Kooperative Interaktionsunterstützung in Groupware

Matthias Ressel
Universität Stuttgart

Zusammenfassung

Sogenannte Groupware unterstützt die Kooperation, Kommunikation und Koordination von Personen, die gemeinsam eine Aufgabe bearbeiten. Bei zahlreichen Anwendungen ist es wichtig, die Interaktion der Kooperationspartner mit Hilfe des Computers koordinierend zu unterstützen. Die vorliegende Arbeit zeigt, daß die Methode der Operationstransformationen zur Implementierung von Gruppeneditoren unter softwareergonomischen Gesichtspunkten besonders geeignet ist. Sie zeigt, welche Eigenschaften Transformationsregeln erfüllen müssen, um als Grundlage einer benutzerorientierten, kooperativen Koordinationsmethode zu dienen. Solche Transformationen ermöglichen die Erstellung eines einheitlichen Interaktionsmodells, das als Grundlage weiterer unterstützender Module, wie etwa für Gruppen-Undo, dient.

1. Einleitung

Rechnerprogramme, die vereinfachend als Groupware [6] bezeichnet werden, sollen *Kooperation*, also die Zusammenarbeit einer Gruppe von Personen, die gemeinsam an einer Aufgabe arbeiten, unterstützen. Voraussetzung für eine solche Kooperation sind gemeinsame Ziele, gemeinsame Handlungspläne, ein gemeinsamer Kontext, flexible Regelbarkeit und Kontrollierbarkeit [9]. *Koordination* dient dazu, die einzelnen Handlungen zeitlich und inhaltlich aufeinander abzustimmen. Sie soll vor allem die Interaktion innerhalb der Gruppe unterstützen. *Kommunikation* der Kooperationspartner untereinander dient übergreifend zur Einigung über Ziele, zur Herstellung eines gemeinsamen Kontextes und allgemein zur Unterstützung der Interaktion.

Interaktionsunterstützung (IU) läßt sich charakterisieren durch die Eigenschaften vorausschauend, begleitend und rückblickend. *Vorausschauende IU* gibt feste Handlungspläne vor, an die sich die Kooperationspartner zu halten haben. Sie zielt vor allem auf präventive Konfliktvermeidung. *Begleitende IU* erlaubt die Anpassung der Koordinationsstrategie an die tatsächlichen Gegebenheiten. Insbesondere kann sie auf Ausnahmesituationen reagieren. Ihr Schwerpunkt liegt auf der Konflikterkennung und Konfliktbehebung. *Rückblickende IU* schließlich erlaubt es, Einblick auf bereits ausgeführte Handlungen zu nehmen, um daraus zukünftige Koordinationsstrategien abzuleiten.

Die Koordination kann von den Partnern selbst (*intern*) durchgeführt werden. Sie kann aber auch durch eine koordinierende Instanz von außen (*extern*) unterstützt werden. Dieser Koordinator kann ein Mensch, aber auch ein Computer sein.

Im folgenden wird ein Ansatz vorgestellt, der vor allem zur begleitenden und rückblickenden IU in Groupware mit direkt manipulativer Benutzungsoberfläche geeignet ist und der sowohl interne als auch externe Koordination unterstützt. Zuvor definieren wir die wichtigsten Anforderungen und stellen weitere bekannte Ansätze vor. Unser eigener Ansatz verwendet die Methode der Operationstransformationen und erweitert sie um ein einheitliches Interaktionsmodell. Dieses Modell dient als Grundlage weiterer interaktionsunterstützender Module, die u. a. Undo für Gruppen und das nachträgliche Navigieren im Interaktionsraum erlauben. Die prototypische Realisierung eines Texteditors für Gruppen und seine Benutzungsoberfläche werden vorgestellt; ein Vergleich mit anderen vorliegenden Arbeiten und ein Ausblick runden diesen Beitrag ab.

2. Problembeschreibung und Anforderungen

Als Szenario dient die Dokumenterstellung durch mehrere Autoren (*co-authoring*). Dies kann z. B. die Fertigstellung eines Projektantrages, das Verfassen eines Abschlußberichtes, die Erstellung von Bedienungsanleitungen oder einfach das Schreiben eines gemeinsamen Briefes sein; auch die Software-Entwicklung im Team gehört i. w. S. zu diesem Anwendungsbereich.

Solche Anwendungen weisen einige gemeinsame Merkmale auf:

1. Mehrere Personen arbeiten an einer Aufgabe zusammen.
2. Die Personen befinden sich nicht notwendig alle an einem Ort.
3. Die einzelnen Teiltätigkeiten (Schreiben, Zeichnen, Modifizieren, Layout) werden bereits mit einem Computer durchgeführt.
4. Die Tätigkeit wird meist über einen längeren Zeitraum ausgeführt, muß aber zu einem bestimmten Zeitpunkt beendet sein.

Eine wichtige Voraussetzung für eine solche Zusammenarbeit ist das gegenseitige Gewahrsein (*awareness*). Solches Gewahrsein entsteht einerseits durch *explizite Kommunikation* miteinander, anderseits aber auch durch *implizite Kommunikation*: Hierzu gehört die bewußte oder unbewußte Wahrnehmung einer Person und ihrer Tätigkeiten, aber auch das Wahrnehmen von *Spuren*, die andere Personen „in der Welt" hinterlassen haben. Gewahrsein liefert also die Antwort auf Fragen wie: *Wer* wird was machen? *Wer* macht gerade was? *Wer* hat was gemacht? Dieses Wissen ist

wichtig für die interne Koordination, d. h. es erlaubt den beteiligten Personen, ihre Handlungen gegenseitig abzustimmen.

Vielfach reicht Gewahrsein aber nicht aus, um Konflikte bei der Interaktion auszuschließen. Durch gleichzeitige, parallele Zugriffe kann es zu inkonsistenten Zuständen von Anwendungsobjekten kommen. Beispiele für Konflikte sind das gleichzeitige Löschen von Objekten, oder widersprüchliches Ändern von Objekteigenschaften. Groupware sollte auf solche Inkonsistenzen aufmerksam machen bzw. helfen, sie zu vermeiden. Groupware sollte verschiedene Möglichkeiten des *Undo* bieten: Es muß möglich sein, auf frühere Zustände zurückzusetzen oder Teile der Interaktion zurückzunehmen. Es stellen sich hierbei folgende Fragen: *Welche* Auswirkung hat es, wenn mehrere Operationen gleichzeitig ausgeführt werden? *Was* passiert bei Konflikten? *Wann* kann eine Operation wieder zurückgenommen werden?

