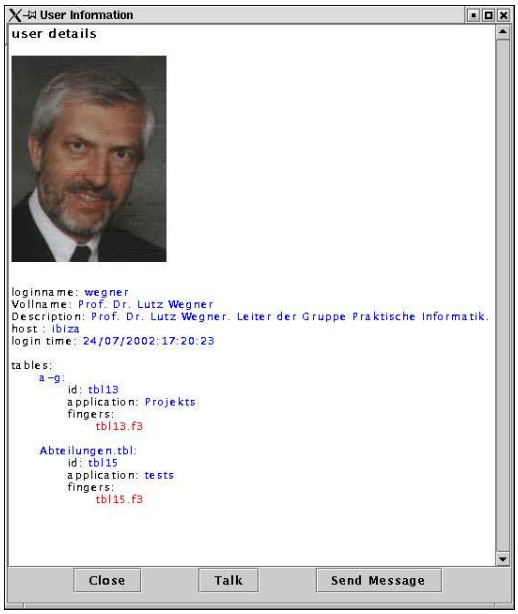


Abbildung 5.5: Anzeige von Benutzerinformationen mit `TclDBStandardUserViewer`

Als Anwendungsbeispiel betrachten wir ein Java-Applet, das von einem beliebigen Browser gestartet werden kann. Das Applet zeigt in seinem Hauptfenster die Applikationen im gemeinsamen Informationsraum, aus denen der Benutzer eine auswählen kann. Auf der linken Seite werden die Tabellen und Schemata der Applikation aufgelistet (siehe Programm C.13).

Als erstes wird eine Verbindung zum Server hergestellt.

```
public class escherC extends JApplet implements ActionListener {

private final String theHost ="elba.db.informatik.uni-kassel.de";
private final int thePort = 9009;
...
tcldb = new TclDB(theHost, thePort);
public Container makeContentPane () {
   try {
         if (tcldb.connect() == false) {
             System.out.println("connection failed! exit...");
             System.exit(1);
         }
         tcldb.login(this);
   } catch (...) {}
   ...
   }
...
}
```

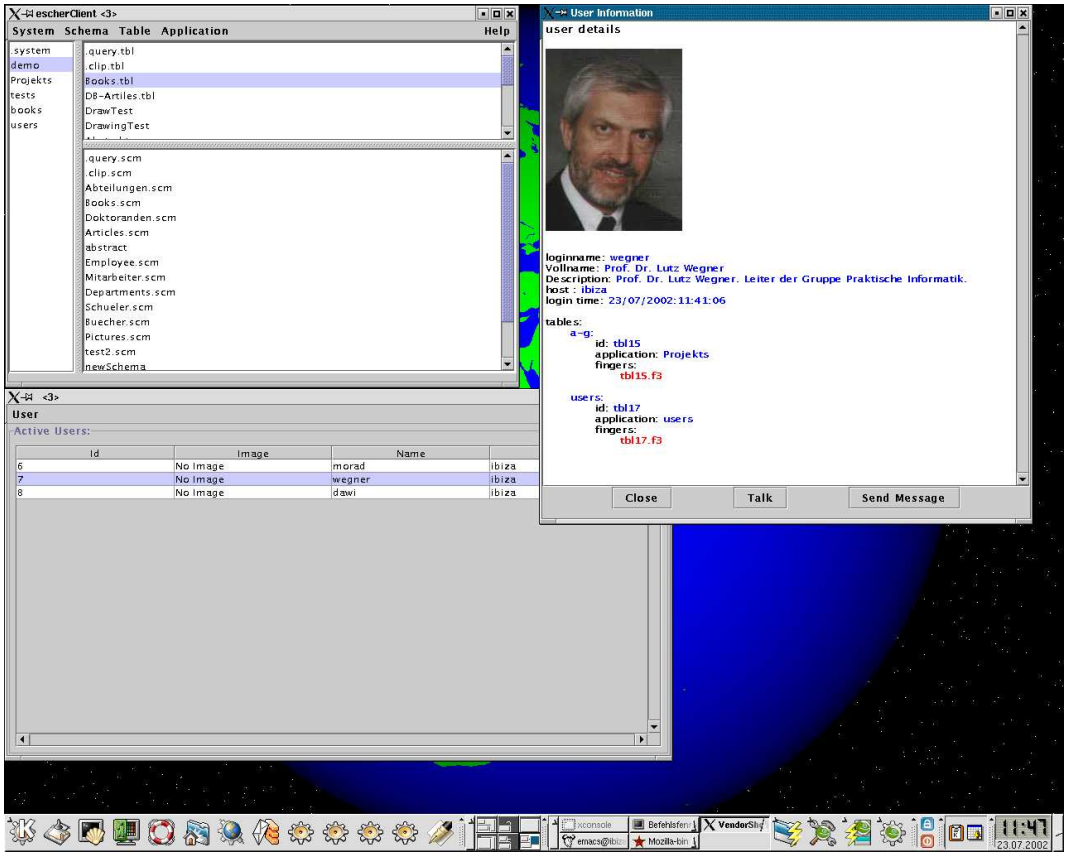


Abbildung 5.6: Hauptfenster einer Benutzerin

Nach dem eine Verbindung hergestellt wird werden Drei Widgets für die Auflistung der Applikationen, Tabellen und Schemata erzeugt. All diese Widgets werden von der Clientbibliothek zur Verfügung gestellt. Die Klasse `TclDBMainPanel` erzeugt aus den Drei Listobjekte ein Fenster indem die Listen wie in Abbildung 5.6 dargestellt werden:

```
try {
   TclDBTableListViewer tableListViewer =
                        new TclDBStandardTableList(tcldb);
   TclDBSchemaListViewer schemaListViewer =
                        new TclDBStandardSchemaList(tcldb);
   TclDBApplicationListViewer appListViewer =
                        new TclDBStandardApplicationList(tcldb);
   atsList = new TclDBMainPanel(tcldb,
             (TclDBAbstractApplicationListViewer) appListViewer,
             (TclDBAbstractTableListViewer) tableListViewer,
             (TclDBAbstractSchemaListViewer) schemaListViewer);
} catch(TclDBEmptyMessageException ex4) {
    ex4.printStackTrace();
}
```

Dannach wird ein Widget für die Anzeige vom Benutzerinformationen erzeugt und registriert, damit andere Komponenten es für Aktualisierung vom Benutzerinformationen verwenden können:

```
TclDBUsersFrame usersFrame = new TclDBUsersFrame(tcldb);
tcldb.setTcldbUsersViewerFrame(usersFrame);
TclDBStandardUserViewer.initialize(tcldb, mainFrame,
                                   "User Information");
```

Nach dem Öffnen einer Tabelle wird ein Finger auf dieser Tabelle erzeugt und der Ausschnitt der Tabelle dargestellt, der in den Viewport paßt. Die Darstellung der Tabelle erfolgt mit dem Standardstylesheet. Dadurch wird der aktive Finger des Benutzers und die Finger der anderen Benutzer in seinem Viewport ebenfalls dargestellt.

Mit dem aktiven Finger kann der Benutzer durch die üblichen Tasten auf das Dokument navigieren und editieren. Dabei werden alle Veränderungen im Dokument in die Anzeige eingebaut, so daß jeder Benutzer die Aktivitäten der anderen verfolgen kann.

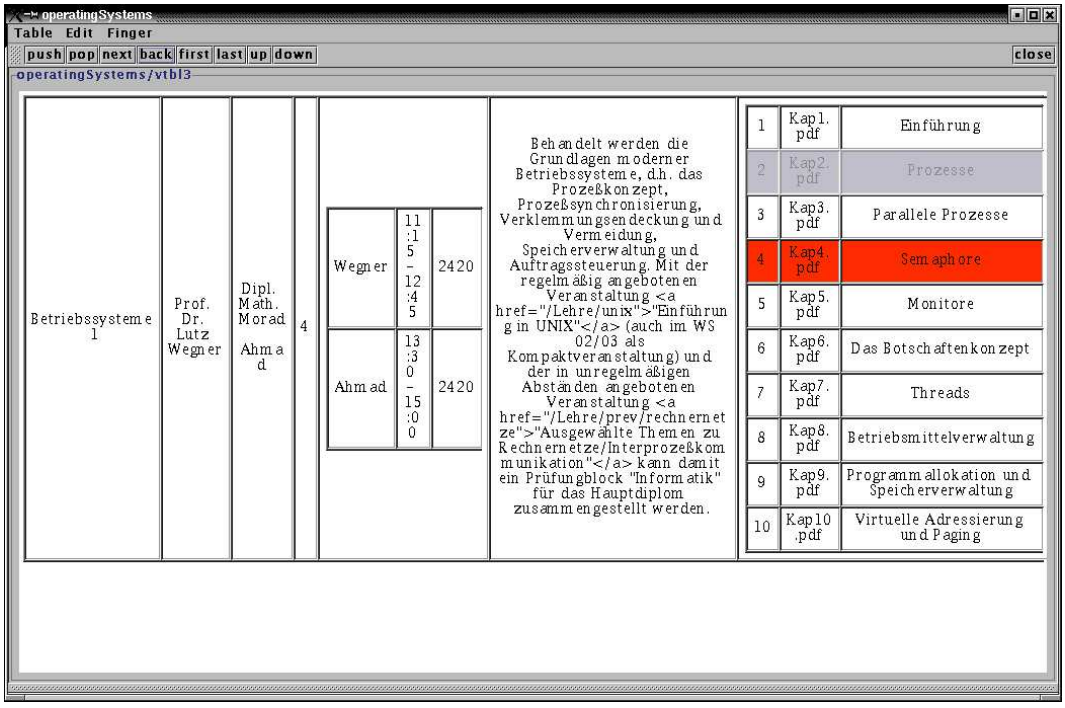


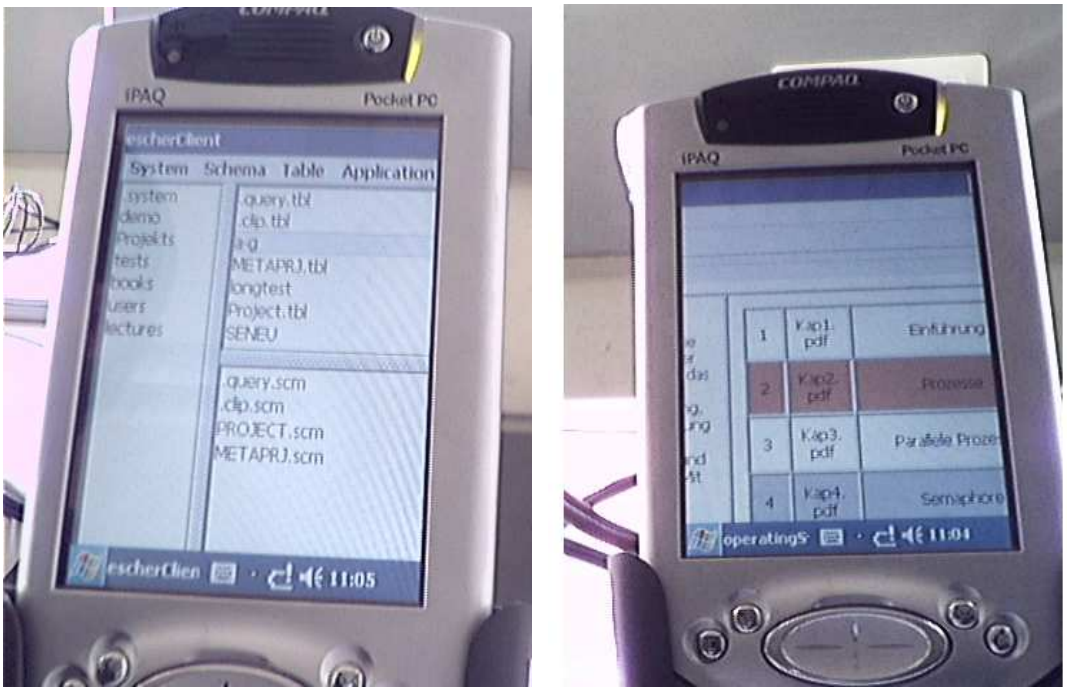
Abbildung 5.7: Darstellung der Tabelle `operatinSystems` auf einen Desktop PC

Abbildung 5.7 zeigt die Standardvisualisierung der Tabelle `operatinSystems` auf einem Desktop PC. Diese enthält alle Daten über die Vorlesung Betriebssyteme WS 02/03. Finger der anderen Benutzer werden in dieser Visualisierung grau dargestellt (hier auf Kap2). Abbildung 5.6 zeigt das Hauptfenster einer Benutzerin `dawi`. Das rechte obere Fenster zeigt detaillierte Informationen über den Benutzer `wegner`. Das

untere Fenster enthält eine Liste aller im System angemeldeten Benutzer im Gegensatz zu Awareness-Anzeige in einer Tabelle, wo nur Benutzer in Abhängigkeit des Awareness-Modus angezeigt werden. Zu den Detailinformationen gehören unter anderem die Tabellen, auf die der Benutzer zugreift, und die Finger, die er auf dieser Tabelle hat.

Auf einem Pocket PC

Um die Möglichkeiten für die Verwendung dieses Modells bei mobilen Geräten darzustellen, wurde die Java-Schnittstelle und der Client auf einem iPAQ 3970 Pocket PC portiert. Dieser besitzt einen 400 MHz Intel PXA250 Prozessor, 64 MB RAM, 48 MB Flash-ROM und eine wechselbare SD-Karte (*Secure Digital*) der Größe 64 oder 128 MB. Die Anzeige hat 240 X 320 Pixeln mit 65.536 Farben. Sein Gewicht ist 190g.



(a) Hauptfenster (b) Tabellendarstellung

Abbildung 5.8: Benutzeranwendung auf dem iPAQ 3970

Als Java virtuelle Maschine wurde Jeode eingesetzt [104] (siehe Abschnitt 2.3.2). Bei der Portierung könnten dieselben Softwarebibliotheken wie auf dem PC verwendet werden, wobei einige Modifikationen vorgenommen werden müssten. Am Quellcode waren ebenfalls wenige Modifikationen notwendig. Für die Programmierung

der grafischen Schnittstelle, wurde die `swing`-Bibliothek verwendet[9]. Die Pakete `xercesImpl.jar`, `jdom.jar` und `svg.jar` könnten ohne jegliche Modifikation eingesetzt werden. Das Stylesheet-Prozessor xalan wurde nicht verwendet, da die Transformation der XML-Dokumente vom VClient, auf der Serverseite, ausgeführt wird.

Abbildungen 5.8(a) zeigt das Hauptfenseter der Anwendung. Abbildung 5.8(b) zeigt die Darstellung derselben Tabelle aus Abbildung 5.7.

5.4 Eingesetzte Software

Die Entwicklungsplattform ist ein Intel Pentium II mit SuSE Linux 7.3, 8.0 und 8.1. Der Datenserver wurde mit Tcl 8.4 [8] implementiert. Als Datenbank wurde der Datenbankeditor Escher [219] verwendet. Der Zugriff auf die Datenbank wurde mit der Tcl-Schnittstelle zu Escher, TclDB [18], realisiert. Für die Kommunikation zwischen dem Datenserver und dem VClient wurde die RPC-Implementierung für Tcl, TclDP 4.0 [9], benutzt.

Die Zwischenschicht (VClient und der VClient-Generator) wurde ebenfalls mit Tcl 8.4 und der objektorientierten Erweiterung Itcl 3.2 implementiert. [1]. Für die Erzeugung mehrerer parallelen Prozesse wurde TclX 8.4 [10] eingesetzt. Als XML-Parser und Implementierung der DOM-Schnittstelle diente das Packt tDOM Version 0.75 [11]. tDOM benutzt den Expat XML-Parser von *James Clark* und enthält einen sehr schnellen Stylesheet-Prozessor, der in C implementiert ist.

Die Client-Schnittstelle wurde mit JDK 1.3 [2] implementiert. Das Paket xerces [15] diente als XML-Parser, xalan [14] als Stylesheet-Prozessor und die „*Open Source SVG Toolkit*" [5] zur Darstellung von SVG-Dokumenten. Die JDOM-Schnittstelle [3] wurde ebenfalls zum Parsen von kleinen XML-Dokumenten eingesetzt.

[9] Je ode unterstützt diese Bibliothek nicht, jedoch könnte dieses Paket nach einiger Änderungen verwendet werden.

Kapitel 6

Ausblick

In dieser Arbeit wurde ein generisches Modell für synchrone Gruppenarbeit auf gemeinsamen Informationsräumen entwickelt. Grundidee war die Bereitstellung einer einfachen und intuitiven Schnittstelle zu Informationsräumen, auf die in naher Zukunft aus verschiedenen Orten mit Hilfe von mobilen Geräten zugegriffen wird. Sowohl die Struktur der Daten als auch die Operationen sind generisch, vom Maschinentyp unabhängig. Lediglich die Fähigkeit XML-Daten anzuzeigen, wird von dem Anzeigegerät vorausgesetzt.

Es wurde auf ein vorhandenes Datenmodell (eNF^2-Datenmodell) aufgesetzt, das Navigation mit einem Cursor (Finger) auf geschachtelte Tabellen ermöglicht. Die Implementierung der Nebenläufigkeitskontrolle und Awareness haben dabei nur die Struktur der XML-Dokumente vorausgesetzt, nicht aber das zugrundeliegende Datenmodell.

Die Generierung von XML-Dokumenten aus eNF^2-Tabellen wurde anhand einiger Beispielen aus dem Quellcode erläutert. Die Visualisierung basierte auf der Transformierung von XML-Dokumenten in anderen XML-basierten Sprachen wie HTML oder SVG durch XSLT-Stylesheets. Zwei Arten solcher Stylesheets wurden vorgestellt. Generische Stylesheets erzeugen eine Visualisierung von einem XML-Dokument anhand seiner Struktur. Spezielle Stylesheets basieren auf den Inhalt eines Dokuments. Für die Visualisierung von Schemainformationen wurde ebenfalls ein Stylesheet angegeben.

Die Benutzung des Fingermechanismus stellte eine wichtige Verbindung zwischen Awareness und der Kontrolle der Nebenläufigkeit dar. Der Finger, mit dem Operationen ausgeführt werden, gibt gleichzeitig Auskunft über die Präsenz der Benutzer. Wichtig war dabei die Einführung des Begriffs der visuellen Transaktion und die Formulierung von Korrektheitskriterien, aus denen Verfahren für die Bewahrung der Konsistenz entwickelt wurden. Mit Hilfe von visuellen Sperren könnte die Definition der aktuellen Visualisierung und der korrekten Operation auf das Viewport eingeschränkt werden. Diese Einschränkung gewährt einerseits ein sinnvolles Korrektheitskriterium, andererseits einen hohen, in der Praxis akzeptablen Grad an Parallelität. Die Einfachheit dieses Metaphers ermöglicht ungeübte Benutzer Awareness zu verstehen.

Die vier wichtigsten Themen Nebenläufigkeit, Awareness, Visualisierung, Daten- und Interaktionsmodell wurden vertieft behandelt. Als Forschungsarbeit war es wichtig, Lösungen für diese Aufgaben sowie passende prototypische Implementierungen vorzustellen. In der Praxis gehören weitere Komponenten dazu, die in dieser Arbeit nicht ausführlich behandelt wurden. Es müssen z. B. Verfahren für die Benutzer- und Zugriffsverwaltung implementiert werden.

