Entwurf eines Gruppeneditors: Erfahrungen mit einem optimistischen Ansatz

Matthias Ressel, Andreas Mailänder

1. Einleitung
2. Koordinationsproblem
3. Entwurf eine Gruppeneditors
4. Interaktionsmodellierung
   4.1. Interpretation nebenläufiger Operationen
5. Allgemeine Methode zur Koordination nebenläufiger Operationen
   5.1. Korrektheit
6. Gruppen-Undo
   6.1. Gruppen-Undo durch Transformation
   6.2. Ordnungsproblem bei Gruppen-Undo
7. Mehrdeutigkeiten
8. Gruppeneditor „Joint Emacs"
9. Einsatz von Joint Emacs
10. Erfahrungen beim Einsatz von Joint Emacs
11. Verwandte Forschungsarbeiten
12. Ergebnisse
Literatur

Zusammenfassung

Bei vielen Aufgaben ist es wünschenswert, daß mehrere Personen ein gemeinsames Dokument gleichzeitig editieren. Gewahrsein, Koordination der nebenläufigen Aktionen und Gruppen-Undo sind wichtige Aspekte beim Entwurf von Groupware, die solche kooperative Tätigkeiten unterstützen soll. Eine geeignete Modellierung paralleler Benutzerinteraktion ist hierzu unerläßlich. Wir schlagen einen neuen optimistischen Ansatz vor, der auf einem mehrdimensionalen, gitterartigen Interaktionsmodell und der Transformation nebenläufiger Aktionen beruht. Er eignet sich damit vor allem für räumlich verteilte Zusammenarbeit. Die Umsetzung des Ansatzes in einem prototypischen Gruppentexteditor und die dabei gewonnenen Erfahrungen werden vorgestellt und diskutiert.

1 Einleitung

Bei vielen Aufgaben wie dem gemeinsamen Verfassen einer E-Mail oder eines Konferenzartikels ist es wünschenswert, daß mehrere Personen ein Dokument gleichzeitig editieren können. Beim Entwurf eines Gruppeneditors, der solche Tätigkeiten unterstützt, sind mehrere wichtige Aspekte zu berücksichtigen. *Gewahrsein* (engl. *awareness*) bezeichnet das Wissen, das jeder Teilnehmer von den Aktionen der anderen Teilnehmer und deren Auswirkungen auf

den Anwendungszustand hat (Dourish/Belotti 1992). Gegenseitiges Gewahrsein ist eine notwendige Voraussetzung für jegliche kooperative Arbeit, insbesondere wenn diese über Raum und Zeit verteilt erfolgt. Kooperation ist meist mit *nebenläufigen Aktionen* verbunden, da die beteiligten Personen nicht aufeinander warten wollen bzw. nicht die Zeit dazu haben. Die Wahl der richtigen Methode, um solche nebenläufigen Aktionen miteinander zu *koordinieren*, stellt ein entscheidendes Designproblem dar, dessen Lösung zusätzlich von Faktoren wie dem Grad des Gewahrseins, der Autonomie der Benutzer und dem Grad der Flexibilität abhängt. Ein weiterer wichtiger Aspekt ist ein *Undo-Mechanismus*, eine Funktionalität, die von Benutzern erwartet und berechtigterweise gefordert wird; dies um so mehr bei Gruppeneditoren, als dort die Möglichkeit, unerwünschte Operationen rückgängig zu machen, die Angst mindert, durch Unachtsamkeit Beiträge anderer Autoren zu „beschädigen". Gruppen-Undo ist allerdings ein nichttriviales Problem, für das bisher keine allgemein anwendbare Lösung gefunden wurde (Abowd/Dix 1992, Prakash/Knister 1994).

Ziel dieses Beitrags ist es, zu zeigen, daß es möglich ist, einen Gruppeneditor zu bauen, der es Benutzern erlaubt, nebeneinander lokale Kopien eines Dokument zu editieren, ohne deren Konsistenz zu verletzen, der Gewahrsein unterstützt und der Undo-Funktionalität anbietet.

2 Koordinationsproblem

Das abschließende Editieren eines Konferenzbeitrags stellt ein typische Beispiel für eine kooperative Tätigkeit dar. Sie zeichnet sich u. a. dadurch aus, daß mehrere Autoren i. d. R. an verschiedenen Orten und u. U. zu verschiedenen Zeiten zusammenarbeiten und die Tätigkeit innerhalb einer vorgegebenen Zeit zu erledigen ist. Außerdem erwarten die Autoren eine gewisse Autonomie: Der individuelle Umgang mit dem Dokument sollte möglichst wenig durch die Aktionen der anderen eingeschränkt sein.