Für Groupware, die Anwendungen mit den oben beschriebenen Merkmalen unterstützt, ergeben sich folgende spezifischen Anforderungen:

1. Den Autoren soll der aktuelle Stand der Aufgabenbearbeitung zur Verfügung stehen, damit inhaltliche Inkonsistenz und unnötige Mehrarbeit vermieden werden können.
2. Die gewohnte Direktheit der Benutzungsschnittstelle darf auch bei räumlich verteiltem Betrieb nicht verloren gehen. Insbesondere müssen solche Aktionen wie Einfügen und Löschen von Zeichen unmittelbar auf dem eigenen Bildschirm zu sehen sein.
3. Das kooperative System sollte nicht allzusehr von den gewohnten Werkzeugen (Editoren) abweichen, die für die Teiltätigkeiten benutzt werden. Auch sollte ein nahtloser Übergang zwischen Ein- und Mehrbenutzerbetrieb ermöglicht werden.
4. Der Zeitdruck, der regelmäßig gegen Ende einer solchen Tätigkeit entsteht, macht eine engere Zusammenarbeit nötig. Parallele Interaktionen – wie gleichzeitiges Editieren einer Datei – sollten daher möglich sein, ohne zu undefinierten Ergebnissen zu führen. Notwendige manuelle Eingriffe sollten sich dabei auf ein Minimum beschränken, damit die Personen nicht von ihrer eigentlichen Tätigkeit des Schreibens (Zeichnens, etc.) abgehalten werden.

Der Schwerpunkt im folgenden wird auf dem letzten Punkt, der parallelen Interaktion, liegen. Es wird sich jedoch zeigen, daß der vorgeschlagene Ansatz zur Unterstützung paralleler Interaktion auch die anderen Anforderungen weitgehend erfüllt.

3. Ansätze zur Interaktionsunterstützung

Zur Koordinierung mehrerer Eingabeströme (*concurrency control*), wie sie bei paralleler Interaktion mehrerer Akteure auftreten, bieten sich verschiedene Ansätze an (vgl. hierzu [6], [1]). Sie unterscheiden sich dadurch, ob

- Konflikte von vornherein vermieden werden oder
- Konflikte bei ihrem Auftreten erkannt und geeignet weiterbehandelt werden.

3.1. Methoden zur Konfliktvermeidung

Konflikte durch konkurrierende Operationen können vermieden werden, indem der Zustandsraum der Anwendung *partitioniert* wird. Verschiedene Personen können dabei gleichzeitig an verschiedenen Objekten einer Anwendung arbeiten. Ein Beispiel hierfür ist *Hypertext*, der Informationen in Netzknoten aufteilt. Ein weiteres Beispiel sind graphische Zeichenprogramme, die für mehrere Benutzer unterschiedliche übereinandergeschichtete, transparente Zeichenebenen (*Layers*) bereitstellen. Gleichzeitiges Editieren desselben Objektes ist hierbei ebenso unmöglich wie das Editieren eines fremden Objekts. Das Ausmaß der Interaktion wird dadurch sehr eingeschränkt.

Konflikte können durch *Kopieren* der Anwendungsobjekte, wenn nicht vermieden, so zumindest hinausgezögert werden. Dieses Verfahren wird beispielsweise in Versionsverwaltungssystemen eingesetzt. Es gilt dabei, aus mehreren Kopien parallel entwickelter Anwendungsobjekte eine einheitliche, konsistente Version herzustellen. Da nicht sichergestellt ist, daß dies überhaupt möglich ist, ist meist eine manuelle Nachbearbeitung erforderlich.

Weitere Verfahren beruhen darauf, den Zugriff auf Anwendungsobjekte nur sequentiell zuzulassen; *feste Dialogabläufe* werden vorgegeben. Dies wird häufig bei Spielprogrammen verwirklicht, bei denen die Spielpartner abwechselnd am Zug sind.

Sperrmechanismen (*Locking*) reservieren Teile des Anwendungsbereichs zeitweise für einen exklusiven Zugriff; sie stellen daher eine dynamische Art der Partitionierung dar. Die Sperren können entweder explizit angefordert werden oder sie werden durch die Ausführung bestimmter Aktionen implizit eingerichtet. Um zu vermeiden, daß bestimmte Benutzer Sperren unnötig lange halten – etwa bei Abwesenheit – können solche Sperren nach einer gewissen Zeit, in der der

Sperrenhalter inaktiv war, automatisch (*tickle locks*) oder auf Anfrage durch andere Benutzer (*soft locks*) entfernt werden. Sperren haben allerdings den Nachteil, daß sie – zumindest zeitweise – den freien Zugriff auf Anwendungsobjekte einschränken und damit unsichere modale Zustände erzeugen. Eine visuelle Hervorhebung gesperrter Objekte kann dazu beitragen, diese Probleme zu reduzieren.

Eine andere Möglichkeit ist die explizite Zuteilung eines Schreib- oder sonstigen Zugriffsrechtes (*floor control*). Soziale Protokolle mittels expliziter Absprachen haben den Nachteil, daß sie umgangen werden können, was in diesem Fall zu undefinierten Zuständen des Anwendungssystems führen kann. Sicherer sind technische Protokolle mittels sogenannter *Tokens*, die das Schreibrecht repräsentieren und jeweils bei Bedarf weitergereicht oder zurückgegeben werden können. Tokens stellen eine spezielle Art von Sperren dar, da genau ein Token für einen bestimmten festen Systembereich zuständig ist. Daher können auch bei Floor-Control-Verfahren die von Sperren her bekannten Methoden angewandt werden, um Tokens bei Passivität des Token-Halters wieder freigeben.

Die Sequentialisierung über eine zentralen Koordinator stellt einen weiteren Ansatz dar. Bei parallel abgegebenen Benutzeranforderungen ist hierbei allerdings nicht gewährleistet, daß sie immer im richtigen Zustand ausgeführt werden. Um solche parallele Eingaben zu vermeiden, ist die Benutzungsoberfläche solange zu sperren, bis eine Anforderung ausgeführt werden konnte.

Allgemein läßt sich feststellen, daß Konfliktvermeidung durch Partitionieren, Kopieren oder Sequentialisieren die Vorteile parallelen Arbeitens nicht nutzen kann. Die Aktionsfreiheit des einzelnen geht verloren: „The computer won't let me“ [3].