Eine Java-Anwendung wurde sowohl auf einem Desktop PC, als auch auf einem Pocket PC entwickelt. Es fehlten die Zeit und die personellen Möglichkeiten, ausgereifte spezielle Testanwendungen zu entwickeln oder den Einsatz bei Mobiltelefone zu untersuchen. Offene Fragen für die spätere Forschung sowie die Weiterentwicklung des Prototyps werden hier in kürze besprochen:

Aufbauen auf andere Datenbanken: Die Navigation auf XML-Dokumente wurde mithilfe des Datenbankeditors ESCHER realisiert. Dabei wurden der Fingermechanismus und die Navigationssemantik von ESCHER auf XML-Dokumente übertragen. Die Implementierung des Fingermechanismus auf XML-Dokumente kann jedoch direkt auf XML, etwa mit Hilfe der DOM-Schnittstelle, aufbauen. Man kann daher auf schon vorhandene XML-Datenbanken wie Tamino, Natix oder das von der Apache Group entwickelte Datenbanksystem, Xindice, aufsetzen. Da diese Datenbanksysteme auf XML zugeschnitten sind, können XML-Dokumente mit weniger Aufwand erzeugt und nachträglich manipuliert werden. Die Unterstützung von verschiedenen XML-Technologien wie die DOM-Schnittstelle, XPath und XML Schema muß nicht neu implementiert werden. Die Untersuchung der neuen *XML Document Navigation Language* [107] und der Möglichkeit diese Sprache in das Modell zu integrieren bzw. den Fingermechanismus basierend auf dieser Sprache zu implementieren, stellt eine interessante Aufgabe, die in späteren Arbeiten untersucht wird.

Visualisierung für verschiedene Geräte: In der Implementierung wurde zwischen drei verschiedenen Gruppen von Geräten unterschieden: pc, mobile und applet. Der Unterschied zwischen diesen Gerätetypen lag lediglich daran, daß bei pc's XML-Daten direkt übertragen wurden, während für mobile Geräte und Applets die Daten erst in HTML bzw. SVG transformiert wurden. Man kann aber zwischen verschiedenen Geräten feinere Unterscheidungen vornehmen, die Rücksicht auf der Größe, Anzeige und Rechenkapazität nehmen. Durch solche Unterscheidung kann auf der Serverseite VClient-Typen für mehrere Gruppen von Geräten maßgeschneidert implementiert werden.

Implementierung auf anderen Geräten: Das in dieser Arbeit vorgestellte Modell zielt auf beliebige mobile Geräte. Voraussetzung war eine Java virtuelle Maschine (JRE), XML-Unterstützung und die Fähigkeit XML-basierte Visualisierungssprache wie SVG oder HTML zu visualisieren.

Testanwendung auf einem mobilen Gerät wurde auf dem iPAQ 3970 Pocket PC entwickelt. Dieser ist wesentlich leistungsstärker als Mobiltelefone, und verfügt über eine

größere Anzeige. Die Einsatzmöglichkeiten auf kleineren Geräten etwa Mobiltelefone wurde in der Praxis nicht untersucht. Modifikationen an der Systemarchitektur oder der Clientschnittstelle könnten notwendig sein, um diese Geräte effizient zu unterstützen. Auch die Akzeptanz der Benutzer und die Schwierigkeiten der Erlernung des Umgangs mit dieser Methodik auf solchen Geräten könnten zu Modifikationen führen. Es müssen also Testanwendungen entwickelt werden, die den Einsatz dieser Geräte in der Praxis demonstrieren. Felduntersuchungen müssen unternommen werden, um Auskunft über den Umgang der Benutzer mit solchen Anwendungen zu geben.

SOAP und XML-RPC's: Für die Übertragung von XML-Daten zwischen dem Server und dem Client wurde ein Protokoll implementiert, das es dem XML-Parser erlaubte, XML-Ströme wie aus normalen Dateien zu lesen und zu verarbeiten. Zur Zeit der Implementierung waren noch keine Protokolle bzw. Implementierungen für den Austausch von XML-Nachrichten zwischen den Anwendungen vorhanden[1]. SOAP [211] ist ein solches Protokoll, das auch als Teil des Apache XML-Projekts implementiert wurde. Daher ist der Einsatz eines solchen Protokolls statt der eigenen Implementierung sinnvoller, da dieses bezüglich Stabilität, Weiterentwicklung und Verbesserung die Unterstützung einer breiteren Gemeinde genießt.

Die Befehle vom Client an den Server werden nicht mit Hilfe einer XML-Technologie übermittelt, sondern als reiner Text. Zur Zeit der Implementierung gab es da auch keine Implementierungen für XML-RPC's [228]. Aus dem selben obigen Grund wäre eine Umstellung auf diese Technologie sinnvoller.

Unterstützung von anderen Programmiersprachen: Für die Entwicklung des Clients wurde Java eingesetzt, da sie die meiste Unterstützung von XML-Technologien genießt, und auf verschiedenen Maschinentypen zur Verfügung steht. Für kleine mobile Geräte könnte sich herausstellen, daß der Einsatz von Java überdimensioniert ist, und deshalb andere maschinennahen Sprachen wie C oder Assembler eingesetzt werden müssen. Die Implementierung von Clientschnittstellen für andere Sprachen könnte sich daher als sinnvoll erweisen. Auch die Implementierung von Schnittstellen für Skriptsprachen wie Tcl, PHP oder Perl sollte untersucht werden, da diese in der Regel einfacher zu erlernen und zu programmieren sind.

[1] Soweit mir zu diesem Zeitpunkt bekannt war!

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] *Incr Tcl*. URL: http://incrtcl.sourceforge.net/itcl/-index.html.

[2] *JDK 1.3*. URL: http://java.sun.com/j2se/1.3.

[3] *JDOM*. URL: http://www.jdom.org.

[4] *Mobile 3 G*. http://www.mobile3G.com.

[5] *Open Source SVG Toolkit*. URL: http://www.cmis.csiro.au/svg.

[6] *RFI Whitepaper*. http://www.anycom.com/about/whitepaper/-RFIWhitepaper.pdf.

[7] *Sametime*. URL: http://www.lotus.com/home.nsf/welcome/-sametime.

[8] *Tcl*. URL: http://www.scriptics.com.

[9] *TclDP*. URL: http://www.cs.cornell.edu/zeno/projects/-tcldp.

[10] *TclX*. URL: http://www.neosoft.com/tclx.

[11] *tDOM*. URL: http://www.tdom.org.

[12] *UMTS Forum*. http://www.umts-forum.com.

[13] *WAP Forum*. http://wap-forum.org.

[14] *xalan*. URL: http://xml.apache.org/xalan-j/index.html.

[15] *xerces*. URL: http://xml.apache.org/xerces2-j/index.html.

[16] *Apache Xindice*. URL: http://xml.apache.org/xindice/index.-html, 2002.

[17] ABITEBOUL, S. (SERGE), PETER BUNEMAN, and DAN SUCIU: *Data on the Web: from Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1999.

[18] AHMAD, MORAD: *TclDB: Entwurf und Implementierung einer Skriptsprache für den Datenbank-Editor ESCHER*. Diplomarbeit, GhK-FB17, 1998.

[19] AHMAD, MORAD, JENS THAMM, and LUTZ WEGNER: *Rapid application development for Web-based collaboration*. In YANCHUN ZHANG, MAREK RUSINKIEWICZ, YAHIKO KAMBAYASHI (editor): *Proceedings of the Second International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'99)*, pages 112–123, Wollongong, Australia, march 1999. Springer, Singapore.

[20] AKSCYN, R. M., D.L. MCCRACKEN, and E.A YODER: *KMS: A distributed hypermedia system for managing knowledge in organizations*. Communications of the ACM, ; ACM CR 8908–575, 31(7):820–835, July 1988.

[21] ALIMADHI, FERDINAND: *Mobile Internet: Wireless access to Web-based interfaces of legacy simulations*. Master's thesis, University of Amsterdam, Faculty of Sience, 2002.

[22] ANDERSEN, FLEMMING, VOLKER LINNEMANN, PETER PISTOR, and NORBERT SÜDKAMP: *Advanced information management prototype — User manual for the online interface of the Heidelberg Data Base Language (HDBL) prototype implementation*. Technical Report TN 86.01, IBM Germany, Heidelberg Scientific Center, March 1987. Release 1.3.

[23] APPELT, W.: *WWW based collaboration with the BSCW system*. In WERNER, BOB (editor): *Proceedings of SOFSEM'99*, pages 66–78, Milovy (Czech Republic), December 1999. Springer Lecture Notes in Computer Science 1725.

[24] APPELT, W.: *What groupware functionality do users really use? analysis of the usage of the BSCW system*. In WERNER, BOB (editor): *Proceedings of the Ninth Euromicro Workshop on Parallel and Distributed Processing*, pages 337–344, Los Alamitos, California, February 7–9 2001. IEEE Computer Society.

[25] ASTRAHAN, M. M., M. W. BLASGEN, D. D. CHAMBERLIN, and OTHERS: *System R: Relational approach to database management*. ACM Transactions on Database Systems, 1(2):97–137, 1976.

[26] ATKINSON, M. P., F. BANCILHON, DEWITT D. J., and OTHERS: *The object-oriented database system manifesto*. In KIM, W., J.-M. NICOLAS, and S. NISHIO (editors): *Proc. DOOD'89*, pages 223–240. Elsevier (North-Holland), 1990.

[27] BALLARD, D. H., M. M. HAYHOE, P. K. POOK, and R. P. N. RAO: *Deictic codes for the embodiment of cognition*. Behavioral and Brain Sciences, 20(4):723–767, 1997.

[28] BASU, JULIE and JULIE SHOME: *Essential SQLJ Programming: The Complete Guide to the ANSI Standard for Embedded SQL in Java*. John Wiley, July 1999.

[29] BEGOLE, JAMES, MARY BETH ROSSON, and CLIFFORD A. SHAFFER: *Flexible collaboration transparency: Supporting worker independence in replicated application-sharing systems*. ACM SIGCHI Bulletin, 6(2):95–132, 1999.

[30] BENFORD, S., J. BOWERS, L. E. FAHLEN, J. MARIANI, and T. RODDEN: *Supporting cooperative work in virtual environments*. The Computer Journal, 37(8), 1994.

[31] BENN, W. und I. GRINGER: *Zugriff auf Datenbanken über das World Wide Web*. Informatik-Spektrum, Seiten 1–8, 1998.

[32] BENTLEY, R., T. HORSTMANN, K. SIKKEL und J. TREVOR: *Supporting Collaborative Information Sharing with the WWW: The BSCW Shared Workspace System*. In: O'REILLY AND ASSOCIATES und WEB CONSORTIUM (W3C) (Herausgeber): *World Wide Web Journal: The Fourth International WWW Conference Proceedings*, Seiten 63–74, 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, 1995. O'Reilly & Associates, Inc.

[33] BERNERS-LEE, T., L. MASINTER, and M. MCCAHILL: *Uniform Resource Locators, RFC 1738*. URL: `http://www.w3.org/Addressing/-rfc1738.txt`, December 1994.

[34] BERNSTEIN, PHILIP, V. HADZILACOS, and N. GOODMAN: *Concurrency Control and Recovery in Database Systems*, chapter 7-8, pages 217–304. Addison-Wesley, 1987.

[35] BILIRIS, A.: *An efficient database storage structure for large dynamic objects*. In GOLSHANI, F. (editor): *Proc. ICDE'92*, pages 301–308. IEEE Computer Science Press, 1992.

[36] BILIRIS, A.: *The performance of three database storage structures for managing large objects*. In STONEBRAKER, M. (editor): *Proc. SIGMOD'92*, ACM SIGMOD Record 21(2). ACM Press, 1992.

[37] BIRMAN, KENNETH P.: *ISIS: A system for fault-tolerant distributed computing*. Technical Report TR86-744, Cornell University, Computer Science Department, April 1986.

[38] BORGHOFF, U. M. und J. H. SCHLICHTER: *Rechnergestützte Gruppenarbeit, Eine Einführung in verteilte Anwendungen*. Springer-Verlag, Berlin, Heidelberg, New York, 1995.

[39] BOTTERWECK, GÖTZ: *Einsatz von XML und WAP zur Realisierung strukturierter, ubiquitärer Informationsdienste*. Diplomarbeit, Universtität Koblenz-Landau, Abteilung Koblenz, Fachbereich 4: Informatik, September 2000.

[40] BRODIE, M. L.: *On the development of data models*. In BRODIE, M. L., J. MYLOPOULOS, and J. W. SCHMIDT (editors): *On Conceptual Modelling*, pages 19–48. Springer, 1984.

[41] BROWN, MARK R.: *FastCGI Specification*. Open Market, Inc., 1.0 edition, 1996.

[42] BÜRGER, MARTIN: *Unterstützung von Awareness bei der Gruppenarbeit mit gemeinsamen Arbeitsbereichen*. Herbert Utz Verlag, München, 1999.

[43] CAREY, M. J., D. J. DEWITT, J. F. NAUGHTON, and OTHERS: *Shoring up persistent applications*. In SNODEGRASS, R. T. and M. WINSLETT (editors): *Proc. SIGMOD'94*, ACM SIGMOD Record 23(2), pages 383–394. ACM Press, 1994.

[44] CATTELL, R. G. G.: *Object Data Managment*. Addison-Wesley, 1991.

[45] CATTELL, R. G. G. (editor): *The Object Database Standard: ODMG-93, Release 1.1*. Morgan Kaufmann, 1994.

[46] CATTELL, R. G. G. (editor): *The Object Database Standard: ODMG-93, Release 1.2*. Morgan Kaufmann, 1996.

[47] CATTELL, R. G. G. (editor): *The Object Database Standard: ODMG-2.0*. Morgan Kaufmann, 1997.

[48] CHOCKLER, GREGORY, IDIT KEIDAR, and ROMAN VITENBERG: *Group communication specifications: a comprehensive study*. ACM Computing Surveys, 33(4):427–469, 2001.

[49] CODD, E. F.: *A relational model of data for large shared data banks*. Communications of the ACM, 13(6):377–387, 1970.

[50] DADAM, P., K. KUESPERT, F**?**. ANDERSEN, and OTHERS: *A DBMS prototype to support extended NF^2 relations: An integrated view on flat tables and hierarchies*. In ZANIOLO, C. [230], pages 356–367.

[51] DADAM, P., K. KUESPERT, F ANDERSEN, and OTHERS: *A DBMS prototype to support extended NF^2 relations: An integrated view on flat tables and hierarchies*. In ZANIOLO, C. [230], pages 356–367.

[52] DADAM, PETER: *Advanced Information Management (AIM): Research in extended nested relations*. Database Engineering, 7:119–129, 1988.

[53] DAYAL, U., P. M. D. GRAY, and S. NISHIO (editors): *Proceedings of 21st International Conference on Very Large Data Bases, September 11–15, 1995, Zurich, Switzerland*. Morgan Kaufmann, 1995.

[54] DEPPISCH, U., V. OBERMEIT, H.-B. PAUL und OTHERS: *Ein Subsystem zur stabilen Speicherung versionsbehafteter, hierarchisch strukturierter Tupel*. In: BLASER, A. und P. PISTOR (Herausgeber): *Datenbanksysteme in Büro, Technik und Wissenschaft, GI-Fachtagung, Karlsruhe, 20.–22. März 1985, Proceedings*, Band 94 der Reihe *Informatik Fachberichte*, Seiten 421–440. Springer, 1985.

[55] DEPPISCH, U., H.-B. PAUL, and H.-J. SCHEK: *A storage system for complex objects*. In DITTRICH, K. R. and U. DAYAL [58], pages 183–195.

[56] DEWITT, DAVID: *Database systems: Road kill on the information superhighway?* In DAYAL, U. et al. [53]. Invited talk.

[57] DITTRICH, K.: *Object-oriented database systems: The notions and the issues*. In DITTRICH, K. R. and U. DAYAL [58], pages 2–4.

[58] DITTRICH, K. R. and U. DAYAL (editors): *International Workshop on Object-Oriented Database Systems, Pacific Grove, September 23–26, 1986*. IEEE Computer Science Press, 1986.

[59] DIX, ALAN: *Challenges for cooperative work on the web: An analytical approach*. Computer Supported Cooperative Work, 6(2/3):135–156, 1997.

[60] DOURISH, PAUL and VICTORIA BELLOTTI: *Awareness and coordination in shared workspaces*. In *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, The Power of Simple Shared Workspaces, pages 107–114, 1992.

[61] DURAND, DAVID G.: *Palimpsest: A data model for change oriented revision control*. In *Proceedings of ACM Hypertext'93 – Posters*, page 29, 1993.

[62] ELLIS, C., S. J. GIBBS, and G. REIN: *Design and use of a group editor*. MCC, Austin, Texas, 1988.

[63] ELLIS, C. A. and SIMON J. GIBBS: *Concurrency control in groupware systems*. In CLIFFORD, JAMES, BRUCE G. LINDSAY, and DAVID MAIER (editors): *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 399–407, Portland, Oregon, 31 May–2 June 1989.

[64] ELLIS, C. A., S. J. GIBBS, and G. L. REIN: *Groupware: Some issues and experiences*. In *Communications of the ACM*, volume 34, pages 38–58. ACM, 1991.

[65] ELLIS, CLARENCE (SKIP) and JACQUES WAINER: *A conceptual model of groupware*. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, Models of Cooperative Work, pages 79–88, 1994.

[66] ENGELBART, DOUGLAS C.: *A conceptual framework for augmenting man's intellect*. In HOWERTON, PAUL W. and DAVID C. WEEKS (editors): *Vistas in Information Handling*, volume 1, pages 1–29. Spartan Books, Washington DC, 1963. Quoted in Chapter 3, [Rheingold 91].

[67] ESWARAN, K. P., J. N. GRAY, R. A. LORIE, and I. L. TRAIGER: *The notions of concurrency and predicate locks in a data base system*. Communications of the ACM, 19(11):624–633, 1976.

[68] FEKETE, A. and I. KEIDER: *Specifying and using a partitionable group communication service*. ACM Transactions on Computer Systems (TOCS), 19(2):171–216, 2001.

[69] FIEBIG, THORSTEN, CARL-CHRISTIAN KANNE und GUIDO MOERKOTTE: *Natix - ein natives XML-DBMS*. Datenbank Spektrum, 1(1):5–13, September 2001.

[70] FISH, ROBERT S., ROBERT E. KRAUT, and MARY D. P. LELAND: *Quilt: A collaborative tool for cooperative writing*. In *Proceedings of the Conference on Office Automation Systems*, Collaborative Work, pages 30–37, 1988.

[71] FITZPATRICK, GERALDINE: *The Locales Framework: Understanding and Designing for Cooperative Work*. PhD thesis, University of Queensland, 1998.

[72] FITZPATRICK, GERALDINE, WILLIAM J. TOLONE, and SIMON M. KAPLAN: *Work, locales and distributed social worlds*. In *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work*, Distributed Social Worlds, pages 1–16, 1995.

[73] FOCHLER, K., P. PERC und J. UNGERMANN: *Lotus Domino 4.6: Internet- und Intranetlösungen mit dem Lotus Domino Server.* Addison-Wesley, 1998.