Das Koordinationsproblem besteht darin, zu entscheiden, wie die Zugriffe auf das Dokument aufeinander abgestimmt werden. Im Hinblick auf den Einsatz von Groupware zur Unterstützung des kooperativen Editierens lassen sich folgende Unterprobleme formulieren: Wie wird die Aufgabe aufgeteilt und wie werden die Teile den Kooperationspartnern zugewiesen? Wie erfolgt die Benachrichtigung untereinander, wann und welche Änderungen ausgeführt wurden? Wie kann die Aufgabe in kürzerer Zeit erledigt werden? Wie kann die Autonomie der Teilnehmer gewahrt bleiben? Welche Systemarchitektur sollte für das Groupware-System gewählt werden?

Für das Koordinationsproblem – gleichgültig, ob mit oder ohne Rechnerunterstützung – existieren drei grundsätzliche Lösungsansätze: Sequentialisierung, Partitionierung und Kopieren.

Bei auf *Sequentialisierung* basierenden Methoden erhält reihum jeder Autor das Dokument zur Bearbeitung. Änderungen können sich somit auf Korrekturen, Modifikationen und Kommentare vorhergehender Bearbeitungen beziehen. Üblicherweise erfolgt dieser Vorgang in mehreren Zyklen, so daß jeder Autor das Dokument mehrmals erhält. Der Hauptnachteil von Sequentialisierung besteht darin, daß der Vorgang keinerlei Parallelität ausnutzt und damit sehr viel Zeit erfordert.

Bei auf *Partitionierung* beruhenden Methoden wird jedem Autor ein bestimmter Teil des Dokuments zugewiesen. Im Vergleich zur Sequentialisierung ist paralleles Arbeiten möglich. Der Hauptnachteil von Partitionierung besteht darin, daß ein Autor nur die ihm zugewiesenen Abschnitte des Dokumentes bearbeiten darf, es aber z. B. nicht möglich ist, Schreibfehler in anderen Abschnitten zu korrigieren.

Bei auf *Kopieren* beruhenden Methoden erhält jeder Autor eine individuelle Kopie des Dokuments und kann somit ohne Einschränkungen das ganze Dokument editieren. Der Hauptnachteil von Kopien besteht darin, daß durch fehlendes Gewahrsein widersprüchliche Änderungen vorgenommen werden und Korrekturen redundant erfolgen. Außerdem entsteht die Notwendigkeit, die entstandenen Kopien in eine gemeinsame Version überzuführen.

3 Entwurf eine Gruppeneditors

Wie bei jeder Groupware muß auch beim Entwurf eines Gruppeneditors zum kooperativen Schreiben entschieden werden, was für ein Lösungsansatz für das Koordinationsproblem gewählt wird (Greenberg/Marwood 1994).

Aus der Sicht eines Implementierers (Programmierers) besteht die einfachste Lösung darin, eine *zentralisierte* Architektur zu wählen, in der die Anwendungsdaten von einem zentralen Prozeß verwaltet werden, der mit den einzelnen Teilnehmerprozessen kommuniziert. Sequentialisierung kann hierbei sehr einfach, z. B. über die exklusive Vergabe von Berechtigungs-Tokens (Floor Control) durch den zentralen Koordinationsprozeß, erzielt werden. Partitionierung läßt sich sehr einfach durch dynamisches Sperren bestimmter Dokumentabschnitte erreichen. Inkonsistente Anwendungszustände kann es daher nicht geben. Die oben geschilderten Nachteile von Sequentialisierung und Partitionierung werden durch den Einsatz vernetzter Computer – etwa durch die beschleunigte Kommunikation – abgemildert. Die zentralisierte Architektur erfordert aber weiterhin, daß jede Benutzeroperation zum zentralen Prozeß übermittelt werden muß, bevor weitere Operationen folgen dürfen. Wenn die Laufzeiten hoch sind, kann dies zu einer erheblichen Einbuße beim Antwortverhalten führen: Die Benutzer werden gezwungen, zwischen aufeinanderfolgenden Operationen zu warten. In graphischen Editoren ist dieses Problem weniger gravierend, da Operationen in aller Regel in

ausreichendem Abstand voneinander ausgeführt werden (weniger als eine Operation pro Sekunde). Wenn die Eingabe hierbei wegen Laufzeitverzögerungen z. B. für mehrere Zehntelsekunden gesperrt werden muß, bekommt dies ein Benutzer kaum mit. In Texteditoren jedoch, in denen Eingabeaktionen von sechs und mehr Tastendrücken in der Sekunde üblich sind, sind schon kleinste Verzögerungen zwischen Eingabe und Anzeige auf dem Bildschirm unannehmbar.