3.2. Methoden zur Konflikterkennung und Konfliktbehebung

Transaktionsmechanismen, wie sie bei Datenbanken eingesetzt werden, versuchen parallele Anforderungen zu sequentialisieren. Treten dabei Konflikte auf, werden solche Transaktionen zurückgesetzt. Ihr Ergebnis wird erst dann öffentlich zugänglich, wenn die Sequentialisierung erfolgreich verlaufen ist. Zwischenzustände sind dadurch für andere Benutzer nie einsehbar. Die Forderung nach Gewahrsein ist nicht erfüllt.

Weitere Verfahren zeichnen sich dadurch aus, daß Benutzeranforderungen lokal immer sofort ausgeführt werden, ohne vorher zu überprüfen, ob die Anforderung mit anderen kollidiert. Sie werden daher als *optimistische* Verfahren bezeichnet.

Ein einfaches Verfahren dieser Art unterbricht die Verarbeitung, wenn es einen Konflikt erkannt hat. Er muß dann von den Benutzern manuell behoben werden. Dies hat den weiteren Nachteil, daß das System keinen konsistenten Zustand garantieren kann.

Eine Verbesserung stellen solche Verfahren dar, die durch eine Rücknahmemöglichkeit (*undo*) von Operationen und durch eine automatische Umsortierung der Interaktionshistorie für eine Konfliktbehebung sorgen.

Die Methode der *Operationstransformation* schließlich versucht, Konflikte durch parallele Eingabeanforderungen dadurch zu lösen, daß sie bei deren Interpretation und Ausführung den Anwendungszustand berücksichtigt, der zum Zeitpunkt der Anforderung gültig war. Bei Bedarf werden Operationen derart transformiert, daß das Resultat für jede mögliche Ausführungsreihenfolge dasselbe ist.

4. Operationstransformationen

Operationstransformationen wurden zum ersten Mal im Gruppen-Outline-Editor GROVE [5] verwendet. Bei diesem Verfahren arbeitet jeder Benutzer mit einem individuellen Editor an Kopien der Anwendungsobjekte. Die Benutzeranforderungen werden im lokalen Editor unmittelbar ausgeführt und dann an die Editoren der anderen Teilnehmer geleitet. Jeder Anforderung wird Information beigefügt, aus der die Empfänger schließen können, in welchem Zustand sich der Editor vor der Eingabe der Anforderung befand. Diese Zustandsinformation spezifiziert die Anzahl der Operationen jedes Teilnehmers, die bereits ausgeführt worden sind. Die verschiedenen Editoren führen die eingehenden Anforderungen unter Umständen in anderen Reihenfolgen aus. Eine Umsortierung wird dadurch möglich gemacht, daß bei der Ausführung fremder Anforderungen berücksichtigt wird, welche zwischenzeitliche Veränderungen des Anwendungszustands durch zusätzlich ausgeführte Operationen entstanden sind. Unter bestimmten Bedingungen ist es hierdurch möglich, in allen Editoren konsistente Zustände zu erzielen, sobald alle Anforderungen abgearbeitet sind.

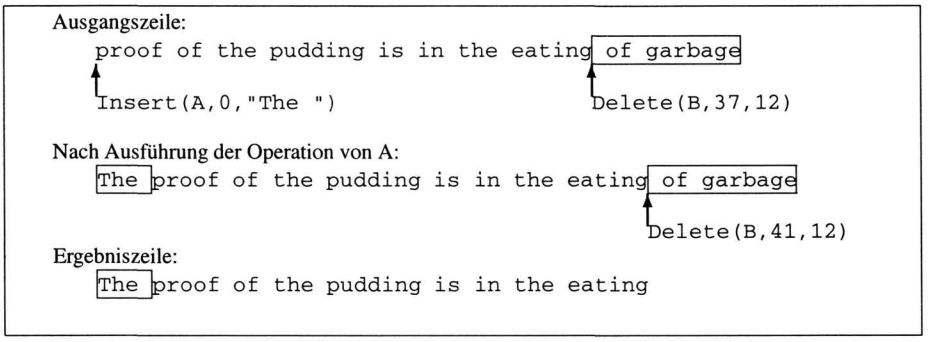

Abb. 1: Synchrones Editieren: Die Einfügeoperation von A muß berücksichtigt werden, wenn die Löschoperation von B danach ausgeführt wird.

Beispiel 1: Zwei Personen, A und B, benutzen gleichzeitig einen Gruppen-Editor. A fügt am Anfang einer Zeile ein Wort ein, gleichzeitig löscht B am Ende derselben Zeile einige Zeichen. Beide Editoren führen zunächst die lokalen Operationen aus. Der Editor von B kann danach das von A eingefügte Wort an derselben absoluten Position einfügen wie A. Der Editor von A hingegen muß beim Löschen der Zeichen berücksichtigen, daß deren Position sich durch die ausgeführte Einfügeoperation verschoben hat (s. Abb. 1).

Wenn, allgemein betrachtet, zwei Operationen o_1 und o_2 im selben Zustand ausgeführt wurden, müssen Operationen o_1' und o_2' so gewählt werden, daß gilt:

$$o_1 o_2' = o_2 o_1', \qquad (1)$$

d. h. o_1 gefolgt von o_2' führt zum selben Resultat wie o_2 gefolgt von o_1'.

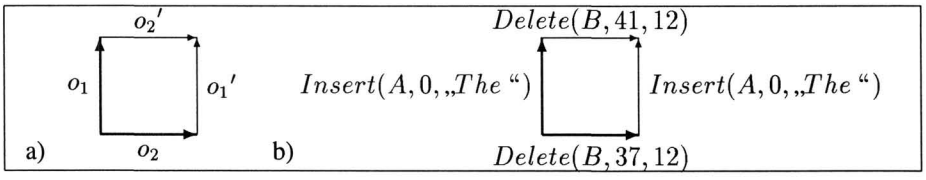

Abb. 2: a) elementare L-Transformation: $o_1 o_2' = o_2 o_1'$, b) Beispieltransformation

Eine Funktion T, die zu zwei Operationen o_1 und o_2 zwei weitere Operationen o_1' und o_1' liefert, so daß Gleichung (1) gilt, wird Transformationsfunktion genannt: $T(o_1,o_2)=(o_1',o_2')$.

Abb. 2a zeigt eine graphische Darstellung. Jede Benutzeranforderung wird dabei durch einen Pfeil repräsentiert. Jedem Benutzer ist eine bestimmte Achse oder Pfeilrichtung zugeordnet. Schließlich erhält jeder dieser Pfeile eine Bezeichnung, die die eigentliche Operation angibt. Operationen, die originale Benutzeranforderungen darstellen, werden i. d. R. fett dargestellt. Auf Grund der Lage der Originalpfeile dieser Operationstransformation, wird sie *L-Transformation* genannt. Die graphische Darstellung des Transformationsschrittes von Beispiel 1 zeigt Abb. 2b.