[74] FRANKLIN, MICHAEL J., MICHAEL J. CAREY und MIRON LIVNY: *Transactional client-server cache consistency: alternatives and performance*. ACM Transactions on Database Systems, 22(3):315–363, September 1997.

[75] FRANZ, MICHAEL: *The Java virtual machine — a passing fad?* IEEE Software, 15(6):26–29, 1998.

[76] FRÖHLICH, STEFAN: *Entwicklung eines Servers für strukturierte Objekträume als Basis eines Mobilfunkdienstes für synchrone Gruppenarbeit mittels Java und XML*. Diplomarbeit, GhK-FB17, 2001.

[77] FUCHS, LUDWIG, UTA PANKOKE-BABATZ und WOLFGANG PRINZ: *Supporting Cooperative Awareness with Local Event Mechanisms: The GroupDesk System*. In: *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work*, CSCW Mechanisms II, Seiten 247–262, 1995.

[78] GARFINKEL, D., P. GUST, M. LEMON und S. LOWDER: *The SharedX multi-user interface user's guide, version 2.0*. Hewlett-Packard Laboratories, Palo Alto, California, 1989.

[79] GILLNER, R., L. WEGNER, and CH. ZIRKELBACH: *Collaborative project management with a Web-based database editor*. In GOLUBCHIK, LEANA and VASSILIS J. TSOTRAS (editors): *Proc. 5th Int. Workshop Multimedia Information Systems (MIS'99)*, pages 72–79, Indian Wells, California, October 1999.

[80] GOLDFARB, CHARLES F. and YURI RUBINSKY: *The SGML Handbook*. Clarendon Press, Oxford, UK, 1990.

[81] GOODGER, BEN, IAN HICKSON, DAVID HYATT, and CHRIS WATERSON: *XML User Interface Language (XUL) 1.0*. URL: `http://www.mozilla.org/projects/xul/xul.html`, 2001.

[82] GOSLING, JAMES, BILL JOY, and GUY STEELE: *The Java^{TM} Language Specification*. Sun Microsystems, 1.0 edition, August 1996. Auch als Buch mit demselben Titel in der Reihe '*The Java Series*' von Addison-Wesley.

[83] GOSLING, JAMES, BILL JOY, GUY L. STEELE, and GILAD BRACHA: *The Java Language Specification*. Java series. Addison-Wesley, Reading, MA, USA, second edition, 2000.

[84] GRAY, JIM and ANDREAS REUTER: *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc., 1993.

[85] GRAY, J. N., R. A. LORIE, G. R. PUTZOLO, and I. L. TRAIGER: *Granularity of locks and degrees of consistency in a shared data base*. IFIP Working Conf. on Modeling of Data Base Management Sys., January 1976. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan Kaufmann, San Mateo, CA, 1988.

[86] GREENBERG, SAUL: *Peepholes: Low cost awareness of one's community*. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, volume 2 of *SHORT PAPERS: Supporting Awareness of Others in Groupware (Short Papers Suite)*, pages 206–207, 1996.

[87] GREENBERG, SAUL, CARL GUTWIN, and ANDY COCKBURN: *Using distortion-oriented displays to support workspace awareness*. In SASSE, A., R. J. CUNNINGHAM, and R. WINDER (editors): *Proc. HCI — People and Computers XI*, pages 299–314. Springer-Verlag, 20–23 August 1996. revised.

[88] GREENBERG, SAUL and DAVID MARWOOD: *Real time groupware as a distributed system: Concurrency control and its effect on the interface*. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, Technologies for Sharing I, pages 207–217, 1994.

[89] GREENBERG, SAUL and MARK ROSEMAN: *GroupKit – A groupware toolkit*. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, Formal Video Program: Prototypes and Enabling Technologies, page 9, 1994.

[90] GREENBERG, SOUL and MARK ROSMAN: *Groupware toolkits for synchronous work*. Department of Compute Science, Univertisty of Calgary, 1996.

[91] GREIF, IRENE and SUNIL SARIN: *Data sharing in group work*. ACM Transactions on Office Information Systems, 5(2):187–211, April 1987. Special Issue on Computer-Supported Cooperative Work.

[92] GREIF, IRENE, ROBERT SELIGER, and WILLIAM WEIHL (editors): *Atomic Data Abstractions in a Distributed Collaborative Editing System (Extended Abstract)*. ACM, 13th Annual ACM Symp. on the Principles of Programming Languages, January 1986.

[93] GROSS, T.: *CSCW3: Transparenz- und Kooperationsunterstützung für das WWW*. In: *Groupware und Organisatorische Innovation, Tagungsband der Deutschen Computer Supported Cooperative Work Tagung - DCSCW'98*, Seiten 37–50, Dortmund, Germany, Sept 1998. Springer-Verlag, Berlin.

[94] GUTWIN, CARL und SAUL GREENBERG: *Focus and Awareness in Groupware*. In: *Proceedings of ACM CSCW'98 Conference on Computer-Supported Cooperative Work*, Videos, Seiten 425–426, 1998.

[95] GUTWIN, CARL, SAUL GREENBERG und MARK ROSEMAN: *Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation*. In: *Proceedings of the HCI'96 Conference on People and Computers XI*, Computer-Supported Cooperative Work, Seiten 281–298, 1996.

[96] HAMILTON, GRAHAM, RICK CATTELL, and MAYDENE FISHER: *JDBC Database Access with Java: A Tutorial and Annotated Reference*. Addison-Wesley, 1997.

[97] HAN, RICHARD, VERONIQUE PERRET, and MAHMOUD NAGHSHINEH: *Websplitter: A unified XML framework for multi-device collaborative web browsing*. In *Proceedings of ACM CSCW'00 Conference on Computer-Supported Cooperative Work*, Mobility, pages 221–230, 2000.

[98] HAN, RICHARD, VERONIQUE PERRET, and MAHMOUD NAGHSHINEH: *Websplitter: A unified XML framework for multi-device collaborative web browsing*. In *Proceedings of ACM CSCW'00 Conference on Computer-Supported Cooperative Work*, Mobility, pages 221–230, 2000.

[99] HAROLD, ELLIOTTE RUSTY: *XML Bible*. I D G Books Worldwide, Indianapolis, IN, USA, 1999.

[100] HARRISON, MARK und MICHAEL MCLENNAN: *Effektive Tcl/Tk programieren*. Addison-Weseley-Longman, 1998.

[101] HERRMANN, U., P. DADAM, K. KÜSPERT, E. A. ROMAN und G. SCHLAGETER: *A Lock Technique for Disjoint and Non-Disjoint Complex Objects*. In: BANCILHON, F., C. THANOS und D. TSICHRITZIS (Herausgeber): *Proceedings of the International Conference on Extending Database Technology : Advances in Database Technology*, Band 416 der Reihe *LNCS*, Seiten 219–237, Berlin, März 1990. Springer.

[102] HEUER, ANDREAS: *Objektorientierte Datenbanken*. Addison-Wesley, 1992.

[103] HÄRDER, THEO und ERHARD RAHM: *Datenbanksysteme: Konzepte und Techniken der Implementierung*. Springer, 1999.

[104] INC., INSIGNIA SOLUTIONS: *Insignia Jeode documentation*. URL: `http://www.insignia.com/content/products/-jeodeRuntime.shtml`, 2003.

[105] INTERNATIONAL ORGANIZATION OF STANDADIZATION: *Database Language SQL*. ISO Copyright Office, Genf, 1987. Document ISO/IEC 9075: 1987.

[106] INTERNATIONAL ORGANIZATION OF STANDADIZATION: *Information Technology – Database Language – SQL – Technical Corrigendum 2*. ISO Copyright Office, Genf, 1996. Document ISO/IEC 9075:1992/Cor. 2.

[107] ITO, NAOKO and KAZUHISA MANABE: *XML Document Navigation Language*. URL: `http://www.w3.org/TR/xdnl`, March 2000.

[108] JEFFERIES, P. and I. CONSTABLE: *Using BSCW in Learning & Teaching*. Educational Technology & Society, 3(3), July 2000.

[109] JING, JIN, ABDELSALAM SUMI HELAL, and AHMED ELMAGARMID: *Client-server computing in mobile environments*. ACM Computing Surveys (CSUR), 31(2):117–157, 1999.

[110] JOHANSEN, R.: *Groupware: Computer Support for Business Teams*. The Free Press, Macmillan Inc., New York, 1988.

[111] JOHANSEN, ROBERT: *Teams for tomorrow*. In IEEE COMPUTER SOCIETY PRESS, LOS ALAMITOS, CALIFORNIA (editor): *24th Annual Hawaii Intl. Conf. on System Sciences*, pages 520–534, 1991.

[112] JOSEPH, ANTHONY D. and M. FRANS KAASHOEK: *Building reliable mobile-aware applications using the rover toolkit*. Wireless Networks, 3(5):405–419, 1997.

[113] KALUS, CHRISTIAN and PETER DADAM: *Flexible relations — operational support of variant relational structures*. In DAYAL, U. et al. [53], pages 539–550.

[114] KELTER, UDO: *The queue protocol: a deadlock-free, homogeneous, non-two-phase locking protocol*. In *Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 142–151. ACM Press, 1988.

[115] KERNIGHAN, BRIAN: *What have you learned today?* IEEE Software, 16(2):66–68, March/April 1999. Interview.

[116] KHARE, ROHIT and ADAM RIFKIN: *XML: Modeling data and metadata*. In *Proceedings of ACM CSCW'98 Conference on Computer-Supported Cooperative Work*, Tutorials, page 430, 1998.

[117] KIM, W. (editor): *Modern Database Systems: The Object Model, Interoperability, and Beyond*. Addison-Wesley, 1995.

[118] KNISTER, J. and A. PRAKASH: *Distedit: a distributed toolkit for supporting multiple group editors*. In BIKSON, T. (editor): *Proc of 3rd Int. Conf. on Computer-Supported Cooperative Work*, pages 343 – 355, Los Angeles, Oct 1990.

[119] KORTH, HENRY F.: *The double life of the transaction abstraction: Fundamental principle and evolving system concept*. In DAYAL, U. et al. [53], pages 2–6.

[120] KOTZ-DITTRICH, ANGELIKA and KLAUS R. DITTRICH: *Where object-oriented DBMSs should do better: A critique based on early experiences*. In KIM, W. [117], chapter 12, pages 238–254.

[121] KRAUSS, MARTIN: *Multimedialer Museumsführer auf Basis des Compaq iPAQ Pocket PC*. Diplomarbeit, Fachhochschule München, Fachbereich Wirtschaftsingenieurwesen, Januar 2002.

[122] KRÄMER, BERND and LUTZ WEGNER: *Beyond the whiteboard: Synchronous collaboration in shared object spaces*. In IEEE CS PRESS, LOS ALAMITOS, CALIF. (editor): *7th IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 131–136, Cape Town, South Africa, Dec. 20-22 1999.

[123] KUNII, T. L. (editor): *Proceedings of the 1st IFIP Working Conference on Visual Database Systems, Tokyo, 1989*. Elsevier (North-Holland), 1989.

[124] LAMPORT, LESLIE: *Time, clocks, and the ordering of events in a distributed system*. Communications of the ACM, 21(7):558–565, 1978.

[125] LEWIS, B. T. and J. D. HODGES: *Shared books: collaborative publication management for an office information system*. In ALLEN, R. B. (editor): *Proc of Conf. on Office Information Systems*, pages 197 – 204, Palo Alto, Mar 1988.

[126] LEWIS, TED G.: *Java holy war '98*. Computer, 31(3):126–128, March 1998.

[127] LITIU, RADU and ATUL PRAKASH: *Developing adaptive groupware applications using a mobile component framework*. In *Proceedings of the ACM 2000 Conference on Computer Supported Cooperative Work (CSCW-00)*, pages 107–116, N. Y., December 2–6 2000. ACM Press.

[128] LOESER, HENRIK: *Datenbankanbindung an das WWW — Techniken, Tools und Trends*. In: DITTRICH, KLAUS R. und ANDREAS GEPPERT (Herausgeber): *Proc. BTW'97*, Informatik Aktuell, Seiten 83–99. Springer, 1997.

[129] LOTUS DEVELOPMENT CORP.: *Lotus Notes User Guide*. Lotus Development Corp., Cambridge, Mass., 1989.

[130] LYNGBAEK, PETER and WILLIAM KENT: *A data modelling methodology for the design and implementation of information systems*. In DITTRICH, K. R. and U. DAYAL [58].

[131] MAKINOUCHI, AKIFUMI: *A consideration on normal form of not-necessarily-normalized relation in the relational data model*. In *Proc. VLDB'77*, pages 447–453. IEEE Computer Science Press, 1977.

[132] MALAIKA, SUSAN: *Resistance is futile: The web will assimilate your database*. Data Engineering Bulletin, 21(2):4–13, June 1998.

[133] MANOLESCU, IOANA, DANIELA FLORESCU, DONALD KOSSMANN, FLORIAN XHUMARI, and DAN OLTEANU: *Agora: Living with XML and relational*. In ABBADI, AMR EL, MICHAEL L. BRODIE, SHARMA CHAKRAVARTHY,

UMESHWAR DAYAL, NABIL KAMEL, GUNTER SCHLAGETER, and KYU-YOUNG WHANG (editors): *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 623–626. Morgan Kaufmann, 2000.

[134] MARIANI, JOHN A.: *SISCO: Providing a cooperation filter for a shared information space*. In *GROUP'97: International Conference on Supporting Group Work*, Group Awareness, pages 376–384, 1997.

[135] MCCOOL, ROB: *The Common Gateway Interface*. NCSA, 1.1 edition, 1994.

[136] MCLAUGHLIN, BRETT: *Java and XML*. O'Reilly, June 2000.

[137] MELTON, JIM (editor): *Database Language SQL (SQL3)*. ANSI, March 1995. ISO-ANSI Working Draft X3H2-95-084/DBL:YOW-004.

[138] MICROSOFT CORPORATION: *Microsoft NetMeeting 2.0: Overview and frequently asked questions*. Technical report, Microsoft Corporation, July 1997.

[139] MOBILE STREAMS: *Yes 2 3g - white paper*. Technical report, Mobile Streams, February 2001.

[140] MYERS, BRAD A., HERB STIEL, and ROBERT GARGIULO: *Collaboration using multiple PDAs connected to a PC*. In *Proceedings of the ACM 1998 conference on Computer supported cooperative work*, pages 285–294. ACM Press, 1998.

[141] NAVATHE, SHAMKANT B.: *Evolution of data modeling for databases*. Communications of the ACM, 35(9):112–123, 1992.

[142] NOBLE, BRIAN D., M. SATYANARAYANAN, DUSHYANTH NARAYANAN, JAMES ERIC TILTON, JASON FLINN, and KEVIN R. WALKER: *Agile application-aware adaptation for mobility*. In *Sixteen ACM Symposium on Operating Systems Principles*, pages 276–287, Saint Malo, France, 1997.

[143] OPPEL, KARIN: *Tamino: The Information Server for Electronic Business*. Technical report, Software AG, October 2001.

[144] OUSTERHOUT, JOHN K.: *Tcl und Tk. Entwicklung grafischer Benutzerschnittstellen für das X Window System*. Addison-Wesley, 1995.

[145] OUSTERHOUT, JOHN K.: *Scripting: Higher-level programming for the 21st century*. Computer, 31(3):23–30, March 1998.

[146] PALMER, CHRISTOPHER R. and GORDON V. CORMACK: *Operation transforms for a distributed shared spreadsheet*. In *Proceedings of ACM CSCW'98 Conference on Computer-Supported Cooperative Work*, Concurrency and Consistency, pages 69–78, 1998.

[147] PAUL, H.-B.: *DAS Datenbank-Kernsystem für Standard- und Nicht-Standard-Anwendungen — Architekur, Implementierung, Anwendungen*. Dissertation, TH Darmstadt, 1988.

[148] PAUL, M.: *Typerweiterungen im eNF^2-Datenmodell*. Dissertation, Universität Gh Kassel, August 1994.

[149] PAUL, MANFRED, S. THELEMANN und L. WEGNER: *User Interface Techniques based on the Non-First Normal-Form Data Model*. Technischer Bericht 14/94, FB Mathematik/Informatik, Univ. Gh Kassel, Dec. 1994. auch: GI-Fachgruppentagung "Benutzungsschnittstellen für Datenbanken", Kassel, Germ any, 17.–18.März 1994, Datenbankrundbrief 13, Seiten 55–57.

[150] PRAKASH, ATUL und MICHAEL J. KNISTER: *Undoing Actions in Collaborative Work*. In: *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, Consistency in Collaborative Systems, Seiten 273–280, 1992.

[151] PRAKASH, ATUL and HYONG SOP SHIM: *DistView: Support for building efficient collaborative applications using replicated active objects*. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, Collaborative Editing and Reviewing, pages 153–164, 1994.

[152] PRAKASH, ATUL, HYONG SOP SHIM, and JANG HO LEE: *Data Managment Issues and Trade-Offs in CSCW Systems*. IEEE Transactions on Knowledge and Data Engineering, 11(1), 1999.

[153] PRASUN DEWAN, HONG HAI SHEN: *Flexible meta access-control for collaborative applications*. In PRESS, ACM (editor): *Proc. Conf. Computer Supported Cooperative Work*, pages 247–256, Seattle, Nov 1998.

[154] QUIX, CHRISTOPH, MAREIKE SCHOOP, and MANFRED JEUSFELD: *Business data management for business-to-business electronic commerce*. ACM SIGMOD Record, 31(1):49–54, 2002.

[155] RAY, ERIK T.: *Einführung in XML*. O'Reilly & Associates, Inc., 2001.

[156] RIPPER, THIEMO: *PalmPilot Spielzeug oder unverzichtbares Werkzeug*. Technischer Bericht, Fachbereich 1 - Wirtschaftsinformatik Technische Universität Darmstadt, 2002.

[157] RODDEN, TOM: *Populating the application: A model of awareness for cooperative applications*. In ACKERMANN, MARK S. (editor): *Proceedings of ACM CSCW'96 Conference on Computer-Supported Cooperative Work*, Learning from Space and Place, pages 87–96, 1996.

[158] ROSEMAN, MARK and SAUL GREENBERG: *GroupKit: A groupware toolkit for building real-time conferencing applications*. In *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, Building Real-Time Groupware, pages 43–50, 1992.

[159] ROSEMAN, MARK and SAUL GREENBERG: *Building flexible groupware through open protocols*. In *Conference on Organizational Computing Systems*, Groupware Architectures, pages 279–288, 1993.

[160] ROSEMAN, MARK and SAUL GREENBERG: *Building real-time groupware with GroupKit, a groupware toolkit*. ACM Transactions on Computer-Human Interaction, 3(1):66–106, 1996.

[161] RÜDEBUSCH, T.: *CSCW: Generische Unterstützung von Teamarbeit in verteilten DV-Systemen*. Deutscher Universitäts Verlag, Wiesbaden, 1993.

[162] SAAKE, GUNTER, VOLKER LINNEMANN, PETER PISTOR, and LUTZ MICHAEL WEGNER: *Sorting, grouping and duplicate elimination in the Advanced Information Management Prototype*. In APERS, P. M. G. and G. WIEDERHOLD (editors): *Proc. VLDB'89*, pages 307–316. Morgan Kaufmann, 1989.