Gleichung (1) reicht nicht aus, eine L-Transformationsfunktion eindeutig zu definieren. Insbesondere darf der Endzustand einer Transformation nicht von der Reihenfolge der Ausgangsoperationen abhängen.

Beispiel 2 liefert die Definition einer allgemeinen anwendungsunabhängigen L-Transformationsfunktion:

$$T(o_1, o_2) = \begin{cases} (I, \overline{o_1} o_2) & \text{für} \quad p(o_1) > p(o_2), \\ (\overline{o_2} o_1, I) & \text{sonst.} \end{cases} \tag{2}$$

I ist dabei die neutrale Identitätsabbildung, $\overline{o}$ die inverse Operation zu o, deren Existenz vorausgesetzt werden muß, und $p(o)$ die Priorität einer Operation o. Die Priorität berechnet sich aus dem Akteur, der die Operation angefordert hat. Die Akteure lassen sich hierzu auf eindeutige Weise ordnen.

Die Transformation T kann so interpretiert werden, daß eine der parallel ausgeführten Operationen wieder rückgängig gemacht und dafür die andere ausgeführt wird. Welche der beiden Operation rückgängig gemacht wird, ist durch die Priorität der Akteure eindeutig festgelegt. Durch Verzicht auf diese eindeutige Fallunterscheidung wäre diese Transformation nicht wohldefiniert.

Die Abfolge paralleler Interaktionen läßt sich in graphischer Form gut darstellen (s. Abb. 3): Jede Benutzeranforderung wird durch einen Pfeil repräsentiert, dessen Ausgangskoordinaten angeben, wieviele Operationen der einzelnen Benutzer bereits abgearbeitet worden sind.

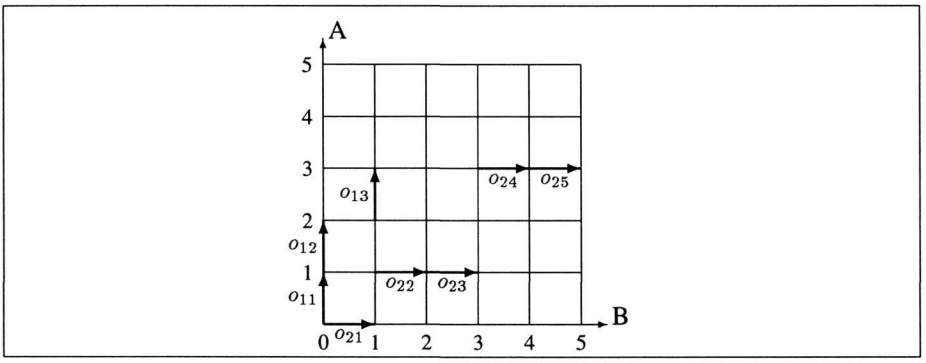

Abb. 3: 2-dimensionales *Interaktionsmodell* mit drei Operationen von Person A und fünf von Person B.

Zwischen den einzelnen Systemzuständen ist eine partielle (zeitliche) Ordnung definiert: Für einen Zustand s_1, dessen Koordinaten alle kleiner oder gleich den jeweiligen Koordinaten eines anderen Zustands s_2 sind, soll gelten: $s_1 \leq s_2$.

Ein *gültige Ausführungsfolge* ist dadurch definiert, daß darin jede Operation im Originalzustand oder einem späteren ausgeführt werden muß. Die Originaloperationen sind dazu bei Bedarf zu transformieren. Abb. 4a zeigt eine mögliche Ausführungsfolge, für die sechs Elementar-Transformationen nötig sind (mit T markierte Quadrate). Abb. 4b zeigt den *Ausführungsraum*, der sich durch Vereinigung aller Ausführungsfolgen ergibt. Damit selbst ausgelöste Operationen möglichst direkt verarbeitet und die Ergebnisse sichtbar werden, werden Operationen i. d. R. lokal sofort ausgeführt. Die obere bzw. untere Begrenzungslinie des Ausführungsraums in Abb. 4b stellt daher typische Ausführungsfolgen für A bzw. B dar.

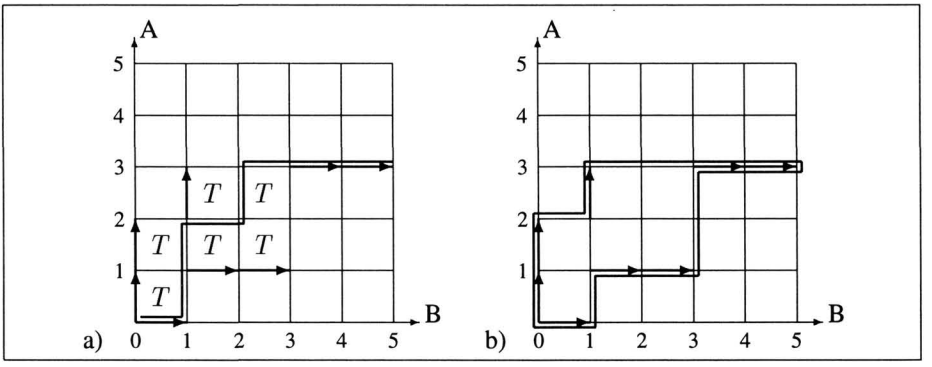

Abb. 4: a) Beispiel einer Ausführungsfolge, b) Ausführungsraum

Im 2-dimensionalen Fall reicht die Transformationsbedingung (1) aus, um zu gewährleisten, daß die Anwendung jeder gültigen Ausführungsfolge auf den Ausgangszustand zum selben Endzustand führt. Der Beweis hierzu läuft induktiv über die Länge der Ausführungsfolge. Bei höheren Dimensionen muß zusätzlich gefordert werden, daß die Transformation einer Operation entlang jeder Ausführungsfolge zur selben resultierenden Operation führt. Dadurch ist gewährleistet, daß jeder Teilschritt einer Ausführungsfolge, der keiner Originaloperation entspricht, eindeutig definiert ist.

Abschätzungen für die Komplexität des Verfahrens ergeben, daß sie im ungünstigsten Fall proportional zum Quadrat der Anzahl der Beteiligten ist. Dies setzt allerdings maximale Parallelität voraus, d. h. auseinanderlaufende Interaktionen, die erst am Ende zusammengefügt werden. In der Praxis wird dies nur dann der Fall sein, wenn die Netzverbindung zeitweise ausfällt.

Aus Benutzersicht ist die Transformationsbedingung (1) nicht hinreichend, da sie auch sinnlose oder triviale Transformationen zuläßt.

Beispiel 3: Eine Transformation T sei definiert durch

$$T(o_1, o_2) = (\overline{o_2}, \overline{o_1}) \tag{3}$$

Diese Transformation bestimmt, daß im Falle zweier gleichzeitig ausgeführter Operationen beide rückgängig gemacht werden. Das Pathologische dieser Transformation zeigt sich z. B. dann, wenn einer der Partner seine Operation

unmittelbar rückgängig macht. Als resultierende Operation ergibt sich o_1 und nicht etwa o_2, was intuitiv eher zu erwarten wäre (s. Abb. 5a).

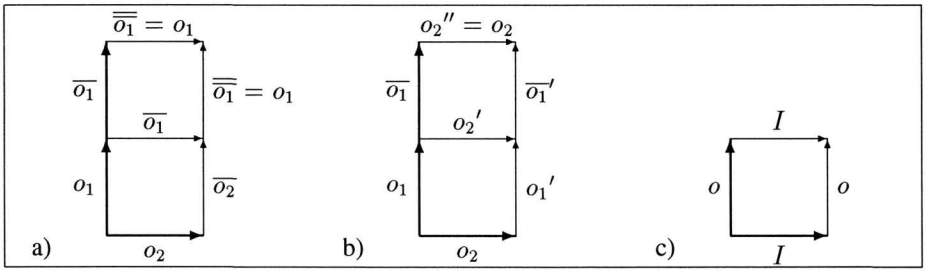

Abb. 5: a) Zu Beispiel 3: $T(o_1,o_2)=(\overline{o_2},\overline{o_1})$, b) Undo-Eigenschaft, c) Neutralitätseigenschaft

Aus Benutzersicht ist das Verhalten gegenüber zurückgenommenen Operation sehr bedeutend. Die unmittelbare Rücknahme von Operationen soll häufig gerade vermeiden, daß Operationen von anderen dadurch beeinflußt werden.

Eigenschaft 1 (*Undo-Eigenschaft*): Sei $\overline{o_1}$ die inverse Operation zu o_2, dann soll für alle Operationen o_1 gelten (Abb. 5b):

$$\text{Wenn } T(o_1, o_2) = (o_1', o_2') \text{ und } T(\overline{o_1}, o_2') = (\overline{o_1}', o_2''), \text{ dann } o_2 = o_2'' \qquad (4)$$

Eine einfachere Bedingung, die an Transformationen gestellt werden kann, besteht darin, daß sich die Identitätsabbildung neutral bezüglich Transformationen verhält.

Eigenschaft 2 (*Neutralitätseigenschaft*): Sei I die neutrale Identitätsabbildung, dann soll für alle Operationen o gelten (Abb. 5c):

$$T(o, I) = (o, I) \qquad (5)$$

Für jede Anwendung müssen spezielle Transformationsregeln gefunden werden, die diese Eigenschaften erfüllen. Falls solche Regeln überhaupt existieren, müssen sie nicht eindeutig sein. Stößt das Computersystem bei den Transformationen auf solche, nicht eindeutig zu klärende Interaktionskonflikte, kann es zunächst eine der Alternativen auswählen und somit den Benutzern das Weiterarbeiten gestatten. Zusätzlich kann das System in solchen Ausnahmefällen die Benutzer warnen und alternative Interaktionen vorschlagen. Durch diese Vorgehensweise werden die

Benutzer möglichst wenig bei ihrer Arbeit unterbrochen. Klärende Dialoge können nachträglich auf Benutzerwunsch angestoßen werden.

Die Attraktivität der Operationstransformationen aus Benutzersicht liegt in der Unterstützung von sowohl interner als auch externer Koordination der Interaktion. Das Computersystem sorgt zunächst für syntaktische Konsistenz. Dazu erhalten die Benutzer durch die Visualisierung fremder Interaktionen jederzeit ausreichend Informationen, um selbst auf inhaltliche Konsistenz achten zu können. Die explizite Repräsentierung des Interaktionsmodell ermöglicht den Benutzern, auch nachträglich den Verlauf vergangener Interaktionen zu inspizieren. Außerdem dient das Interaktionsmodell als Grundlage umfangreicher Rücknahmemöglichkeiten. Damit ist eine kooperative Zusammenarbeit von Mensch und Computer bei der Konfliktbewältigung paralleler Interaktion in Groupware weitgehend verwirklicht.

5. Transformationsregeln für Texteditoren

Die grundlegenden Operationen eines Texteditors sind das Einfügen und das Löschen von Einzelzeichen. Komplexere Operationen lassen sich durch deren Zusammensetzung erzeugen. Wir definieren drei Operationen: *Ins*, *Del* und *Seq*. Eigentlich wären nun $3^2 = 9$ Transformationsregeln zu definieren, für jede Kombination der drei Operationen eine. Dies ist jedoch nicht nötig. Zum einen wurde bereits oben bemerkt, daß Transformationen nicht von der Reihenfolge ihrer Argumente abhängen dürfen. Es reicht also, für ungleiche Paare von Operationen nur die eine Richtung zu definieren. Wird die andere Richtung benötigt, werden die Operation vor und nach der Transformation einfach vertauscht.

Wir definieren (für Personen A und B):

$$T(Ins[A,p_1,c_1],Ins[B,p_2,c_2]) = \begin{cases} (Ins[A,p_1,c_1],Ins[B,p_2+1,c_2]), \\ \quad \text{falls } p_1 < p_2 \text{ oder} \\ \quad p_1 = p_2 \text{ und } prio(A) < prio(B), \\ (Ins[A,p_1+1,c_1],Ins[B,p_2,c_2]), \\ \quad \text{sonst.} \end{cases} \tag{6}$$

$$T(Del[A,p_1,c_1],Del[B,p_2,c_2]) = \begin{cases} (I,I), \\ \quad \text{falls } p_1 = p_2 \text{ und damit } c_1 = c_2, \\ (Del[A,p_1,c_1],Del[B,p_2-1,c_2]), \\ \quad \text{falls } p_1 < p_2, \\ (Del[A,p_1-1,c_1],Del[B,p_2,c_2]), \\ \quad \text{sonst.} \end{cases} \tag{7}$$

$$T(Seq[o_1, o_2], o) = (Seq[o_1', o_2'], o''),$$
$$\text{wobei } T(o_1, o) = (o_1', o') \text{ und } T(o_2, o') = (o_2', o''). \tag{8}$$

Die Definition einer Regel für das Paar *Ins* und *Del* erfolgt entsprechend.