[163] SANDOR, OVIDIU, CHRISTIAN BOGDAN, and JOHN BOWERS: *Aether: An awareness engine for CSCW*. In *Proceedings of the Fifth European Conference on Computer-Supported Cooperative Work*, Objects, Spaces and Bodies, pages 221–236, 1997.

[164] SCHEK, HANS-JORG, GERHARD WEIKUM, and HAIYAN YE: *Towards a unified theory of concurrency control and recovery*. In BEERI, CATRIEL (editor): *Proceedings of the 12th Symposium on Principles of Database Systems*, pages 300–311, New York, NY, USA, May 1993. ACM Press.

[165] SCHEK, H.-J. and M. H. SCOLL: *The relational model with relation-valued attributes*. Information Systems, 11(2):137–147, 1986.

[166] SCHMIDT, CHRISTIAN: *Client-seitige Unterstützung eines Mobilfunkdienstes für synchrone Gruppenarbeit mittels Java und XML*. Diplomarbeit, GhK-FB17, 2001.

[167] SCHMIDT, HEINZ W. and LUTZ M. WEGNER: *Shared xml documents in service centers of the future*. In *Proceeding Of The* 1^{st} *International Conference on Web Information Systems Engineering*, pages 102–108, Hong Kong, Juni 2000.

[168] SEGALL, BILL and DAVID ARNOLD: *Elvin has left the building: A publish/subscribe notification service with quenching*. In *Proceedings AUUG97*, Brisbane, Australia, September 1997.

[169] SHEN, H. and P. DEWAN: *Access control for collaborative environments*. In TURNER, J. and R. KRAUT (editors): *CSCW'92: Sharing Perspectives*, pages 51 – 58, Toronto, Nov 1992.

[170] SILBERSCHATZ, A., H. F. KORTH, and S. SUDARSHAN: *Database System Concepts*. McGraw-Hill, New York, 3 edition, 1997.

[171] SILBERSCHATZ, ABRAHAM and ZVI KEDEM: *Consistency in hierarchical database systems*. Journal of the ACM (JACM), 27(1):72–80, 1980.

[172] SRINIVASAN, RAJ: *RPC: Remote Procedure Call Protocol Specification, Version 2, RFC 1831*. URL: `http://www.ietf.org/rfc/rfc1831.txt`, August 1995.

[173] STEEL, T. B.: *Interim report of the ANSI-SPARC study group*. Technical Report 2, American National Standards Committee (Washington DC)., 1975.

[174] STEPHANIE TEUFEL, CHRISTIAN SAUTER, THOMAS MÜHLERR und KURT BAUKNECHT: *Computer Unterstützung für die Gruppenarbeit.* Addison-Wesley, 1995.

[175] ST. LAURENT, SIMON: *Dynamic HTML: a primer*. MIS Press, P. O. Box 5277, Portland, OR 97208-5277, USA, Tel: (503) 282-5215, 1997.

[176] STONEBRAKER, MICHAEL: *Object-Relational DBMSs: The Next Great Wave*. Morgan Kaufmann, 1996.

[177] STROM, BANAVAR, MILLER, PRAKASH, and WARD: *Concurrency control and view notification algorithms for collaborative replicated objects*. IEEETC: IEEE Transactions on Computers, 47, 1998.

[178] SU, CHI-JIUN and LEANDROS TASSIULAS: *Joint broadcast scheduling and user's cache management for efficient information delivery*. In *Proceedings of the fourth annual ACM/IEEE international conference on Mobile computing and networking*, pages 33–42. ACM Press, 1998.

[179] SULEIMAN, MAHER, MICHELE CART, and JEAN FERRIE: *Serialization of concurrent operations in a distributed collaborative environment*. In *GROUP'97: International Conference on Supporting Group Work*, Algorithms and Formalizations, pages 435–445, 1997.

[180] SUN MICROSYSTEMS: *Java RMI*. URL: `http://java.sun.com/-products/jdk/rmi`.

[181] SUN MICROSYSTEMS: *PersonalJava 1.0 Specification*. URL: `http://-java.sun.com/products/personaljava/spec-1-0-1/-personalJavaSpec1.0.1.html`, 1997.

[182] SUN MICROSYSTEMS: *J2ME Connected, Limited Device Configuration*. URL: `http://jcp.org/aboutJava/communityprocess-/final/jsr030/index.html`, March 2000.

[183] SUN MICROSYSTEMS: *J2ME Connected Device Configuration*. URL: `http://java.sun.com/products/cdc`, March 2002.

[184] SUN MICROSYSTEMS: *Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices*. URL: `http://java.sun.com/products/-cldc/wp/KVMwp.pdf`, March 2002.

[185] SUN MICROSYSTEMS: *Mobile Information Mobile Profile for Java 2 Micro Edition, version 2.0*. URL: `http://java.sun.com/products/midp`, March 2002.

[186] TAYLOR, ROBERT W. TAYLOR and RANDALL L. FRANK: *CODASYL database management systems*. ACM Computing Surveys, 8(1):67–103, March 1976.

[187] TEEGE, GUNNAR: *Individuelle Groupware Gestaltung durch den Endbenutzer*. DUV, 1998.

[188] TER HOFTE, G. HENRI: *Working apart together : Foundations for component groupware*. Number 001 in *Telematica Instituut Fundamental Research Series*. Telematica Instituut, Enschede, the Netherlands, publications@telin.nl, P.O. Box 589, 7500 AN Enschede, the Netherlands, 1998.

[189] THAMM, J. and L. WEGNER: *What you see is what you store: Database-driven interfaces*. In IOANNIDIS, YANNIS and WOLFGANG KLAS (editors): *Visual Database Systems 4 (VDB4)*, pages 69–84. Chapman & Hall, 1998.

[190] THAMM, JENS: *A Database Approach to GUI Management*. In: SCHOLL, MARC H., HOLGER RIEDEL, TORSTEN GRUST und DIETER GLUCHE (Herausgeber): *10. Workshop „Grundlagen von Datenbanken“*, Nummer 63 in *Konstanzer Schriften in Mathematik und Informatik*, Seiten 125–129, Mai 1998.

[191] THAMM, JENS: *Visualisierungsverfahren zur Interaktion mit objekt-relationalen Datenbanken*. Dissertation, Universität Gh Kassel, August 1999.

[192] THAMM, JENS and LUTZ WEGNER: *What you see is what you store: Database-driven interfaces*. In IOANNIDIS, YANNIS and WOLFGANG KLAS (editors): *Proc. 4th IFIP WG 2.6 Working Conf. on Visual Database Systems (VDB4), L'Aquila, Italy, May 27-29, 1998*, London SE1 8HN, UK, May 1998. Chapman & Hall. in print.

[193] THAMM, JENS, STEPHAN WILKE, and LUTZ WEGNER: *A Web solution to concurrency awareness in shared data spaces*. In KAMBAYASHI, YAHIKO, DIK LUN LEE, EE-PENG LIM, and OTHERS (editors): *Advances in Database Technologies, ER'98 Workshops on Data Warehousing and Data Mining, Mobile Data Access, and Collaborative Work Support and Spatio-Temporal Data Management, Singapore, November 19–20, 1998, Proceedings*, number 1552 in *Lecture Notes in Computer Science*, pages 382–395. Springer, 1999.

[194] THELEMANN, SVEN: *Semantische Anreicherung eines Datenmodells für komplexe Objekte*. Dissertation, Universität Gh Kassel, Juni 1996.

[195] THOMAS, S. J. and P. C. FISCHER: *Nested relational structures*. In *Advances in Computing Research*, volume 3, pages 269–307. JAI Press, 1986.

[196] TOLKSDORF, ROBERT: *Coordination technology for workflows on the web: Workspaces*. In *Proceedings of the Fourth International Conference on Coordination Models and Languages COORDINATION 2000*, LNCS. Springer-Verlag, 2000.

[197] TREVOR, JONATHAN, THOMAS KOCH, and GERD WOETZEL: *MetaWeb: Bringing synchronous groupware to the world wide web*. In *Proceedings of the Fifth European Conference on Computer Supported Cooperative Work*, Shared Information Spaces, pages 65–80, Dordrecht, The Netherlands, 1997.

[198] TSICHRITZIS, DENNIS and FREDERICK H. LOCHOVSKY: *Hierarchical database management: A survey*. ACM Computing Surveys, 8(1):105–123, March 1976.

[199] TURAU, VOLKER: *Techniken zur Realisierung Web-basierter Anwendungen*. Informatik-Spektrum, 22, 1999.

[200] TWIDALE, M. and D. NICHOLS: *Browsing is a collaborative process*. CSEG 1/96, Computing Dept, Lancaster University, 1996.

[201] TWIDALE, M. B., D. NICHOLS, J. M. MARIANI, T. RODDEN, and P. SAYER: *Issues in collaborative database browsing*. Research Report CSCW/14/1994, Lancaster University, Computing Department, 1994.

[202] VOSSEN, GOTTFRIED: *Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme*. Addison Wesley, 1994.

[203] VOSSEN, GOTTFRIED: *The CORBA Specification for Cooperation in Heterogeneous Information Systems*. In: KANDZIA, PETER und MATTHIAS KLUSCH (Herausgeber): *Proceedings ot the First International Workshop on Cooperative Information Agents*, Band 1202 der Reihe *LNAI*, Seiten 101–115, Berlin, Februar 26–28 1997. Springer.

[204] W3C: *The Extensible Stylesheet Language (XSL)*. URL: http://www.w3.org/Style/XSL.

[205] W3C: *HTML 4.01 Specification*. URL: http://www.w3.org/TR/html4, December 1999.

[206] W3C: *XML Path Language (XPath)*. URL: http://www.w3c.org/TR/xpath, 1999.

[207] W3C: *XSL Transformations (XSLT) Version 1.0*. URL: http://www.w3c.org/TR/xslt, November 1999.

[208] W3C: *Document Object Model (DOM) Level 2 Core Specification*. URL: http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113, November 2000.

[209] W3C: *Document Object Model (DOM) Level 2 Traversal and Range Specification*. URL: http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113, November 2000.

[210] W3C: *Extensible Markup Language (XML) 1.0 (Second Edition)*. URL: http://www.w3.org/TR/2000/REC-xml-20001006, October 2000.

[211] W3C: *Simple Object Access Protocol (SOAP) 1.1* . URL: http://www.w3.org/TR/SOAP, 2000.

[212] W3C: *Mathematical Markup Language (MathML) Version 2.0*. URL: http://www.w3.org/TR/MathML2/, February 2001.

[213] W3C: *Scalable Vector Graphics (SVG) 1.0 Specification*. URL: http://www.w3.org/TR/SVG/, September 2001.

[214] W3C: *Synchronized Multimedia Integration Language (SMIL 2.0), W3C Recommendation 07 August 2001*. URL: http://www.w3.org/TR/smil20, August 2001.

[215] W3C: *XML Schema Part 0: Primer*. URL: http://www.w3.org/TR/xmlschema-0, May 2001.

[216] W3C: *XQuery 1.0: An XML Query Language*. URL: http://www.w3.org/TR/xquery, 2001.

[217] W3C: *XForms 1.0 W3C Working Draft*. URL: http://www.w3.org/TR/2002/WD-xforms-20020118, January 2002.

[218] WEGNER, L.: *A Portable Record Manager*. Mathematische Schriften Kassel 11/90, Universität Gh Kassel, Fachbereich Mathematik/Informatik, Oktober 1990.

[219] WEGNER, L.: *Let the fingers do the walking: Object manipulation in an NF^2 database editor*. In MAURER, H. (editor): *Proceedings of the Symposium on New Results and New Trends in Computer Science, Graz, June 20–21, 1991*, number 555 in *Lecture Notes in Computer Science*, pages 337–358. Springer, 1991.

[220] WEGNER, L. M.: *Let the fingers do the walking: Object manipulation in an NF^2 database editor*. In MAURER, HERMANN (editor): *Proceedings of New Results and New Trends in Computer Science*, volume 555 of *LNCS*, pages 337–358, Berlin, Germany, June 1991. Springer.

[221] WEGNER, LUTZ: *ESCHER – interactive, visual handling of complex objects in the extended NF^2-database model*. In *'Visual Database Systems', T.L.Kunii(ed), Proc. IFIP TC-2.6, Tokyo*, pages 277–297. North-Holland, April 1989.

[222] WEGNER, LUTZ and MORAD AHMAD: *Orthogonality in DBMS design: the XML approach*. In WERNER, BOB (editor): *Proceedings of the International Workshop on Foundations of Models for Information Integration (FMII-2001)*, pages 66–78, Viterbo, Italy, Sept, 16-18 2001.

[223] WEGNER, LUTZ, MORAD AHMAD, STEFAN FRÖHLICH, CHRISTIAN SCHMIDT, and WILFRIED EVERS: *A collaborative infrastructure for mobile and wireless systems*. Lecture Notes in Computer Science, 2538:136–148, 2002.

[224] WEISER, M.: *The computer for the 21st century*. Scientific American (International Edition), 265(3):66–75, 1991.

[225] WHITTAKER, STEVE, ERIK GEELHOED, and ELIZABETH ROBINSON: *Shared workspaces: How do they work and when are they useful?* International Journal of Man-Machine Studies, 39(5):813–842, 1993.

[226] WIHARTO, MARSELINA: *Comparing Java 2 Micro Edition and .NET Compact Framework*. Master's thesis, Monash University, August 2002.

[227] WILKE, DAGMAR, LUTZ WEGNER, and JENS THAMM: *Database-driven GUI programming*. In *Proceedings of the Second International Conference on Visual Information Systems, San Diego, December 15–17, 1997*, pages 205–214. Knowledge Systems Institute, Skokie, IL, USA, 1997.

[228] WINER, DAVE: *XML-RPC Specification*. URL: `http://www.-xmlrpc.com/spec`, 99.

[229] ZANIOLO, C.: *The database language: GEM*. In *19 ACM SIGMOD Conf. on the Management of Data83, San Jose CA, pp. 207-218*. May 1983. Also published in/as: In "Readings in Object-Oriented Database Systems" edited by S.Zdonik and D.Maier, Morgan Kaufman, 1990.

[230] ZANIOLO, C. (editor): *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, Washington, DC. May 28–30, 1986*, ACM SIGMOD Record 15(2). ACM Press, 1986.

Anhang A

TclDB

Die Befehle von TclDB können in vier Kategorien unterteilt werden; für jede Kategorie ist ein Tcl-Befehl implementiert, der mit weiteren Optionen (Unterbefehlen) gesteuert werden kann.

System-Aufrufe: Für System-Aufrufe von ESCHER, wie Datenbank hochfahren, Datenbank herunterfahren, Applikation auswählen usw. ist der TclDB-Befehl `escher` zuständig. Die Optionen von `escher` sind in der Tabelle A.1 aufgelistet.

`escher boot` *`path`* : Fährt die System-Datenbank im Verzeichnis *`path`* hoch und wählt die System-Applikation als aktive Applikation aus. Bei erfolgreichem Hochfahren des Systems werden die Befehle `table` und `finger` dem Interpreter bekannt gemacht. Liefert *`true`* oder *`false`* zurück.
`escher shutdown`: Fährt die Datenbank herunter. Zuvor werden alle geöffneten Tabellen geschlossen und dabei die vorhandenen *Finger* freigegeben. Die Befehle `finger`, `table` und `fid` werden dabei gelöscht. Liefert *`true`* oder *`false`* zurück.
`escher list application` : Liefert alle vorhandenen `escher` -Applikationen im *`path`* als Tcl-Liste zurück.
`escher list (tables | schemas)` : Liefert alle Tabellen (Schemata) in der aktiven Applikation als Tcl-Liste zurück.
`escher application select` *`name`* : Wählt *`name`* als aktive Applikation aus. Liefert bei Erfolg den Namen der Applikation zurück, sonst wird eine Fehlermeldung ausgegeben.
`escher application new` *`name`* : Erstellt eine neue Applikation mit dem Namen *`name`* und wählt sie als aktive Applikation aus. Wenn die Applikation im *`path`* schon vorhanden ist, wird eine Fehlermeldung ausgegeben.
`escher application delete` *`name`* : Löscht die Applikation *`name`*. Wenn *`name`* die aktive Applikation ist, wird eine Fehlermeldung ausgegeben.

Tabelle A.1: Die Optionen des `escher` und `table`-Kommandos

Tabellen Operationen: Sobald ESCHER hochgefahren ist, hat der Benutzer den Zugriff auf die Datenbank von TclDB aus. Der Benutzer kann dann Tabellen erstellen, löschen, öffnen und schließen. Für diese Operationen ist der Befehl `table` zuständig. Tabelle A.2 enthält die Beschreibung von `table` .

`table open` *`name ?-id tid?`* : Öffnet die Tabelle *`name`* und speichert einen Verweis auf deren Rootfinger unter dem Namen *`tid.root`*. Wenn keine *`tid`* gegeben wird, wird eine systemgewählte *`tid`* ermittelt. Dabei bezeichnet *`name`* den eigentlichen Namen der Tabelle, unter dem ESCHER die Tabelle identifiziert und *`tid`* den Namen, unter dem die Tabelle in der Shell identifiziert wird.
`table list` : Liefert eine Tcl-Liste aller geöffneten Tabellen in der Form *`name1 tid1`* usw.
`table close` *`tid`* : Schließt die Tabelle *`tid`*. Alle existierenden Finger, die zu dieser Tabelle gehören (Finger, die aus dem Rootfinger dieser Tabelle erzeugt wurden), werden zuvor freigegeben.
`table new` *`name schemaname ?-id tid?`* : Erstellt eine neue Tabelle mit dem Namen *`name`* anhand des Schemas *`schemaname`* . Wenn eine *`tid`* gegeben wird, wird die Tabelle unter diesem Namen in TclDB benutzt, sonst wird eine eindeutige Systembezeichnung benutzt.
`table delete` *`name`* : Löscht die Tabelle *`name`* aus der zugehörigen Applikation. Wenn die Tabelle geöffnet ist, wird eine Fehlermeldung ausgegeben.
`table save` *`tid`* : Speichert die Tabelle, die durch *`tid`* angegeben wird.

Tabelle A.2: Die Optionen des `table`-Kommandos

Erzeugen und Freigeben von Fingern : Wenn eine Tabelle geöffnet wird, wird der erste Finger, der Rootfinger der Tabelle, unter dem Namen *tid.root* erzeugt. Von diesem Finger können dann mit dem Befehl `finger` weitere Finger erzeugt (geforkt) oder freigegeben werden. Tabelle A.3 enthält die Beschreibung von `finger` .

Finger Operationen: Jeder erzeugte Finger funktioniert als TclDB-Befehl, den wir als `fid` bezeichnen. Mit diesem Befehl können Operationen auf den Finger selbst ausgeführt werden. Finger Operationen können in zwei Hauptkategorien unterteilt werden: Positionierungsoperationen wie `push` und `go` positionieren einen Finger von einer Stelle im Tabellenbaum an eine andere Stelle. Lese, Schreiboperationen wie `set` , `get` , `on` usw. führen bestimmte Operationen auf Tabellenelementen relativ zur Position eines Fingers aus. Tabellen A.4 und A.5 enthalten die Optionen von `fid`.