Diese Definitionen unterscheiden sich an einigen Punkten von jenen in [5]. Zwei Einfügeoperationen desselben Zeichens an derselben Stelle ergeben hier zwei Zeichen, deren Reihenfolge durch die Priorität der beteiligten Partner eindeutig bestimmt ist. Die Originaldefinition unterdrückt eines der Zeichen und erfüllt damit nicht die Undo-Eigenschaft. Tatsächlich unterscheiden sich die beiden eingefügten Zeichen, da sie von unterschiedlichen Autoren stammen. Gruppeneditoren, die – wie unsere Implementierung – die Herkunft von Textteilen farblich markieren, lassen dies auch visuell erkennen. Sollte der Fall eintreten, daß auf diese Weise versehentlich ein Text doppelt eingetragen wird, ist dies auf einen Blick zu erkennen. Es zeigt sich außerdem, daß auf die komplizierte Prioritätsfunktion, wie sie in [5] für den Fall der gleichzeitigen Einfügung an derselben Stelle durch mehr als zwei Benutzer definiert wird, verzichtet werden kann.

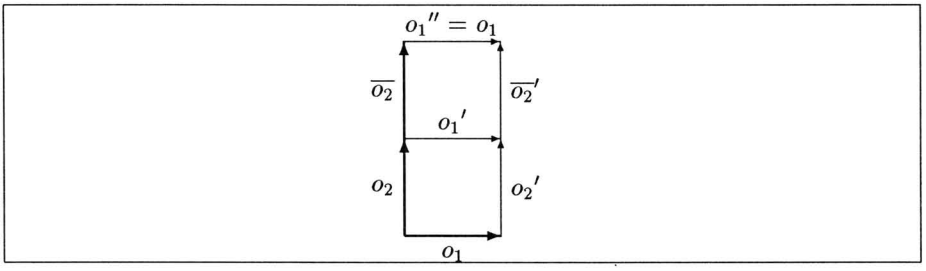

Abb. 6: Kompositionsregel

Die obige Definition (8) einer Transformation für zusammengesetzte Operationen ist neu. Tatsächlich steht sie für fünf einzelne Transformationsregeln. Die Gültigkeit läßt sich leicht anhand des zugehörigen Graphen (s. Abb. 6) ersehen. Diese Regel kann dazu dienen, Einfüge- oder Löschoperationen einer ganzer Zeichenkette als eine elementare Operation aufzufassen. Die Definition (8) garantiert zudem, daß der Effekt derselbe ist, wie wenn unmittelbar aufeinanderfolgende Einzeloperationen ausgeführt würden. Dies führt unter anderem zu der erfreulichen Eigenschaft, daß Einfügungen von mehreren aufeinanderfolgenden Zeichen, die zwei Personen an derselben Position starten, nicht vermischt werden.

Bei der Definition der Transformation eines *Del*-Paares zeigt sich, daß sie einen pathologischen Fall beinhaltet: das Löschen desselben Zeichens. Dieser Fall erfüllt i. d. R. nicht die Undo-Eigenschaft. Wird von einer Partie eine Löschoperation rückgängig gemacht, so wird dies vom System ausgeführt, obwohl eigentlich noch eine weitere, aber bisher unterdrückte Löschoperation vorliegt. Wenn dieser Fall bei einem Transformationsschritt auftritt, sollten die Benutzer informiert werden. Zu beachten ist, daß diese Anomalie am konsistenten Endzustand der einzelnen Editoren nach Ausführung der Löschoperation nichts ändert. Bisher sind keine Transformationsregeln bekannt, die dieses Löschproblem eindeutig lösen.

Allerdings existiert ein anderer Weg, um dieses Problem zu vermeiden. Löschoperationen können auch so interpretiert werden, daß die gelöschten Objekte lediglich als gelöscht markiert und als solche am Bildschirm dargestellt bleiben, etwa durchgestrichen. Ein solches Objekt kann von jeder Person einmal als gelöscht markiert werden. Eine Löschung wird erst dann rückgängig gemacht, wenn alle ihre Löschung zurückgenommen haben. Löschungen sind dann natürlich neutral bezüglich positionsabhängiger Operationen. Offen bleibt hier die Frage nach der inversen Funktion der Einfügeoperation, die Löschoperation scheidet hierzu aus. Die Einführung einer sogenannten Destroy-Funktion könnte helfen, falls diese ausdrücklich als nicht rücknehmbar definiert wird.

6. Implementierung

Der Gruppeneditor „Joint Editor" wurde in CLOS implementiert. Die Benutzungsoberfläche (s. Abb. 7) basiert auf dem Toolkit XIT [8]. Der Gruppeneditor ähnelt von der Funktionalität im wesentlichen einem Einbenutzereditor (als Vorbild diente EMACS in der Version 18 [7]), um den Übergang von einem Ein- zu einem Mehrbenutzereditor zu erleichtern. Textoperationen besitzen einen zusätzlichen Parameter, der den Benutzer angibt, der die Operation angestoßen hat. Der Editor wurde derart erweitert, daß Texte verschiedener Autoren in unterschiedlichen Farben – oder Schriftarten – dargestellt werden. Dies trägt zum gegenseitigen Gewahrsein bei und unterstützt somit die interne Koordination. Pastellartige Farben als Hintergrund zu schwarzer Schrift haben sich als zweckmäßig erwiesen. Farben stellen eine bekannte Metapher dar, um verschiedene Personen zu unterscheiden (vgl. Spielfiguren). Obwohl sie prinzipiell frei wählbar sind, sollte eine feste Zuordnung bevorzugt werden. Bei Bedarf ist die farbliche Markierung auch abschaltbar.