`fid push` *(?-first? | ?-last?)* : Positioniert den Finger `fid` auf das erste bzw. letzte unterliegende Element eines komplexen Objektes. `push` *-first* ist die Standardoption. `fid push` *?-index index?* : Positioniert den Finger `fid` auf das mit *index* angegebene Element. Der *index* bezieht sich auf die Elemente eines komplexen Objektes (*set*, *list* oder *tuple*), wobei die Indizierung der Elemente bei Eins beginnt. Falls ein ungültiger Index eingegeben wird, wird eine Fehlermeldung ausgegeben. `fid push` *?-name name?* : Positioniert den Finger `fid` auf das mit *name* angegebene Subattribut eines komplexen Objektes. Falls kein Subattribut mit diesem Namen existiert, wird eine Fehlermeldung ausgegeben. `fid push` *?path?* : Positioniert den Finger anhand einer *Pfadangabe*, die einer in 4.1.1 beschriebenen Syntax unterliegt.
`fid pop` : Positioniert den Finger `fid` auf das umgebende Objekt. Liefert *true* oder *false* zurück (*false*, wenn `fid` auf dem äußersten Objekt der Tabelle steht). `fid top` : Positioniert den Finger `fid` auf das äußerste Objekt der zugehörigen Tabelle. (Rootfinger der Tabelle)
`fid go` *(?-first? | ?-last?)* : Positioniert `fid` auf das erste bzw. letzte Element des umgebenden Objektes. Liefert *true* oder *false* zurück. `fid go` *(?-next? | ?-back?)* : Positioniert `fid` auf das nächste bzw. vorhergehende Element des übergeordneten Objektes. Liefert *true*, wenn die Positionierung erfolgreich war, *false* sonst. `go` *-next* ist die Standardoption. `fid go` *(?-forcedup? | ?-forceddown?)* : Positioniert `fid` auf das vertikal vorhergehende bzw. nächste Objekt der „Tabellenanzeige“. Liefert *true* bei erfolgreicher Positionierung und *false* sonst. `fid go` *?-index* $[+\,-]$ *index?* : Bewegt den Finger `fid` *index*-Schritte (wenn möglich) in der umgebenden Menge. Bei der Eingabe eines ungültigen `index` wird eine Fehlermeldung ausgegeben (Benutzer muß den Index genau angeben.). Bei Eingabe von «+» oder «-» wird der Index *relativ* zu der aktuellen Fingerposition betrachtet.

Tabelle A.4: Positionierungsoperationen des `fid`-Kommandos.

`finger fork` *fid ?-id newfid?* : Erstellt eine Kopie des Fingers *fid* und gibt ihm eine vom Benutzer oder vom System gewählte Bezeichnung *newfid* Dabei sorgt das System für eine eindeutige Bezeichnung, die sich aus dem Tabellennamen, gefolgt von einer eindeutigen Nummer zusammensetzt. *newfid* funktioniert dann als TclDB-Befehl, dessen Funktion weiter unten erklärt wird.
`finger free` *fid* : Gibt den Finger *fid* wieder frei und löscht den mit ihm korrespondierenden TclDB-Befehl. Beim Versuch, den Rootfinger der Tabelle zu löschen, wird eine Fehlermeldung ausgegeben.
`finger list` *?-table tid?* : Gibt eine Liste aller vorhandenen Finger zurück. Mit der Option `-table` *tid* werden nur die Finger ausgegeben, die zu der Tabelle *tid* gehören.

Tabelle A.3: Die Optionen des `finger`-Kommandos

`fid get ?path? :`
Liefert den Wert des Objektes, auf dem `fid` steht, als String zurück. Beim Versuch, den Wert eines komplexen Objektes zu lesen, wird eine Fehlermeldung ausgegeben.

`fid info (-all | -type | -attr | -card) ?path ? :`
Liefert mit der Option `-all` den Typ, den Attributnamen und die Kardinalität des Objektes, auf dem `fid` steht, als Tcl-Liste in der Form `-attr attr` usw. zurück. Sonst wird mit der jeweiligen Option nur die entsprechende Information ausgegeben.

`fid set value ?path? :`
Setzt den Wert des Attributes, auf dem `fid` steht, auf den Wert `value` . Das Attribut des Objektes muß atomar sein.

`fid set -tonull ?path? :`
Setzt den Wert des Mengenobjektes, auf dem `fid` steht, auf Null (DB-NULL). Dabei muß das Objekt leer sein (leere Menge oder Liste). Falls der Finger auf einem atomaren Attribut steht, wird eine Fehlermeldung ausgegeben.

`fid set -toempty ?path? :`
Wandelt das Mengenobjekt, auf dem `fid` steht, von einer DB-NULL-Menge (Liste) in eine leere Menge um. Falls die Menge keine Null-Menge ist, wird eine Fehlermeldung ausgegeben. Die Position des Fingers wird dabei nicht verändert.

`fid set -tosingleton ?path? :`
Fügt in eine leere Menge (Liste) ein neues Element ein. Falls die Menge nicht leer ist oder das Objekt, auf dem der `fid` steht, kein Mengenobjekt ist, wird eine Fehlermeldung ausgegeben. Die Position des Fingers wird nicht geändert.

`fid insert (?-before? | ?-after?) ?path? :`
Fügt ein Null-Objekt in die umgebende Menge (Liste) vor (nach) dem Objekt, auf dem `fid` steht, ein. Der Finger wird auf das neu eingefügte Objekt positioniert. Der Aufruf von `insert` ohne eine der beiden Optionen wandelt eine Null-Menge zu einer leeren Menge um (`set -tonull`) oder fügt ein leeres Element in die Menge ein (`set -tosingleton`), falls die Menge leer ist. Trifft beides nicht zu (Menge ist keine Null-Menge und nicht leer), wird `-after` als Standardoption angenommen.

`fid delete ?path? :`
Löscht das Objekt, auf dem `fid` steht, und versucht, den Finger zuerst auf das vorhergehende, sonst auf das nächste Element der umgebenden Menge zu positionieren. Ist beides nicht möglich (letztes Element des Objektes wurde gelöscht), so wird der Finger auf das umgebende (leere) Objekt positioniert. Bei Eingabe einer Pfadangabe bleibt der Finger auf seiner Position.

`fid on (-first | -last) :`
Liefert `true` , falls `fid` auf dem ersten bzw. letzten Objekt der umgebenden Menge steht. Steht der Finger auf der äußersten Stelle der Tabelle, wird `false` zurückgeliefert.

`fid on -index :`
Liefert den Index des Objektes innerhalb der umgebenden Menge zurück. Der Rootfinger hat den Index 0.

`fid (isempty | isnull) ?path? :`
Liefert `true` , falls `fid` auf einem leeren bzw. DB-NULL-Objekt steht. Ist das Objekt atomar, so wird eine Fehlermeldung ausgegeben. Falls das Objekt ein Null-Objekt ist, wird bei `isempty` DB-NULL zurückgeliefert.

`fid (isatomic | iscomplex) ?path? :`
Liefert `true` , falls `fid` auf einem atomaren bzw. komplexen Objekt steht.

`fid istype (-set | -tuple | -list) ?path? :`
Liefert `true` , falls `fid` auf einer Menge, einer Liste oder auf einem Tupel steht, je nachdem, ob die Option `set` , `list` oder `tuple` eingegeben wird. `false` sonst.

Tabelle A.5: Die Optionen des `fid`-Kommandos. Lese/Schreibe-Operationen

Anhang B

Stylesheets

B.1 Darstellung in XML

Programm B.1 *Das folgende Dokument stellt eine eNF^2-Tabelle als XML-Dokument dar (siehe Abb. 4.7):*

```
<?xml version='1.0'  encoding="ISO-8859-1" standalone="no" ?>
<TASKS nf2type="sett" id="0x5" bg="none">
    <TASK id="0x6" nf2type="ptuplet" bg="none">
        <TASKID id="0x7" nf2type="textt" bg="none">START</TASKID>
        <DESCRIPTION id="0x8" nf2type="textt" bg="none">
             project kickoff
        </DESCRIPTION>
        <DUR id="0x9" nf2type="intt" bg="none">0</DUR>
        <ES id="0xa" nf2type="intt" bg="none">0</ES>
        <LS id="0xb" nf2type="intt" bg="none">0</LS>
        <EF id="0xc" nf2type="intt" bg="none">0</EF>
        <LF id="0xd" nf2type="intt" bg="none">0</LF>
        <ISOTIME id="0xe" nf2type="textt" bg="none">01-01-1998</ISOTIME>
        <REQUIRES id="0xf" nf2type="sett" bg="none" state="empty"/>
    </TASK>
    <TASK id="0x10" nf2type="ptuplet" bg="salmon">
        <TASKID id="0x11" nf2type="textt" bg="salmon">REQ</TASKID>
        <DESCRIPTION id="0x12" nf2type="textt" bg="salmon">
             requirement analysis
        </DESCRIPTION>
        <DUR id="0x13" nf2type="intt" bg="salmon">2</DUR>
        <ES id="0x14" nf2type="intt" bg="salmon">0</ES>
        <LS id="0x15" nf2type="intt" bg="salmon">0</LS>
        <EF id="0x16" nf2type="intt" bg="salmon">2</EF>
        <LF id="0x17" nf2type="intt" bg="salmon">2</LF>
        <ISOTIME id="0x18" nf2type="textt" bg="salmon">15-01-1998</ISOTIME>
        <REQUIRES id="0x19" nf2type="sett" bg="salmon">
            <TASK id="0x1a" nf2type="textt" bg="salmon">START</TASK>
```

```
    </REQUIRES>
</TASK>
<TASK id="0x1b" nf2type="ptuplet" bg="none">
    <TASKID id="0x1c" nf2type="textt" bg="none">STAFF-PREP</TASKID>
    <DESCRIPTION id="0x1d" nf2type="textt" bg="none">
         staff preparation
    </DESCRIPTION>
    <DUR id="0x1e" nf2type="intt" bg="none">24</DUR>
    <ES id="0x1f" nf2type="intt" bg="none">0</ES>
    <LS id="0x20" nf2type="intt" bg="none">2</LS>
    <EF id="0x21" nf2type="intt" bg="none">24</EF>
    <LF id="0x22" nf2type="intt" bg="none">26</LF>
    <ISOTIME id="0x23" nf2type="textt" bg="none">18-06-1998</ISOTIME>
    <REQUIRES id="0x24" nf2type="sett" bg="none">
        <TASK id="0x25" nf2type="textt" bg="none">START</TASK>
    </REQUIRES>
</TASK>
<TASK id="0x26" nf2type="ptuplet" bg="none">
    <TASKID id="0x27" nf2type="textt" bg="none">GSPEC</TASKID>
    <DESCRIPTION id="0x28" nf2type="textt" bg="none">
         global specification
    </DESCRIPTION>
    <DUR id="0x29" nf2type="intt" bg="none">16</DUR>
    <ES id="0x2a" nf2type="intt" bg="none">2</ES>
    <LS id="0x2b" nf2type="intt" bg="none">2</LS>
    <EF id="0x2c" nf2type="intt" bg="none">18</EF>
    <LF id="0x2d" nf2type="intt" bg="none">18</LF>
    <ISOTIME id="0x2e" nf2type="textt" bg="none">07-05-1998</ISOTIME>
    <REQUIRES id="0x2f" nf2type="sett" bg="none">
        <TASK id="0x30" nf2type="textt" bg="none">REQ</TASK>
    </REQUIRES>
</TASK>
<TASK id="0x31" nf2type="ptuplet" bg="none">
    <TASKID id="0x32" nf2type="textt" bg="none">GDESIGN</TASKID>
    <DESCRIPTION id="0x33" nf2type="textt" bg="none">
         global design
    </DESCRIPTION>
    <DUR id="0x34" nf2type="intt" bg="none">15</DUR>
    <ES id="0x35" nf2type="intt" bg="none">24</ES>
    <LS id="0x36" nf2type="intt" bg="none">26</LS>
    <EF id="0x37" nf2type="intt" bg="none">39</EF>
    <LF id="0x38" nf2type="intt" bg="none">41</LF>
    <ISOTIME id="0x39" nf2type="textt" bg="none">01-10-1998</ISOTIME>
    <REQUIRES id="0x3a" nf2type="sett" bg="none">
        <TASK id="0x3b" nf2type="textt" bg="none">GSPEC</TASK>
        <TASK id="0x3c" nf2type="textt" bg="none">STAFF-PREP</TASK>
    </REQUIRES>
</TASK>
<TASK id="0x3d" nf2type="ptuplet" bg="none">
    <TASKID id="0x3e" nf2type="textt" bg="none">FSPEC</TASKID>
    <DESCRIPTION id="0x3f" nf2type="textt" bg="none">
```

```
            fine specification
        </DESCRIPTION>
        <DUR id="0x40" nf2type="intt" bg="none">23</DUR>
        <ES id="0x41" nf2type="intt" bg="none">18</ES>
        <LS id="0x42" nf2type="intt" bg="none">18</LS>
        <EF id="0x43" nf2type="intt" bg="none">41</EF>
        <LF id="0x44" nf2type="intt" bg="none">41</LF>
        <ISOTIME id="0x45" nf2type="textt" bg="none">15-10-1998</ISOTIME>
        <REQUIRES id="0x46" nf2type="sett" bg="none">
            <TASK id="0x47" nf2type="textt" bg="none">GSPEC</TASK>
        </REQUIRES>
    </TASK>
    <TASK id="0x48" nf2type="ptuplet" bg="none">
        <TASKID id="0x49" nf2type="textt" bg="none">SPEC-DOC</TASKID>
        <DESCRIPTION id="0x4a" nf2type="textt" bg="none">
            specification document
        </DESCRIPTION>
        <DUR id="0x4b" nf2type="intt" bg="none">0</DUR>
        <ES id="0x4c" nf2type="intt" bg="none">41</ES>
        <LS id="0x4d" nf2type="intt" bg="none">41</LS>
        <EF id="0x4e" nf2type="intt" bg="none">41</EF>
        <LF id="0x4f" nf2type="intt" bg="none">41</LF>
        <ISOTIME id="0x50" nf2type="textt" bg="none">15-10-1998</ISOTIME>
        <REQUIRES id="0x51" nf2type="sett" bg="none">
            <TASK id="0x52" nf2type="textt" bg="none">FSPEC</TASK>
        </REQUIRES>
    </TASK>
    <TASK id="0x53" nf2type="ptuplet" bg="none">
        <TASKID id="0x54" nf2type="textt" bg="none">FDESIGN</TASKID>
        <DESCRIPTION id="0x55" nf2type="textt" bg="none">
            fine design
        </DESCRIPTION>
        <DUR id="0x56" nf2type="intt" bg="none">27</DUR>
        <ES id="0x57" nf2type="intt" bg="none">41</ES>
        <LS id="0x58" nf2type="intt" bg="none">41</LS>
        <EF id="0x59" nf2type="intt" bg="none">68</EF>
        <LF id="0x5a" nf2type="intt" bg="none">68</LF>
        <ISOTIME id="0x5b" nf2type="textt" bg="none">22-04-1999</ISOTIME>
        <REQUIRES id="0x5c" nf2type="sett" bg="none">
            <TASK id="0x5d" nf2type="textt" bg="none">SPEC-DOC</TASK>
            <TASK id="0x5e" nf2type="textt" bg="none">GDESIGN</TASK>
        </REQUIRES>
    </TASK>
    <TASK id="0x5f" nf2type="ptuplet" bg="none">
        <TASKID id="0x60" nf2type="textt" bg="none">CODE+UTEST</TASKID>
        <DESCRIPTION id="0x61" nf2type="textt" bg="none">
            coding and unit test
        </DESCRIPTION>
        <DUR id="0x62" nf2type="intt" bg="none">44</DUR>
        <ES id="0x63" nf2type="intt" bg="none">68</ES>
        <LS id="0x64" nf2type="intt" bg="none">68</LS>
```

```
        <EF id="0x65" nf2type="intt" bg="none">112</EF>
        <LF id="0x66" nf2type="intt" bg="none">112</LF>
        <ISOTIME id="0x67" nf2type="textt" bg="none">24-02-2000</ISOTIME>
        <REQUIRES id="0x68" nf2type="sett" bg="none">
            <TASK id="0x69" nf2type="textt" bg="none">FDESIGN</TASK>
            <TASK id="0x6a" nf2type="textt" bg="none">SPEC-DOC</TASK>
        </REQUIRES>
    </TASK>
</TASKS>
```

B.2 Standard Visualisierung

Programm B.2 *Das folgende Stylesheet erzeugt aus dem Dokument in B.1 eine Visualisierung als geschachtelte HTML-Tabelle (siehe Abb. 4.7):*

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="ISO-8859-1"/>
<!-- Null Representation für NF² Typen -->
<xsl:variable name="null-sett">???</xsl:variable>
<xsl:variable name="null-listt">&lt;???&gt;</xsl:variable>
<xsl:variable name="empty-sett"></xsl:variable>
<xsl:variable name="empty-listt">&lt;&gt;</xsl:variable>
<xsl:variable name="null-textt">?s?</xsl:variable>
<xsl:variable name="null-intt">?d?</xsl:variable>
<xsl:variable name="null-float">?f?</xsl:variable>
<xsl:variable name="null-chart">?c?</xsl:variable>
<xsl:variable name="lock-icon">lock.gif</xsl:variable>

<!-- Rootnode -->
<xsl:template match="/">
 <html>
   <xsl:apply-templates select="view"/>
 </html>
</xsl:template>

<xsl:template match="view">
  <head>
     <title>
        <xsl:value-of select="@name"/>
     </title>
  </head>
  <body>
  <xsl:apply-templates select="/view/ETable"/>
  </body>
</xsl:template>

<!-- Escher-Tablle  -->
<xsl:template match="/view/ETable">
```

```
    <table align="center" border="1" cellpadding="5">
       <xsl:apply-templates/>
    </table>
</xsl:template>

<!-- Mengen oder Listen -->
<xsl:template match="/view/ETable//*[@nf2type='sett'
                           or @nf2type='listt' or @nf2type='msett']">
   <xsl:element name="td">
   <xsl:choose>
      <xsl:when test="not(@bg='none')">
         <xsl:attribute name="bgcolor">
             <xsl:value-of select="@bg"/>
         </xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
         <xsl:attribute name="bgcolor">
              #FFFFFFFF
         </xsl:attribute>
      </xsl:otherwise>
   </xsl:choose>
   <xsl:choose>
     <xsl:when test="@state='null'">
         <xsl:attribute name="align">
              center
         </xsl:attribute>
         <xsl:choose>
            <xsl:when test="@nf2type=listt">
                <xsl:value-of select = "$null-listt"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select = "$null-sett"/>
            </xsl:otherwise>
         </xsl:choose>
     </xsl:when>
     <xsl:when test="@state='empty'">
         <xsl:attribute name="align">
              center
         </xsl:attribute>
         <xsl:choose>
           <xsl:when test="@nf2type=listt">
             <xsl:value-of select = "$empty-listt"/>
           </xsl:when>
           <xsl:otherwise>
              <xsl:value-of select = "$empty-sett"/>
           </xsl:otherwise>
         </xsl:choose>
     </xsl:when>
     <xsl:otherwise>
         <xsl:attribute name="align">
             left
```