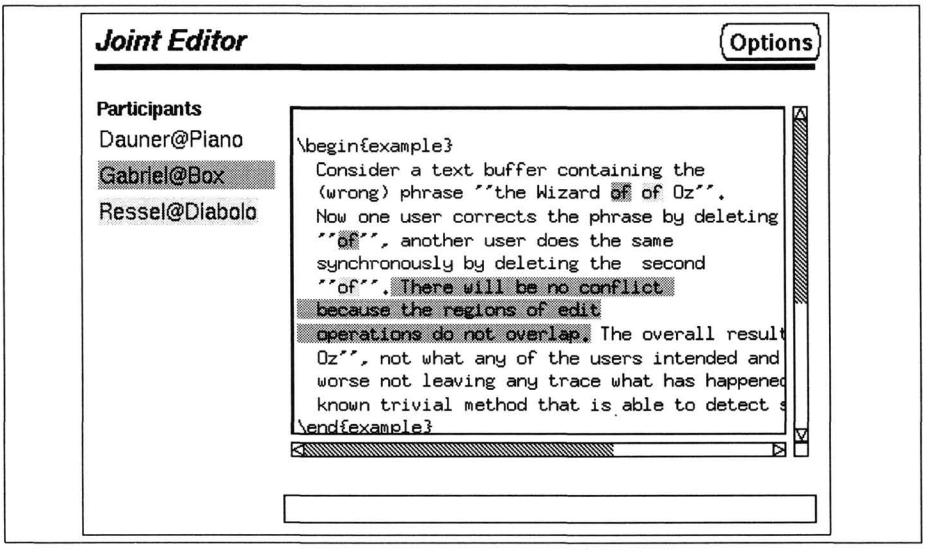

Abb. 7: Oberfläche des Texteditors für Gruppen mit Textbeiträgen von drei Teilnehmern

Die objektorientierte Repräsentation von Operationen erlaubt es, diese in eine Vererbungshierarchie einzubetten, die die Komplexität der Transformationsregeln vereinfacht und die Zahl der Regeln verringert.

Das mehrdimensionale Interaktionsmodell, das während der Verarbeitung aufgebaut wird, kann zusammen mit dem Dokument abgespeichert und wieder geladen werden. Es reicht hierbei aus, die Originaloperationen abzuspeichern, da alle anderen Informationen daraus wieder eindeutig abgeleitet werden können. Eine zusätzliche chronologische, lineare Historie, die natürlich bei jedem Benutzer unterschiedlich sein kann, kann ebenfalls abgespeichert und wieder geladen werden. Sie dient dazu, mit Hilfe einer einfachen Bedienoberfläche, ähnlich der eines Videogeräts, vorhergehende Änderungen und sogar vergangene Sitzungen animationsartig abzurufen.

Ein Menü mit Scroll-Balken stellt die chronologische Historie zusätzlich dar. Einträge in der Historie können ausgewählt werden, um sie selektiv rückgängig zu machen. Hierzu wird lokal eine Undo-Operation erzeugt und zwar im Systemzustand, wie er nach Ausführung der ausgewählten Operation bestand. Mit Hilfe der Transformationsregeln wird diese Operation in den aktuellen Zustand transformiert; das Resultat wird dann wie eine Benutzereingabe behandelt, also lokal

ausgeführt und an die anderen Editoren weitergeleitet. Der Definitionsbereich der Transformationsregeln muß hierbei auf solche Fälle ausgedehnt werden, in denen die Akteure zu transformierender Operationen übereinstimmen. Dies kann dazu führen, daß gewisse Transformationen nicht mehr eindeutige Ergebnisse liefern. Dies beeinträchtigt aber die Ermittlung der Undo-Operation nicht, da die Transformation nur einmal lokal ausgeführt wird und erst das Endergebnis weitergeleitet wird. Es kann somit nicht zu Inkonsistenzen zwischen den verschiedenen Editoren kommen.

7. Vergleich mit anderen Arbeiten

Es gibt eine Reihe von Gruppeneditoren oder allgemeinen Ansätzen, die im folgenden mit unserem Ansatz verglichen werden.

GROVE [5] verwendete als erster Editor Operationstransformationen. Der Originalalgorithmus konnte allerdings für Sitzungen mit mehr als zwei Personen noch keine Konsistenz garantieren. Wir erweiterten das Verfahren außerdem um das Interaktionsmodell, das die komplexen zeitlichen Beziehungen repräsentiert und um eine Undo-Funktion. Im Gegensatz zu unserem Editor bietet GROVE die Möglichkeit, Dokumentteile in verschiedene Bearbeitungsmodi (*private, public, shared*) zu versetzen.

ShrEdit [4] verwendet Sperrmechanismen auf der Ebene von Textselektionen. Zwei Einfügemarken an derselben Textposition sind verboten, das Auftreten dieses Konflikts wird akustisch angezeigt. Allerdings verhindert dies z. B. das parallele Editieren eines leeren Textpuffers. Zudem gelten die Einschränkungen, die für Sperrmechanismen gelten.

Der Gruppeneditor DistEdit [10] benützt ebenfalls Sperrmechanismen, um Konflikte zu vermeiden. Sowohl temporäre, implizite als auch explizit anzufordernde Sperren werden unterstützt. Eine den L-Transformationen verwandte Transformationsart (*Γ-Transformationen*) wird dort zum Vertauschen benachbarter Operationen in einer Historie verwendet, um Gruppen-Undo und selektives Undo im allgemeinen zu implementieren. Zusätzlich werden sogenannte Konfliktfunktionen definiert, die testen, ob zwei benachbarte Operationen in einer Historie miteinander vertauschbar sind. Konflikte mit Operationen, die bereits rückgängig gemacht wurden, werden extra behandelt und können so eliminiert werden. Interessant ist in diesem Zusammenhang, daß sich die hier vorgestellten L-Transformationen und die Γ-Transformationen eindeutig aufeinander abbilden lassen.

Abowd und Dix [1] schlagen als Alternative zu Operationstransformationen die Verwendung dynamischer Zeiger (*dynamic pointers*) vor, die Einfügepositionen repräsentieren und bei Modifikationen automatisch verschoben werden. Dieses Vorgehen setzt voraus, daß bei der Ausführung einer Aktion bereits alle davon abhängigen Aktionen bekannt sind. Das Verfahren hilft auch nicht, Konflikte bei identischen Bezugspunkten zu lösen. Für Benutzer ist unklar, ob und welche Zeiger am Rande eines Bereichs mitberücksichtigt werden sollen, wenn darauf Move- oder Copy-Operationen ausgelöst werden. Außerdem wird eine lineare Historie vorausgesetzt. Unser Editor verwaltet zwar auch eine lineare Historie, um für jeden Benutzer chronologisches Navigieren im Interaktionsraum zu ermöglichen; wir halten aber das für alle Benutzer identische umfassende Interaktionsmodell für ebenso wichtig, um den Benutzern die Orientierung im Interaktionsraum zu ermöglichen.

GINA [2] verwaltet einen History-Baum, dessen Äste verschiedene Versionen darstellen. Diese Äste können mittels der Technik des *selektiven Redo* kombiniert werden. Hierzu werden keine Transformationen benutzt, sondern es wird durch Testfunktionen ermittelt, ob der neue Zustand mit dem alten derart kompatibel ist, daß die Operation unverändert ausgeführt werden kann. Dieser Mischprozeß muß manuell vorgenommen werden. Das Ergebnis ist davon abhängig, welcher Ast zu welchem „dazugemischt" wird. Dadurch, daß GINA bei der Ausführung nur Ausgangs- und Zielzustand betrachtet, ist es nicht möglich, bestimmte Konfliktarten zu erkennen. Ein Beispiel ist das Färben und gleichzeitige Kopieren eines Objektes. Das Färben bezieht sich nur auf das Originalobjekt, obwohl es Sinn macht, das Einfärben auch auf die Objektkopie anzuwenden. Unser Ansatz erlaubt dies. Hierzu wird eine der beiden Alternativen standardmäßig bei der Transformation durchgeführt. Gleichzeitig kann eine Meldung bzw. Abfrage erfolgen, ob die andere Alternative durchgeführt werden soll. Es ist denkbar, die Voreinstellung, welche Alternative standardmäßig ausgewählt wird, interaktiv und kooperativ mit Hilfe desselben Verfahrens im laufenden Betrieb zu ändern.

8. Ergebnisse und Ausblick

Operationstransformationen erfüllen die weiter oben gestellten Anforderungen und erweisen sich somit als ein geeignetes Verfahren zur kooperativen Interaktionsunterstützung in Groupware:

1. Ein „feinkörniger" Informationsaustausch und die Visualisierung fremder Aktionen gewährleisten hohes gegenseitiges Gewahrsein.
2. Sofortige lokale Ausführung einer Eingabe garantiert kurze Antwortzeiten, eine wesentliche Voraussetzung für direkte Manipulation.

3. Die replizierte Architektur erleichtert die Erweiterung bestehender Anwendungssysteme für Einzelbenutzer.
4. Die formale Grundlage erlaubt es, Vorhersagen über Konsistenzverhalten zu machen. Der Computer kann jederzeit einen syntaktisch konsistenten Zustand herstellen. Benutzereingriffe können so weitgehend vermieden werden. Über das Interaktionsmodell haben die Benutzer darüber hinaus jederzeit Zugriff auf vergangene Interaktionen.

Damit sie Benutzererwartungen entsprechen, sollten Transformationen zwei Eigenschaften, die Undo- und die Neutralitätseigenschaft, haben. Für die Implementierung eines einfachen Texteditors konnten Transformationen bestimmt werden, die bis auf eine Ausnahme den genannten Eigenschaften entsprechen. Insbesondere ist es mit Hilfe einer zusammengesetzten Operation möglich, auch komplexere Editorfunktionen als elementare Operationen anzusehen. Durch Ausnutzung spezieller Eigenschaften ist es zudem möglich, mit wesentlich weniger Regeldefinitionen auszukommen, als zunächst theoretisch zu erwarten waren. Wir suchen weitere Anwendungsbereiche für Operationstransformationen. Davon können auch Anwendungen profitieren, die selektives Undo mittels Γ-Transformationen implementieren.

Wegen der Toleranz gegenüber Netzausfällen eignet sich das Verfahren auch für asynchrone Anwendungen, bei denen die Teilnehmer nicht gleichzeitig am Computer anwesend sein müssen. Ein Beispiel wäre verteiltes „Brainstorming": Jeder Beteiligte erweitert lokal eine Brainstorming-Datei; die jeweiligen Änderungen werden per elektronischer Post an alle anderen verteilt, wo sie mittels Operationstransformationen mit den aktuellen Zuständen verrechnet werden.

Das Interaktionsmodell kann schließlich als Grundlage weiterer unterstützender Module dienen. Ein Interaktions-Browser kann beispielsweise das Navigieren durch den Ausführungsraum erlauben und so vergangene Zustände auf einfache Weise zugänglich machen. Darüber hinaus können Planerkennungs- oder Analysekomponenten auf dem Interaktionsmodell aufsetzen. Damit können Interaktionsmuster gesucht und z. B. der Parallelitätsgrad der Zusammenarbeit oder die Häufigkeit von Konfliktfällen analysiert werden.

Literatur

[1] G. D. Abowd, A. J. Dix. Giving Undo Attention. *Interacting with Computers*, 4(3):317–342, 1992.

[2] T. Berlage, A. Genau. A Framework for Shared Applications with a Replicated Architecture. In *Proceedings of the UIST '93*. ACM Press, 1993.

[3] C. Condon. The Computer Won't Let Me: Cooperation, Conflict and the Ownership of Information. In Steve Easterbrook (Hrsg.), *CSCW: Cooperation or conflict?*, Computer Supported Cooperative Work, Kapitel 8, S. 171–185. Springer-Verlag, London, 1993.

[4] P. Dourish, V. Bellotti. Awareness and Coordination in Shared Workspaces. In Jon Turner, Robert Kraut (Hrsg.), *Proceedings of the CSCW '92*, S. 107–114, Toronto, Canada, 31.Okt.–4.Nov 1992. ACM SIGCHI & SIGOIS, ACM Press.

[5] C. A. Ellis, S. J. Gibbs. Concurrency control in groupware systems. In *Proceedings of the ACM SIGMOD '89 Conference on the Management of Data*, S. 399–407, Seattle, Washington, 1989. ACM, New York.

[6] C. A. Ellis, S. J. Gibbs, G. L. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):38–58, Januar 1991.

[7] J. Gosling. *Unix Emacs*. Carnegie-Mellon University, Pittsburgh, 1982.

[8] J. Herczeg, H. Hohl, M. Ressel. Progress in Building User Interface Toolkits: The World According to XIT. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, S. 181–190, November 1992.

[9] U. Piepenburg. Ein Konzept von Kooperation und die technische Unterstützung kooperativer Prozesse in Bürobereichen. In J. Friedrich, K.-H. Rödiger (Hrsg.), *Computergestützte Gruppenarbeit (CSCW)*. B. G. Teubner, 1991.

[10] A. Prakash, M. J. Knister. Undoing Actions in Collaborative Work. In Jon Turner, Robert Kraut (Hrsg.), *Proceedings of the CSCW '92*, Consistency in Collaborative Systems, S. 273–280, Toronto, Canada, 31.Okt.–4.Nov 1992. ACM SIGCHI & SIGOIS, ACM Press.

Danksagung

Prof. Rul Gunzenhäuser danke ich für die Unterstützung und die hilfreichen Kommentare. Jürgen Herczeg und Hubertus Hohl steuerten wichtige Grundlagen der Implementierung bei.

Matthias Ressel
Institut für Informatik
Universität Stuttgart
Breitwiesenstr. 20-22
70565 Stuttgart