```
        </xsl:attribute>
        <table aling="center" border="1" cellpadding="5">
              <xsl:apply-templates/>
        </table>
     </xsl:otherwise>
   </xsl:choose>
  </xsl:element>
</xsl:template>

<xsl:template match="/view/ETable//*[@nf2type='ptuplet']">
   <tr>
      <xsl:choose>
         <xsl:when test="not(@bg='none')">
            <xsl:attribute name="bgcolor">
              <xsl:value-of select="@bg"/>
            </xsl:attribute>
         </xsl:when>
         <xsl:otherwise>
            <xsl:attribute name="bgcolor">
               <xsl:text>#FFFFFFFF</xsl:text>
            </xsl:attribute>
         </xsl:otherwise>
     </xsl:choose>
     <xsl:if test="@lock>0">
        <xsl:attribute name="background">
            <xsl:value-of select="$lock-icon"/>
        </xsl:attribute>
     </xsl:if>
     <xsl:apply-templates/>
  </tr>
</xsl:template>

<!-- Explizites Tupel -->
<xsl:template match="/view/ETable//*[@nf2type='gtuplet']">
     <xsl:apply-templates/>
</xsl:template>

<!--Atomare Attribute -->
<xsl:template match="/view/ETable//*[@nf2type='intt' or
             @nf2type='floatt' or @nf2type = 'textt' or
             @nf2type='chart']">
  <td align="center">
    <xsl:choose>
       <xsl:when test="@bg='none'">
         <xsl:attribute name="bgcolor">#FFFFFFFF</xsl:attribute>
       </xsl:when>
       <xsl:otherwise>
        <xsl:attribute name="bgcolor">
             <xsl:value-of select="@bg"/>
         </xsl:attribute>
       </xsl:otherwise>
```

```
    </xsl:choose>
     <xsl:if test="@lock>0">
        <xsl:attribute name="background">
            <xsl:value-of select="$lock-icon"/>
        </xsl:attribute>
     </xsl:if>
    <xsl:choose>
       <xsl:when test="@state='null'">
          <xsl:choose>
             <xsl:when test="@nf2type='textt'">
                 <xsl:value-of select="$null-textt"/>
             </xsl:when>
             <xsl:when test="@nf2type='intt'">
                 <xsl:value-of select="$null-intt"/>
             </xsl:when>
             <xsl:when test="@nf2type='floatt'">
                 <xsl:value-of select="$null-floattt"/>
             </xsl:when>
             <xsl:when test="@nf2type='chart'">
                 <xsl:value-of select="$null-chart"/>
             </xsl:when>
          </xsl:choose>
         </xsl:when>
         <xsl:otherwise>
           <xsl:value-of select="."/>
         </xsl:otherwise>
    </xsl:choose>
   </td>
</xsl:template>

<xsl:template match="/view/ETable//*[@nf2type='sett']/*[
              @nf2type = 'intt' or @nf2type='floatt' or
              @nf2type = 'textt' or @nf2type='chart']">
  <tr>
    <xsl:choose>
       <xsl:when test="@bg='none'">
         <xsl:attribute name="bgcolor">#FFFFFFFFF</xsl:attribute>
       </xsl:when>
       <xsl:otherwise>
        <xsl:attribute name="bgcolor">
             <xsl:value-of select="@bg"/>
         </xsl:attribute>
       </xsl:otherwise>
   </xsl:choose>
   <td>
    <xsl:choose>
       <xsl:when test="@bg='none'">
         <xsl:attribute name="bgcolor">#FFFFFFFFF</xsl:attribute>
       </xsl:when>
       <xsl:otherwise>
        <xsl:attribute name="bgcolor">
```

```
            <xsl:value-of select="@bg"/>
          </xsl:attribute>
        </xsl:otherwise>
    </xsl:choose>

    <xsl:choose>
       <xsl:when test="@state='null'">
          <xsl:attribute name="align">center</xsl:attribute>
          <xsl:choose>
             <xsl:when test="@nf2type='textt'">
                 <xsl:value-of select="$null-textt"/>
             </xsl:when>
             <xsl:when test="@nf2type='intt'">
                 <xsl:value-of select="$null-intt"/>
             </xsl:when>
             <xsl:when test="@nf2type='floatt'">
                 <xsl:value-of select="$null-floattt"/>
             </xsl:when>
             <xsl:when test="@nf2type='chart'">
                 <xsl:value-of select="$null-chart"/>
             </xsl:when>
          </xsl:choose>
      </xsl:when>
      <xsl:otherwise>
          <xsl:value-of select="."/>
     </xsl:otherwise>
   </xsl:choose>
   </td>
   </tr>
 </xsl:template>
</xsl:stylesheet>
```

B.3 Spezielle Visualisierung

Programm B.3 *Das folgende Stylesheet erzeugt eine spezielle Visualisierung vom selben XML-Dokument aus B.1. Es produziert ein GANT-Diagram in SVG (siehe Abb. 4.8):*

```
<?xml version='1.0'?>
<xsl:stylesheet version='1.0'
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns="http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<xsl:output method="xml" indent="yes" media-type="image/svg"/>
<!-- globale Variablen -->
<xsl:variable name="lineDist" select="10"/>
<xsl:variable name="gantHeight" select="20"/>
<!-- Begin Koordinate in X-Richtung (nicht 0) -->
<xsl:variable name="XBegin" select="100"/>
<!-- Skalierung in X-Richtung -->
```

```
<xsl:variable name="XScale" select="4"/>
<!--Zeichne rekursiv 5 vertikale Linien auf der X-Achse-->
<!--bis die Länge dieser Achse überschritten wird.-->
<xsl:template name="drawLines">
    <xsl:param name="begin"/>
    <xsl:param name="lastX"/>
    <!--Die Höhe auf der die Striche gezeichnet werden-->
    <xsl:param name="Y"/>
    <xsl:param name="number"/>
    <xsl:variable name="y0" select="$Y - 7"/>
    <xsl:variable name="y1" select="$Y + 7"/>
    <xsl:variable name="currentNumber" select="$number"/>
    <xsl:variable name="x0" select="$begin"/>
    <xsl:variable name="currentNumber" select="$number"/>
    <xsl:choose>
     <xsl:when test="not($currentNumber mod 2)" >
        <xsl:variable name="y0" select="$lineDist - 7"/>
        <line style="stroke-width:0.5;fill:none; stroke:black"
           x1="$x0" y1="$y0" x2="$x0" y2="$y1"/>
        <xsl:variable name="ty" select="$y1 + 10"/>
        <xsl:choose>
            <xsl:when test="$currentNumber &gt; 100" >
               <xsl:variable name="tx" select="$x0 - 10"/>
               <text x="$tx" y="$ty">
                   <xsl:value-of select="$currentNumber"/>
               </text>
            </xsl:when>
            <xsl:otherwise>
               <xsl:variable name="tx" select="$x0 - 5"/>
               <text x="$tx" y="$ty">
                  <xsl:value-of select="$currentNumber"/>
               </text>
            </xsl:otherwise>
        </xsl:choose>
     </xsl:when>
     <xsl:otherwise>
       <line style="stroke-width:2;fill:none; stroke:black"
          x1="$x0" y1="$y0" x2="$x0" y2="$y1"/>
     </xsl:otherwise>
    </xsl:choose>
    <xsl:variable name="currentNumber" select="$currentNumber + 5" />
    <xsl:variable name="x0" select="$begin + (5 * $XScale)"/>
    <xsl:if test="($x0 + (5 * $XScale))  &lt; $lastX" >
        <xsl:call-template name="drawLines">
             <xsl:with-param name="begin" select="$x0"/>
             <xsl:with-param name="lastX" select="$lastX"/>
             <xsl:with-param name="Y" select="$Y"/>
            <xsl:with-param name="number" select="$currentNumber"/>
        </xsl:call-template>
    </xsl:if>
</xsl:template>
```

```
<!--Zeichne X-, Y-Koordinaten-->
<xsl:template name="drawCoords">
  <xsl:param name="xend">0</xsl:param>
  <xsl:param name="yend">0</xsl:param>
  <!--Begin der Zeichnung-->
  <xsl:variable name="CXBegin" select="5"/>
  <xsl:variable name="CYBegin" select="5"/>
  <line x1="$CXBegin" y1="$CYBegin" x2="$xend"  y2="$CYBegin"
        style="stroke-width:3;fill:none; stroke:red"/>
  <line x1="$CXBegin" y1="$CYBegin" x2="$CXBegin"  y2="$yend"
        style="stroke-width:3;fill:none; stroke:red"/>
  <line x1="$xend" y1="$CYBegin" x2="$xend"  y2="$yend"
        style="stroke-width:3;fill:none; stroke:red"/>
  <line x1="$CXBegin"  y1="$yend"  x2="$xend" y2="$yend"
        style="stroke-width:3;fill:none; stroke:red"/>
  <!--Zeichne vertikale Linien mit Skala-->
  <xsl:call-template name="drawLines">
      <xsl:with-param name="number" select="0"/>
      <xsl:with-param name="begin" select="$XBegin"/>
      <xsl:with-param name="lastX" select="$xend"/>
      <xsl:with-param name="Y" select="$yend"/>
  </xsl:call-template>
</xsl:template>
<!--Zeichne eine Task deren Dauer größer Null ist -->
<xsl:template name="drawTask">
   <xsl:param name="X0">0</xsl:param>
   <xsl:param name="X1">0</xsl:param>
   <xsl:param name="Y0">0</xsl:param>
   <xsl:param name="Y1">0</xsl:param>
   <xsl:param name="slack">0</xsl:param>
   <xsl:param name="taskid"/>
   <xsl:param name="descr"/>
   <xsl:param name="date"/>
   <!--Breite des Rechtecks: X1 -X0-->
   <xsl:variable name='w' select='$X1 -$X0'/>
   <g id="$taskid">
   <xsl:choose>
       <xsl:when test="$slack!=0">
           <rect x="$X0" y="$Y0" width="$w"  height='$gantHeight'
                 style="stroke:blue; fill:blue"/>
           <rect x="$X1" y="$Y0" width="$slack" height="$gantHeight"
                 style="stroke:blue; fill:yellow"/>
        </xsl:when>
        <xsl:otherwise>
            <!--Kritischer Task-->
            <rect x="$X0" y="$Y0" width="$w"  height='$gantHeight'
                  style="stroke:red; fill:blue"/>
        </xsl:otherwise>
    </xsl:choose>
    <!--Die X-, Y-koordinate für den Text-->
    <xsl:variable name='ty' select='$Y0 + 15'/>
```

```
    <xsl:variable name='tx' select='$X1 + $slack + 5'/>
    <text x="$tx" y="$ty">
          <xsl:value-of select="$descr"/>
    </text>
    <!--Das Datum links vom Balken-->
    <xsl:variable name='dx' select='$X0 - 80'/>
    <text x="$dx" y="$ty">
         <xsl:value-of select="$date"/>
   </text>
   </g>
</xsl:template>
<!--Zeichnet einen Task dessen Dauer gleich 0 ist (eine Route)-->
<xsl:template name="drawNullTask">
   <xsl:param name="X0">0</xsl:param>
   <xsl:param name="Y0">0</xsl:param>
   <xsl:param name="slack">0</xsl:param>
   <xsl:param name="descr"/>
   <xsl:param name="date"/>
   <xsl:param name="taskid"/>
   <xsl:variable name="py0" select="$Y0 + ($gantHeight * 0.5)"/>
   <xsl:variable name="px1" select="$X0 + ($gantHeight * 0.5)"/>
   <xsl:variable name="py1" select="$py0 - ($gantHeight * 0.5)"/>
   <xsl:variable name="px2" select="$X0 +  $gantHeight"/>
   <xsl:variable name="py3" select="$py0 + ($gantHeight * 0.5)"/>
   <g id="$taskid">
   <polygon style="stroke-width:1"
         points="$X0,$py0 $px1,$py1 $px2,$py0 $px1,$py3 $X0,$py0"/>
   <xsl:variable name="ry" select="$py0 - ($gantHeight * 0.25)"/>
   <xsl:variable name="rx" select="$X0 + ($gantHeight * 0.25) + 1"/>
   <xsl:variable name="rw" select="($gantHeight * 0.5) - 2"/>
   <xsl:choose>
       <xsl:when test="$slack!=0">
          <rect x="$rx" y="$ry" width="$rw" height="$rw"
                style="stroke:red; fill:yellow"/>
        </xsl:when>
        <xsl:otherwise>
            <!--Kritischer Task -->
          <rect x="$rx" y="$ry" width="$rw" height="$rw"
                style="stroke:red; fill:red"/>
        </xsl:otherwise>
    </xsl:choose>
    <!--Die X-, Y-koordinate für den Text-->
    <xsl:variable name='ty' select='$Y0 + 15'/>
    <xsl:variable name='tx' select='$X0 + $gantHeight + $slack + 5'/>
    <text x="$tx" y="$ty">
          <xsl:value-of select="$descr"/>
    </text>
    <!--Das Datum links vom Balken-->
    <xsl:variable name='dx' select='$X0 - 80'/>
    <text x="$dx" y="$ty">
         <xsl:value-of select="$date"/>
```

```
   </text>
   </g>
</xsl:template>

<xsl:template match='/'>
 <svg>
   <xsl:variable name="lastX"
                 select="((./TASKS/TASK[TASKID='END']/LF) * $XScale) +
                         3*$XBegin"/>
   <xsl:variable name="lastY"
         select="count(/TASKS/TASK) * ($gantHeight + $lineDist)
                 + $gantHeight + $lineDist"/>
    <xsl:attribute name="width">
         <xsl:value-of select="$lastX + $gantHeight"/>
    </xsl:attribute>
    <xsl:attribute name="height">
         <xsl:value-of select="$lastY + 2*$gantHeight"/>
    </xsl:attribute>
   <xsl:call-template name="drawCoords">
        <xsl:with-param name="xend" select="$lastX"/>
        <xsl:with-param name="yend" select="$lastY"/>
   </xsl:call-template>
   <xsl:variable name='vY0' select='0'/>
   <xsl:variable name='vY1' select='0'/>
   <xsl:for-each select="TASKS/TASK">
       <xsl:variable name="duration" select="DUR"/>
       <xsl:variable name="vY0" select="$vY1 + $lineDist"/>
       <xsl:variable name="vY1" select="$vY0 + $gantHeight"/>
       <xsl:choose>
           <xsl:when test="$duration=0">
               <xsl:call-template name="drawNullTask">
                   <xsl:with-param name="X0"
                                   select="$XBegin + (ES * $XScale)"/>
                   <xsl:with-param name="Y0" select="$vY0"/>
                   <xsl:with-param name="slack"
                                   select="(LF -EF) * $XScale"/>
                   <xsl:with-param name="descr">
                      <xsl:value-of select="./DESCRIPTION"/>
                   </xsl:with-param>
                   <xsl:with-param name="date">
                       <xsl:value-of select="./ISOTIME"/>
                   </xsl:with-param>
               </xsl:call-template>
           </xsl:when>
           <xsl:otherwise>
              <xsl:call-template name="drawTask">
                   <xsl:with-param name="X0"
                                   select="$XBegin + (ES * $XScale)"/>
                   <xsl:with-param name="X1"
                                   select="$XBegin + (EF * $XScale)"/>
                   <xsl:with-param name="Y0" select="$vY0"/>
```

```
                    <xsl:with-param name="Y1" select="$vY1"/>
                    <xsl:with-param name="slack"
                                    select="(LF -EF) * $XScale"/>
                    <xsl:with-param name="taskid">
                       <xsl:value-of select="./TASKID"/>
                    </xsl:with-param>
                    <xsl:with-param name="descr">
                       <xsl:value-of select="./DESCRIPTION"/>
                    </xsl:with-param>
                    <xsl:with-param name="date">
                        <xsl:value-of select="./ISOTIME"/>
                    </xsl:with-param>
                </xsl:call-template>
             </xsl:otherwise>
         </xsl:choose>
      </xsl:for-each>
   </svg>
  </xsl:template>
</xsl:stylesheet>
```

B.4 Schemavisualisierung

Programm B.4 *Durch ein generisches Stylesheet kann jedes XML Schema als geschachteltes eNF^2-Schema visualisiert werden (siehe Abschnitt 4.2). Wir betrachten als Beispiel das folgende Schema:*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:prj="http://www.db.informatik.uni-kassel.de/DTDS/Projects"
            targetNamespace="http://www.db.informatik.uni-kassel.de/DTDS/Projects">

<xsd:annotation>
   <xsd:documentation xml:lang="de-DE">
      "PROJECTS"-Schema.
      Morad Ahmad.
   </xsd:documentation>
</xsd:annotation>

<xsd:element name="TASKS" type="TasksType"/>

<xsd:complexType name="TasksType">
    <xsd:attribute name="id" type="xsd:ID"/>
    <xsd:attribute name="nf2type" type="abt:nf2typeType"
                   use="optional" default="set"/>
    <xsd:element name="TASK" type="TaskType"
                 minOccurs="0" maxOccurs="unbounded"/>
</xsd:complexType>

<xsd:complexType name="TaskType">
```

```
   <xsd:attribute name="id" type="xsd:ID"/>
   <xsd:attribute name="nf2type" type="abt:nf2typeType"
                  use="optional" default="ptupel"/>
   <xsd:element name="TASKID" type="atomicType"/>
   <xsd:element name="DESCRIPTION" type="atomicType"/>
   <xsd:element name="DUR" type="atomicType"/>
   <xsd:element name="ES" type="atomicType"/>
   <xsd:element name="LS" type="atomicType"/>
   <xsd:element name="EF" type="atomicType"/>
   <xsd:element name="LF" type="atomicType"/>
   <xsd:element name="ISOTIME" type="atomicType"/>
   <xsd:element name="REQUIRES" type="RequiresType"/>
</xsd:complexType>

<xsd:complexType name="RequiresType">
   <xsd:attribute name="id" type="xsd:ID"/>
   <xsd:attribute name="nf2type" type="abt:nf2typeType"
                  use="optional" default="set"/>
   <xsd:element name="TASK" type="atomicType"
                minOccurs="0" maxOccurs="unbounded"/>
</xsd:complexType>

<xsd:simpleyType name="nf2typeType" >
     <xsd:restriction base="xsd:string">
         <xsd:enumeration value="atomic"/>
         <xsd:enumeration value="gtuple"/>
         <xsd:enumeration value="list"/>
         <xsd:enumeration value="ptuple"/>
         <xsd:enumeration value="set"/>
     </xsd:restriction>
</xsd:simpleyType>

<xsd:complexType name="atomicType" >
    <xsd:simpleContent>
       <xsd:extension base="xsd:string">
            <xsd:attribute name="id" type="xsd:ID"/>
       </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

</xsd:schema>
```

Programm B.5 *Mit dem folgenden Stylesheet kann jedes XML-Schema als geschachtelte HTML-Tabelle dargestellt werden:*

```
<?xml version="1.0" encoding='ISO-8859-1' standalone='yes'?>
<xsl:stylesheet version='1.0' xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsl:output method="html"/>

<xsl:template match="/">
    <HTML>
```

```
        <BODY><xsl:apply-templates/></BODY>
    </HTML>
</xsl:template>

<xsl:template match="xsd:schema">
    <xsl:element name="TABLE">
        <xsl:attribute name="CELLPADDING">
           10
        </xsl:attribute>
        <xsl:attribute name="CELLSPACING">
           0
        </xsl:attribute>
        <xsl:element name="TR">
            <xsl:attribute name="BORDER">
               1
            </xsl:attribute>
            <xsl:apply-templates select="xsd:element"/>
        </xsl:element>
    </xsl:element>
</xsl:template>

<xsl:template match="xsd:element">
       <xsl:choose>
            <xsl:when test="//xsd:complexType
                    [@name=current()/@type]/xsd:attribute
                    [@name='nf2type' and @use='optional' and
                    @default='set']">
               <xsl:element name="TD">
                   <xsl:element name="TABLE">
                       <xsl:attribute name="CELLPADDING">
                          10
                       </xsl:attribute>
                       <xsl:attribute name="CELLSPACING">
                          0
                       </xsl:attribute>
                       <xsl:attribute name="BORDER">
                          1
                       </xsl:attribute>
                       <xsl:element name="TR">
                            <xsl:element name="TD">
                              <xsl:attribute name="COLSPAN">
                                  10
                              </xsl:attribute>
                                <xsl:element name="CENTER">
                                   <xsl:value-of select="./@name"/>
                                </xsl:element>
                            </xsl:element>
                       </xsl:element>
                       <!-- Verarbeite die Definition von Mengen -->
                       <xsl:apply-templates
                        select="//xsd:complexType[@name=current()/@type]"/>
```

```
                    </xsl:element>
                </xsl:element>
            </xsl:when>
            <xsl:when test="//xsd:complexType
                [@name=current()/@type]/xsd:attribute
                [@name='nf2type' and @use='optional' and
                 @default='ptuple']">
                <tr>
                   <xsl:apply-templates
                   select="//xsd:complexType[@name=current()/@type]"/>
                </tr>
            </xsl:when>
            <xsl:otherwise>
                <xsl:element name="TD">
                    <xsl:value-of select="./@name"/>
                    <xsl:apply-templates
                    select="//xsd:complexType[@name=current()/@type]"/>
                </xsl:element>
            </xsl:otherwise>
        </xsl:choose>
</xsl:template>

<xsl:template match="xsd:complexType">
    <xsl:apply-templates select=".//xsd:element"/>
</xsl:template>
</xsl:stylesheet>
```

Anhang C

Beispiele aus dem Quellcode

C.1 Schemabehandlung

Programm C.1 *`finger2dtd` erzeugt aus einem eNF^2-Schema eine DTD, die die Struktur der Tabelle als XML-Dokument beschreibt. Auf ähnlicher Weise kann man ein XML Schema erzeugen:*

```
proc tcldb::finger2dtd {sf {to stdout}} {
    set attr       [tcldb::sf_attr $sf]
    set type       [tcldb::sf_type $sf ]
    switch -regexp $type {
        ptuplet|gtuplet {
            puts $to "<!ELEMENT $attr-tuple ("
            tcldb::sf_push $sf
            while {![tcldb::$sf on -last]} {
                puts -nonewlinw $to "[tcldb::sf_attr $sf],"
            }
            puts -nonewline $to "[tcldb::sf_attr $sf])>"
            tcldb::sf_pop $sf
            tcldb::$sf push -last
            tcldb::iterate $sf {
                finger2dtd $sf $to
            }
            tcldb::$sf pop
        }
        sett|msett|listt {
            puts $to "<!ELEMENT $attr ($attr-tuple)*>"
            puts $to "<!ATTLIST $attr nf2type CDATA #REQUIRED>"
            puts $to "<!ATTLIST $attr state CDATA #REQUIRED>"

            puts -nonewline $to "<!ELEMENT $attr-tuple ("
            tcldb::sf_push $sf
            while {![tcldb::$sf on -last]} {
                puts -nonewline $to "[tcldb::sf_attr $sf],"
```

```
                tcldb::$sf go -next
            }
            puts $to "[tcldb::sf_attr $sf])>"
            tcldb::sf_pop $sf
            # Die Unterattribute.
            tcldb::$sf push -last
            tcldb::iterate $sf {
                finger2dtd $sf $to
            }
            tcldb::$sf pop
            puts $to "<!ATTLIST $attr-tuple nf2type CDATA #REQUIRED>"
            puts $to "<!ATTLIST $attr-tuple state CDATA #REQUIRED>"
        }
        undefinedt {
            puts $to "<!ELEMENT $attr (#PCDATA)>"
            puts $to "<!ATTLIST $attr-tuple nf2type CDATA #REQUIRED>"
            puts $to "<!ATTLIST $attr-tuple state CDATA #REQUIRED>"
        }
        default {
            puts $to "<!ELEMENT $attr (#PCDATA)>"
            puts $to "<!ATTLIST $attr-tuple nf2type CDATA #REQUIRED>"
            puts $to "<!ATTLIST $attr-tuple state CDATA #REQUIRED>"
        }
    }
}
```

C.2 Auszüge aus dem Datenerver

Programm C.2 *`finger_write` Behandlung von Finger-Schreiboperationen:*

```
proc finger_write {tf args} {
    global XLInfo clientInfo tableInfo fingerInfo
    global dp_rpcFile clientInfo cid2id
    set client   $cid2id($dp_rpcFile)
    if {$fingerInfo($tf,vfinger) == ""} {
        error "finger is not registred!"
    }
    set commands {set insert delete swap}
    set length   [llength $args]
    set cmd      [lindex $args 0]
    set options  [lreplace  $args 0 0]
    set tid [tcldb::$tf tid]
    foreach {table rid} [set link [tcldb::$tf link]] {}
    # Der Finger eines anderen Benutzers steht auf dem Objekt!
    # Löschen ist ist in diesem Fall nicht erlaubt.
    if {$cmd == "delete"} {
        if {[llength $args] == 2} {
            set where [lindex $args 1]
        } else {set where "-none"}
        foreach c $tableInfo($tid,clients) {
```

```
            foreach finger $clientInfo($c,$tid,fingers) {
                if {($finger != $tf) && \
                        ([lsearch [tcldb::$finger rid -stack] $rid] >= 0)} {
                    error "finger on object, could't delete"
                }
            }
        }
    }
    # Objekt ist nicht gesperrt! Kann daher nicht modifiziert werden.
    if {[set lRid [Xlock_rid $tid $rid]] == ""} {
        error "object not writable, no lock seted on objekt!"
    }
    # Überprüfen der Sperrgranularität.
    if {[check_granularity $tf $lRid $cmd] == 0} {
        error "parent attribute must be locked"
    }
    # Benutzer ist nicht der Besitzer der Sperre.
    if {$client != $XLInfo($table,$lRid,client)} {
        puts "$client != $XLInfo($table,$lRid,client)"
        error "The specified action isn't allowed"
    }
    # Ok! Operation ausführen.
    uplevel tcldb::$tf $args
    set rid2 [tcldb::$tf rid]
    switch -- $cmd {
        "delete" {
            set last [tcldb::$tf on -last]
            table_notify $tid $client \
                    delete o_$tf $rid $rid2 $last
        }
        "insert" {
            if {[llength $args] == 2} {
                table_notify $tid $client insert [lindex $args 1] o_$tf $rid2
            } else {
                table_notify $tid $client insert o_$tf $rid2
            }
        }
        "set" {
            if {$length == 2} {
                table_notify $tid $client setatomic $rid2 [lindex $args 1]
            } else {
                switch -- $options {
                    "-tonull"  {
                        table_notify $tid $client nulltoempty $rid
                    }
                    "-toempty" {
                        table_notify $tid $client emptytonull $rid
                    }
                }
            }
        }
```

```
    }
    # Aktualisiere letzte Veränderungszeit.
    set XLInfo($table,$lRid,etime) [clock seconds]
    return
}
```

C.3 Auszüge aus dem VClient

Programm C.3 *`Tablview::visual` visualisiert Datenbankoperationen in der virtuellen Visualisierung bei dem VClient:*

```
itcl::body Tableview::visual {other method args} {
    set length [llength $args]
    switch -- $method {
        insert {
            if {$length == 3} {
                set where [lindex $args 0]
                set fgr [lindex $args 1]
                set rid [lindex $args 2]
                if {($where == "-before") || ($where == "-after")} {
                    $vtable insert $fgr $rid $where
                    if $other {
                        generateView
                    }
                } else {
                    error "bad option should be -before or -after"
                }
            } elseif {$length == 2} {
                # Normaler insert.
                set fgr [lindex $args 0]
                set rid [lindex $args 1]
                $vtable insert $fgr $rid -none
                if $other {
                    generateView
                }
            } else {
                error "wrong # args should be $this visual insert options"
            }
        }
        delete {
            if {$length == 3} {
                foreach {fgr rid1 rid2 } $args {}
                $vtable delete  $fgr $rid1 $rid2
                if $other {
                    generateView
                }
            } else {
                error "wrong # args should be $this visual delete options"
            }
```

```
}
singleTone2Empty {
    if {$length == 3} {
        foreach {fgr rid1 rid2 } $args {}
        $vtable singleTone2Empty $fgr $rid1 $rid2
        if $other {
            generateView
        }
    } else {
        error "wrong # args should be $this visual delete options"
    }
}
emptytonull {
    if {$length == 2} {
        foreach {fgr rid} $args {}
        $vtable emptytonull $fgr $rid
        if $other {
            generateView
        }
    } else {
        error "wrong # args should be $this visual emptytonull rid"
    }
}
nulltoempty {
    if {$length == 1} {
        set rid [lindex $args 0]
        $vtable nulltoempty  $rid
        if $other {
            generateView
        }
    } else {
        error "wrong # args should be $this visual nulltoempty rid"
    }

}
setatomic {
    if {$length == 2} {
        foreach {rid text} $args {}
        $vtable setatomic $rid $text
        if $other {
            generateView
        }
    } else {
        error "wrong # args should be $this visual setatomic rid text"
    }
}
newlock {
    if {$length == 3} {
        foreach {client table rid} $args {}
        $vtable newlock $client $table $rid
    } else {
```

```
                error "wrong # args should be\
                    $this visual newlock client table rid"
            }
        }
        deletelock {
            if {$length == 3} {
                foreach {client table rid} $args {}
                $vtable deletelock $client $table $rid
                if $other {generateView}
            } else {
                error "wrong # args\
                    should be $this visual deletelock client table rid"
            }
        }
        default {error "bad option\
            should be insert, delete, emptytonull, \
            nulltoempty, setatomic, newlock or deletelock"}
    }
}
```

Programm C.4 *`Vtable::insert` visualisiert die Datenbankoperation `insert -after` durch Modifizierung des DOM-Baumes:*

```
itcl::body Vtable::insert {fgr rid {where "-none"}} {
    $finger push -rid $rid
    set ridStack   [$finger rid -stack]
    set parentRid  [lindex $ridStack [expr [llength $ridStack] -2]]
    switch -- $where {
        -after {
            set parentNode [$domRoot find id $parentRid]
            set newNode [fingertodom]
            if {[$finger go -next]} {
                set beforeRid [$finger rid]
                if {[isObjectDrawed $beforeRid]} {
                    # Tupel ist schon gezeichnet.
                    set beforeNode [$domRoot find id "$beforeRid"]
                    $parentNode insertBefore $newNode $beforeNode
                } else {
                   # Am Ende im DOM-Tree.
                    $parentNode appendChild $newNode
                }
                $finger go -back
            } else {
                # Wurde am Ende der Tabelle (Menge) eingefügt.
                $parentNode appendChild $newNode
            }
        }
        -before {
            set parentNode [$domRoot find id $parentRid]
            set newNode [fingertodom]
            #"next" funktioniert, weil "insert -before" erfolgreich war.
            $finger go -next
```

```
            set afterRid [$finger rid]
            $finger go -back
            set afterNode [$domRoot find id $afterRid]
            $parentNode insertBefore $newNode $afterNode
        }
        -none {
            # Kann nicht "null" sein, da es einen erfolgreichen "insert" gab.
            if {[$finger isatomic] || ![$finger isempty]} {
                set parentNode [$domRoot find id $parentRid]
                set newNode [fingertodom]
                $parentNode setAttribute state ""
                $parentNode appendChild $newNode
            }
            if {[$finger isempty]} {
                set parentNode [$domRoot find id $rid]
                $parentNode setAttribute state "empty"
            }
        }
        default {
            error "bad option should be after or before"
        }
    }
    $fgr set_position
}
```

Programm C.5 *`Tableview::colorizeFingers` Färben von Fingern in der Tabellenanzeige:*

```
# movedFinger: Finger der seine Position geändert hat.
itcl::body Tableview::colorizeFingers {movedFinger} {
    # Nur Finger, die im Fokus eines Benutzers sind, werden auch
    # in  der Schemaanzeige  gezeigt.
    set allFingersCount [expr  [llength $ofingers] + $fingercount]
    if {$allFingersCount > 1 } {
        $movedFinger hide;# Erst Finger verstecken.
        set sectionBg          red
        set movedFingerbg      [$movedFinger cget -standardbg]
        set movedFTableStack   [$movedFinger getStack]
        set movedFSchemaStack  [$movedFinger getSchemaStack $schemafinger]
        # Liste der anderen Finger im viewport
        if [$movedFinger isa Ofinger] {
            set otherFingers "$fingerlist [ldelete $ofingers $movedFinger]"
        } else {
            set otherFingers "[ldelete $fingerlist $movedFinger] $ofingers"
        }
        foreach vf $otherFingers {
            set vfTableStack  [$vf getStack]
            set vfSchemaStack [$vf getSchemaStack $schemafinger]
            set vfbg          [$vf cget -standardbg]
            # bestimme Farbe von vf
            switch -- [comparefingers $movedFTableStack $vfTableStack] {
                "same" {
```

```
                SFinger::raise $fingerTree $vf $movedFinger
                $vf configure -background $sectionBg
                $vf configure -schemabg $sectionBg
                $vf show
            }
            "different" {
                $vf configure -background $vfbg
                $movedFinger configure -background $movedFingerbg
                switch -- [comparefingers $movedFSchemaStack $vfSchemaStack] {
                    "different" {
                        $vf configure -schemabg $vfbg
                    }
                    "same" {
                        SFinger::raise $fingerTree $vf $movedFinger
                        $vf configure -schemabg $sectionBg
                    }
                    "parent" {
                        SFinger::raise $fingerTree $vf $movedFinger
                        $vf configure -schemabg $vfbg
                    }
                    "child" {
                        SFinger::raise $fingerTree $movedFinger $vf
                        $vf configure -schemabg  $vfbg
                        $movedFinger configure -schemabg $movedFingerbg
                    }
                }
                $vf show
                $movedFinger show
            }
            "parent" {
                $vf hide
                SFinger::raise $fingerTree $vf $movedFinger
                $movedFinger configure -background $movedFingerbg
                $movedFinger configure -schemabg $movedFingerbg
                $movedFinger show
                $vf configure -background $vfbg
                $vf configure -schemabg $vfbg
                $vf show
            }
            "child" {
                $vf hide
                SFinger::raise $fingerTree $movedFinger $vf
                $vf configure -background $vfbg
                $vf configure -schemabg $vfbg
                $vf show
                $movedFinger configure -background $movedFingerbg
                $movedFinger configure -schemabg $movedFingerbg
                $movedFinger show
            }
            default {error "bad finger compare value"}
        }
```

```
        }
    } else {
  # some methods calls hide bevor, and expect me to show finger here
  # if everything is currectly implemented, then this will be unnecessary!
        $movedFinger show
    }
}
```

C.4 Auszüge aus der Clientschnittstelle

Programm C.6 *Konstruktor und einige Methoden der Klasse* `TclDBSocket`*:*

```
package tcldbc;
import java.io.InputStream;
import java.io.BufferedReader;
import java.io.PrintStream;
import java.io.IOException;
import java.io.InterruptedIOException;
import java.io.InputStreamReader;
import java.net.Socket;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import javax.xml.transform.TransformerConfigurationException;

/**
 * Baut eine Socket-Verbindung zum VClient auf und implementiert Methoden
 * Für die Kommunikation mit dem VClient. Die Clientschnittstelle benutzt
 * ausschließlich diese Methoden zur Kommunikation mit VClient.
 */
public class TclDBSocket {
    /** Die Socket zum Groupware-Server (VClient)*/
    private Socket tcldbSocket;
    /** Servername.*/
    private String tcldbHost;
    private int tcldbPort;
    /** "listener", der auf Nachrichten (Notifications) vom Server wartet.*/
    private TclDBListener listener;
    /** Socket-output-stream: zum Verschicken von Nachrichten an den Server.*/
    private PrintStream tcldbOutputStream;
    /** input stream: Lesesn von Servernachrichten.*/
    private InputStream tcldbInputStream;
    private BufferedReader tcldbBufferedReader;
    private WrappedInputStream tcldbWrappedInputStream;
    private TclDBMessageHandler messageHandler=null;
    private String standardTableStyle = null;
    /** Gibt an, ob das socket-input-stream frei ist.*/
    private boolean free = false;
    /** Lokaler Buffer zum Lesen.*/
    private TclDBByteBuffer localBuffer;
```

```
/** Escher-Kommand*/
private EscherCommand escher;
/** Table-Kommand*/
private TableCommand table;
/** Finger-Kommand*/
private FingerCommand finger;

public TclDBSocket (String host, int port)
    throws TclDBEmptyMessageException,
           TransformerConfigurationException,
           ParserConfigurationException,
           SAXException, IOException {
    tcldbHost = host;
    tcldbPort = port;
    // Buffer = 16384 = 16 KB.
    localBuffer = new TclDBByteBuffer(4096<<2);
    try {
        tcldbSocket = new Socket(host,port);
        tcldbInputStream = tcldbSocket.getInputStream();
        tcldbBufferedReader = new BufferedReader(
                              new InputStreamReader(tcldbInputStream));
        tcldbOutputStream = new PrintStream(tcldbSocket.getOutputStream());
        tcldbSocket.setSoTimeout(5000);
    } catch (Exception e) {
        e.printStackTrace();
    }
    listener = new TclDBListener(this, tcldbInputStream);
    listener.setPriority(Thread.MIN_PRIORITY);
    messageHandler = new TclDBMessageHandler(this);
    listener.setMessageHandler(messageHandler);
    listener.start();
    escher = new EscherCommand(this);
    table  = new TableCommand(this);
    finger = new FingerCommand(this);
}
public boolean login (String name, String pwd)
    throws TclDBEmptyMessageException
{
    String res = execR("login " + name + " " + pwd + " " + "ibiza");
    if (res.equals("1")) {
        return true;
    }
    else {
        return false;
    }
}

/**
 * Liefert das Ergebnis des Befehls an das aufrufende Objekt Zurück.
 * Kommt keine Antwort für eine bestimmte endliche Zeit, so wird ein
 * Exception weitergeleitet.
```

```
 */

public synchronized String execR(String cmd)
    throws TclDBEmptyMessageException
{
    String res=null;
    lock();
    send(cmd);
    try {
        for (int i = 0; i < 19 ; i++) {
            if (!tcldbBufferedReader.ready()) {
                wait(500);
            } else {
                res = messageHandler.getResponde(tcldbInputStream);
                break;
            }
        }
    }
    catch (IOException e1) {
        e1.printStackTrace();
    }
    catch (Exception ex) {
        unlock();
        ex.printStackTrace();
    }
    unlock();
    return res;
}




/**
 * Schicke einen Befehl an dem Server und warte 5 Sekunden auf eine
 * Antwort. Die Nachricht wird vom MessageHandler gelesen, und durch die
 * Visualisierungsmethode des aufrufenden Objekts angezeigt.
 */
public synchronized void execV (Tableview sender, String cmd)
    throws
        Exception, TclDBEmptyMessageException,
        ParserConfigurationException
{
    boolean gotMessage = false;
    lock(); // listener shouldn't read the message.
    send(cmd);
    try {
        for (int i = 0; i < 19 ; i++) {
            if (!tcldbBufferedReader.ready()) {
                wait(500);
```

```
                } else {
                    gotMessage = true;
                    messageHandler.handleMessage(sender,tcldbInputStream);
                    break;
                }
            }
        }
        catch (Exception ex) {
            unlock();
            throw ex;
        }
        if (!gotMessage) {
            throw new TclDBEmptyMessageException();
        }
        unlock();
    }

    /**
     * Schicke einen Befehl an dem Server und versuche die Antwort zu lesen.
     * Kommt keine Antwort, wird, nach 10 Versuchen, eine
     * TclDBEmtyServerMessageException "geworfen"
     */
    public synchronized String sendR(String cmd)
        throws TclDBEmptyMessageException
    {
        String res = null;
        lock();
        try {
            send(cmd);
            for (int i = 0; i < 19 ; i++)
                if (!tcldbBufferedReader.ready()) {
                    wait(500);
                } else {
                    res = read();
                    break;
                }
        }
        catch (InterruptedException e) {}
        catch (TclDBEmptyMessageException e) {}
        catch (IOException e) {}
        unlock();
        if (res == null)
            throw new TclDBEmptyMessageException();
        else
            return res;
    }
}
```

Programm C.7 *TclDBMessageHandler.viewMessage:*

```
private synchronized void viewMessage(Tableview viewObj,
                                      TclDBHeader header,
```

```
                                        WrappedInputStream wrappedReader)
{
    int msgType = header.getMessageType();
    org.jdom.Document errorDoc = null;
    switch (msgType) {
    case TclDBHeader.SYSTEM_TYPE: break;
    case TclDBHeader.RESPONDE_TYPE: break;
    case TclDBHeader.ERROR_TYPE:
        try {
            errorDoc = readMessageBodyJDOM(wrappedReader);
        }
        catch (JDOMException ex) {
            ex.printStackTrace();
        }
        errorHandler.handleError(errorDoc, (JFrame) viewObj);
        break;
    case TclDBHeader.UNSUCCESSOP_TYPE:
        try {
            errorDoc = readMessageBodyJDOM(wrappedReader);
        }
        catch (JDOMException ex) {
            ex.printStackTrace();
        }
        viewObj.handleUnsuccessOp(errorDoc);
        break;
    case TclDBHeader.AWARENESS_TYPE:
        org.jdom.Document jdomdoc = null;
        try {
            jdomdoc = readMessageBodyJDOM(wrappedReader);
        }
        catch (JDOMException ex) {
            ex.printStackTrace();
        }
        viewObj.viewAwarenessInfo(jdomdoc);
        break;
    case TclDBHeader.VIEW_TYPE:
        InputSource in = new InputSource(wrappedReader);
        org.w3c.dom.Document domdoc = null;
        try {
            String tid = viewObj.getId();
            domdoc = readMessageBody(in);
            viewObj.setView(domdoc);
        }
        catch (TransformerConfigurationException ex) {
            ex.printStackTrace();
        }
        catch (ParserConfigurationException ex) {
            ex.printStackTrace();
        }
        catch (TransformerException ex) {
            ex.printStackTrace();
```

```
        }
        catch (SAXException ex) {
            ex.printStackTrace();
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
        break;
    case TclDBHeader.MERGE_TYPE:
        InputSource mergeIn = new InputSource(wrappedReader);
        org.w3c.dom.Document changes = null;
        try {
            String tid = viewObj.getId();
            changes = readMessageBody(mergeIn);
            viewObj.mergeView(changes);
        }
        catch (TransformerConfigurationException ex) {
            ex.printStackTrace();
        }
        catch (ParserConfigurationException ex) {
            ex.printStackTrace();
        }
        catch (TransformerException ex) {
            ex.printStackTrace();
        }
        catch (SAXException ex) {
            ex.printStackTrace();
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Programm C.8 *Vtable:*

```
package tcldbc;

/**
 * Implementiert ein Vtable-Objekt bei dem Client. Kommandos werden
 * an die entsprechende Vtable-Instantz des VClients weitergeleitet.
 */
public class Vtable
{
    /** table id.*/
    private String id;
    private TclDBSocket tcldb;
    /** Visualisierungsobjekt der zugehörigen Tabelle. */
    private Tableview tableview;
    /**
     * Konstruktor.
     */
```

```
public Vtable (Tableview tv, TclDBSocket tcldb, String id)
{
    this.tcldb = tcldb;
    this.id = id;
    this.tableview = tv;
}

/** Zeichne in der Höhe von <atomics> atomare attribute*/
public void draw (int atomics)
    throws Exception, TclDBEmptyMessageException
{
    tcldb.execV(tableview, id + " draw " + atomics);
}

public void setAwarenessMode (String mode)
    throws Exception, TclDBEmptyMessageException
{
    tcldb.execV(tableview, id + " setAwarenesseMode " + mode);
}

/** Erzeuge einen neuen Finger.*/
public void newfinger()
    throws Exception, TclDBEmptyMessageException
{
    tcldb.execV(tableview, id + " newfinger");
}

/** push */
public void push ()
    throws Exception, TclDBEmptyMessageException
{
    tcldb.execV(tableview, id + " push");
}
// pop, next  ...

public void insertAfter ()
    throws Exception, TclDBEmptyMessageException
{
    tcldb.execV(tableview, id + " insertAfter");
}

// insertBefore, delete  ...

public boolean editAtomic ()
    throws Exception, TclDBEmptyMessageException
{
    String res = tcldb.execR(id + " editAtomic");
    if (res.equals("1")) {
        return true;
    }
    return false;
```

```
    }

    public boolean updateAtomicValue (String value)
        throws Exception, TclDBEmptyMessageException
    {
        String res = tcldb.execR(id + " updateAtomicValue " + value);
        if (res.equals("1")) {
            return true;
        }
        return false;
    }

    public void setAtomicValue (String value)
        throws Exception, TclDBEmptyMessageException
    {
        tcldb.execV(tableview, id + " setAtomicValue \"" + value + "\"");
    }

    /**
     * Beende das Editieren ohne den Wert zu ändern. Gebe Sperre frei.
     */
    public void endAtomicEditing ()
    {
        tcldb.exec(id + " endAtomicEditing");
    }

    public String get()
        throws Exception, TclDBEmptyMessageException
    {
        return tcldb.execR(id + " getAtomic");
    }
}
```

Programm C.9 *AbstractTableView:*

```
/**
 * Implementiert den generischen Teil eines Tableview-Objekts.
 */

public abstract class AbstractTableview
                      extends JFrame implements Tableview {

    protected String tableId;
    protected DomNavigation domNavigation;
    protected TclDBMessageHandler messageHandler;
    protected AwarenessViewer awarenessViewer;
    protected Nf2Viewer nf2Viewer;
    ...
```

```
public AbstractTableview (String name, TclDBSocket tdb) {
    super(name);
    this.tableName = name;
    this.tcldb = tdb;
    this.table = tcldb.getTableCommand();
    try {
        this.tableId = table.open(tableName);
    }
    catch (TclDBEmptyMessageException ex) {...}
    // Erzeuge domNavigation bei dem ersten Aufruf von setView.
    domNavigation = null;
    this.messageHandler = tcldb.getMessageHandler();
    this.vtable = new Vtable(this, tcldb, tableId);
    // Registriere dieses Objekt als Visualisierungsklasse
    // für die Tabelle <tid>.
    messageHandler.setTableView(tableId, this);
    // Verwende Standard-Stylesheet für die Visualisierung.
    try {
        messageHandler.setTransformer(this.tableId);
    } catch (TclDBEmptyMessageException ex){...}
}

public void setView(org.w3c.dom.Document newView)
    throws ParserConfigurationException,
        TransformerException, IOException {
    if (domNavigation == null)
        domNavigation = new DomNavigation(newView);
    else
        domNavigation.setTree(newView);
    viewTable(messageHandler.transformMessage(
              domNavigation.getTree(), tableId));
}

public void mergeView(org.w3c.dom.Document changes)
    throws ParserConfigurationException,
        TransformerException, IOException {
    domNavigation.mergeChanges(changes);
    viewTable(messageHandler.transformMessage(
              domNavigation.getTree(), tableId));
}

public void viewTable(org.w3c.dom.Document visualNode)
    throws IOException, TransformerConfigurationException,
        TransformerException {
    nf2Viewer.viewTable(visualNode);
}

public void viewAwarenessInfo(org.jdom.Document doc) {
    awarenessViewer.redraw(doc);
}
```

```
    public void push() throws Exception,
        TclDBEmptyMessageException {}

    public void insertAfter() throws
        Exception, TclDBEmptyMessageException {}
    ...
}
```

Programm C.10 *`TclDBTransformer` verwendet den xalan-Stylesheet-Prozessor für die Transformierung von XML-Dokumenten in andere Formate (HTML, SVG):*

```
public class TclDBTransformer {
    private final String STANDARD_STYLESHEET = "nf2tohtml";
    private String name; //Name des Stylesheets
    private String xsltCode;
    private Transformer transformer;
    private TclDBSocket tcldbSocket;

    /**
     * Erzeuge ein Transformer-Objekt aus dem Stylesheet.
     */
    public  TclDBTransformer (TclDBSocket tcldb, String name)
            throws TclDBEmptyMessageException {
        this.tcldbSocket = tcldb;
        this.name = name;
        xsltCode = tcldbSocket.sendR("getstylesheet " + name);
        transformer = createTransformer(xsltCode);
    }

    public void transform(Source from, Result nodeResult) throws
         TransformerException {
        transformer.transform(from, nodeResult);
    }

    private synchronized
        Transformer createTransformer(String stylesheet) {
        Transformer tableTransformer = null;
        TransformerFactory tfactory = TransformerFactory.newInstance();
        if (tfactory.getFeature(SAXTransformerFactory.FEATURE)) {
            try {
                SAXTransformerFactory stfactory =
                                    ((SAXTransformerFactory) tfactory);
                TemplatesHandler handler = stfactory.newTemplatesHandler();
                XMLReader reader =  new  LocalSaxParser();
                reader.setContentHandler(handler);
                StringReader strR = new StringReader(stylesheet);
                InputSource src = new InputSource(strR);
                reader.parse(src);
                Templates templates = handler.getTemplates();
                tableTransformer = templates.newTransformer();
            }
            catch (Exception ex1) {...}
```

```
        }
        return tableTransformer;
    }
}
```

Programm C.11 *DomNavigation:*

```
public class DomNavigation  {
    private Document domTree;
    ...
    public DomNavigation (Document d) {
        this.domTree = d;
    }
    public void mergeChanges(Document changes) {
        Element rElement = changes.getDocumentElement();
        String  attrName = rElement.getAttribute("attr");
        Element treeElement = domTree.getDocumentElement();
        unsetAttr(treeElement, attrName);
        NodeList cNodes = rElement.getChildNodes();
        if(cNodes != null) {
            String id, oldValue, newValue;
            for (int i = 0; i < cNodes.getLength(); i++) {
                if (cNodes.item(i).getNodeType() == Node.ELEMENT_NODE) {
                    Element cElement = (Element) cNodes.item(i);
                    id = cElement.getAttribute("id");
                    Element newValueElement = (Element)
                            cElement.getElementsByTagName("newvalue").item(0);
                    newValue = newValueElement.getFirstChild().getNodeValue();
                    Element changeNode = getElementById(treeElement, id);
                    if (changeNode != null) {
                        setAttrs(changeNode, attrName,  newValue);
                    }
                }
            }
        }
    }

    private void setAttrs(Element node, String attr, String newValue) {
        NodeList childNodes = node.getChildNodes();
        if (childNodes != null) {
            for (int i = 0; i < childNodes.getLength(); i++) {
                if (childNodes.item(i).getNodeType() == Node.ELEMENT_NODE) {
                    Element element = (Element) childNodes.item(i);
                    setAttrs(element, attr, newValue);
                    element.setAttribute(attr, newValue);
                }
            }
        }
        Element elementNode = (Element) node;
        elementNode.setAttribute(attr, newValue);
    }
```

```
    private void unsetAttr (Element node, String attr) {
        setAttrs(node, attr, "none");
    }
    public void setTree(Document newTree) {
        domTree = newTree;
    }
}
```

Programm C.12 *Ausschnitte aus den Klassen* `TableAwarenessInfo` *und* `AwarenessPanel`*:*

```
public class TableAwarenessInfo {
    private Document document; // Awareness-Information als jdom
    private Vector users;      // Liste der Benutzer
    public TableAwarenessInfo(org.jdom.Document doc) {
        users = new Vector();
        reRead(doc); // neues Dokument parsen
    }
    // Ersetze das Awareness-Dokument
    public void reRead(org.jdom.Document doc) {
        this.document = doc;
        root = document.getRootElement();
        users.removeAllElements();
        for (Iterator i=root.getChild("users").getChildren().iterator();
             i.hasNext();) {
            users.addElement((Element)i.next());
        }
    }
    // Liefert eine Liste mit allen Benutzer Id's
    public Vector getUsers() {...}
}


public class AwarenessPanel extends JPanel implements AwarenessViewer {
    private TableAwarenessInfo awarenessInfo;
    private JTextArea viewport;
    public void draw() {
        Vector users =  awarenessInfo.getUsers();
        String hName, uName, uId;
        for (Enumeration e = users.elements(); e.hasMoreElements();) {
            uId = (String) e.nextElement();
            if (uId != null) {
                hName = awarenessInfo.getHost(uId);
                uName = awarenessInfo.getName(uId);
                drawUser(uId, uName, hName);
            }
            viewport.repaint();
        }
    }
    public void redraw(org.jdom.Document doc) {
        awarenessInfo.reRead(doc);
        redraw();
```

```
    }
    private void drawUser(String uId, String  uName, String hName) {
        viewport.append("  "  + uId   + "           " +...)
    }
}
```

C.5 Clientbeispiel

Programm C.13 *escherClient:*

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import tcldbc.*;

public class escherC06 extends JApplet
    implements ActionListener
{
    private final String theHost ="elba.db.informatik.uni-kassel.de";
    private final int thePort = 9009;
    private TclDBSocket tcldb;
    private JPanel atsList;
    private boolean inAnApplet = true;

    public escherC06() {
        this(true);
    }

    public escherC06(boolean inAnApplet) {
        this.inAnApplet = inAnApplet;
        if (inAnApplet) {
            getRootPane().putClientProperty("defeatSystemEventQueueCheck",
                                            Boolean.TRUE);
        }
    }

    public void init () {
        setContentPane(makeContentPane());
    }

    public Container makeContentPane () {
        try {
            UIManager.setLookAndFeel(
                      UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { }
        this.setFont(new Font("Times",Font.BOLD,14));
        JPanel mainFrame = new JPanel();
```

```
        JPanel bottomPane = null;
        JMenuBar menuBar = null;
        TclDBLoginDialog.initialize(null, "Login");
        try {
            tcldb  = new TclDBSocket(theHost,thePort);
            int i;
            String userName, passwd;
            boolean loggedIn = false;
            for (i = 0; i < 3; i++) {
                TclDBLoginDialog.showDialog(this);
                userName = TclDBLoginDialog.getUserName();
                passwd = TclDBLoginDialog.getPwd();
                if (userName.equals("")) {
                    System.out.println("login failed, exit.");
                    System.exit(1);
                }
                if (tcldb.login(userName,passwd)) {
                    loggedIn = true;
                    break;
                }
            }
            if (!loggedIn) {
                System.out.println("login failed, exit.");
                System.exit(1);
            }
            menuBar = createMenuBar();
            bottomPane = createBottomButtons();
            bottomPane.setMinimumSize(bottomPane.getPreferredSize());
            atsList = new TclDBMainPanel(tcldb);

        } catch (TclDBEmptyMessageException ex) {
            ex.printStackTrace();
        } catch (Exception ex2) {
            ex2.printStackTrace();
        }
        mainFrame.setLayout(new BoxLayout(mainFrame,BoxLayout.Y_AXIS));
        mainFrame.add(menuBar);
        mainFrame.add(atsList);
        mainFrame.add(bottomPane);
        return mainFrame;
    }

    public void destroy () {
        try {
            tcldb.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
```

```
        JFrame frame = new JFrame("escherClient");
        frame.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    System.exit(0);
                }
            });
        escherC06 applet = new escherC06(false);
        frame.setContentPane(applet.makeContentPane());
        frame.pack();
        frame.setVisible(true);
    }

    private JPanel createBottomButtons ()
    {
        JPanel bottom = new JPanel();
        bottom.add(Box.createHorizontalGlue());
        return bottom;
    }

    private JMenuBar createMenuBar()
    {
        JMenuBar menuBar = new JMenuBar();
        JMenu menu;
        JMenuItem item;
        String [] systemItems = new String [] {"Options", "Test", "Exit"};
        String [] schemaItems = new String [] {"Open", "New", "Delete"};
        String [] tableItems  = new String [] {"Open", "New", "Delete"};
        String [] applicationItems  = new String [] {"New", "Delete", "Readra"};
        String [] helpItems  = new String [] {"Help",
                                              "Contents", "about jescherc"};
        int i = 0;
        menu = new JMenu("System");
        for (i = 0; i < systemItems.length; i++) {
            item = new JMenuItem(systemItems[i]);
            item.addActionListener(this);
            menu.add(item);
        }
        menuBar.add(menu);
        menu = new JMenu("Schema");
        for (i = 0; i < schemaItems.length; i++) {
            item = new JMenuItem(schemaItems[i]);
            item.addActionListener(this);
            menu.add(item);
        }
        menuBar.add(menu);
        menu = new JMenu("Table");
        for (i = 0; i < tableItems.length; i++) {
            item = new JMenuItem(tableItems[i]);
            item.addActionListener(this);
            menu.add(item);
        }
```

```
        menuBar.add(menu);
        menu = new JMenu("Application");
        for (i = 0; i < applicationItems.length; i++) {
            item = new JMenuItem(applicationItems[i]);
            item.addActionListener(this);
            menu.add(item);
        }
        menuBar.add(menu);
        menuBar.add(Box.createHorizontalGlue());
        menu = new JMenu("Help");
        for (i = 0; i < helpItems.length; i++) {
            item = new JMenuItem(helpItems[i]);
            item.addActionListener(this);
            menu.add(item);
        }
        menuBar.add(menu);
        return menuBar;
    }

    public void actionPerformed(ActionEvent e)
    {
        String cmd = e.getActionCommand();
        if (cmd.equals("Exit")) {
            System.exit(0);
        } else {
            notJetImplemented(cmd);
        }
    }
}
```