Aus der Sicht eines Benutzers erscheint eine *replizierte* Architektur erfolgversprechender, bei der alle wesentlichen Anwendungsdaten bei jedem Teilnehmer in einer Kopie vorhanden sind und lokal bearbeitet werden. Der Zugriff erfolgt damit einfach und schnell. Die sich notwendigerweise ergebenden nebenläufigen Benutzeroperationen werden durch ein sogenanntes *optimistisches* Koordinationsverfahren kombiniert. Bei einem solchen optimistischen Ansatz werden Benutzeroperationen lokal unmittelbar ausgeführt; der individuelle Arbeitsfluß wird nicht vom Übertragungsverhalten des Kommunikationsnetzes beeinflußt. Operationen anderer Benutzer werden so bald wie möglich ausgeführt, und zwar derart, daß sich trotz der unterschiedlichen Ausführungsreihenfolgen bei allen Benutzern ein identischer Endzustand des Dokuments einstellt. Auch Gewahrsein wird auf diese Weise ausreichend unterstützt.

Da die benutzerorientierten, software-ergonomischen Kriterien für uns im Vordergrund stehen, haben wir uns dafür entschieden, zum Entwurf unseres Gruppeneditors eine replizierte Architektur zu verwenden und hierfür ein geeignetes optimistisches Verfahren für paralleles Texteditieren zu entwickeln.

4 Interaktionsmodellierung

Sowohl für die Koordination nebenläufiger Aktionen und für Gruppen-Undo als auch zur nachträglichen Information über vergangene Aktionen ist es notwendig, ein geeignetes Modell der Interaktion aufzubauen. Im Kontext des kooperativen Editieren muß dieses Modell berücksichtigen, daß zwar jeder Benutzer eigene Operationen sequentiell erzeugt, daß aber die Operationen verschiedener Benutzer nebenläufig erfolgen können.

Bei Einbenutzerprogrammen sind sequentielle Interaktionen typisch, die als lineare Historie repräsentiert werden können. Auch bei Groupware könnten nebenläufige Benutzeraktionen prinzipiell – für alle Teilnehmer identisch – sequentiell geordnet werden. Die hierzu erforderlichen Verfahren für verteilten Architekturen sind aber zeitaufwendig und ungeeignet für Benutzungsschnittstellen von Editoren, die unmittelbares Feedback erfordern (Ellis/Gibbs 1989).

Wir haben daher ein neuartiges Interaktionsmodell entwickelt, das strukturell in ein mehrdimensionales Gitternetz eingebettet ist. Darin entspricht jeder Gitterpunkt einem Anwendungszustand und jeder mit einem Label markierte Pfeil einer Benutzeraktion. Genauer gelten folgende Beziehungen:

Pfeil	**Benutzeraktion**
Richtung	Teilnehmer
Ausgangspunkt	Anwendungszustand vor Ausführung
Zielpunkt	Anwendungszustand nach Ausführung
Label	Auszuführende Operation
Gitterpunktkoordinaten	Anzahl ausgeführter Operationen

Beispiel 1: Abb. 1 zeigt ein Interaktionsmodell für zwei Benutzer, A und B. Aktionen von Benutzer A sind durch nach rechts weisende Pfeile dargestellt, die hier mit den Operationen o_{11} bis o_{15} markiert sind; analog Aktionen von Benutzer B durch nach oben weisende Pfeile, hier markiert mit den Operationen o_{21} bis o_{23}. Der Pfeil für die dritte Eingabe von Benutzer B, die Operation o_{23}, startet z.B. im Gitterpunkt (1,2), da die Eingabe in einem Anwendungszustand erfolgte, in dem *eine* Operation von Benutzer A und *zwei* von Benutzer B ausgeführt waren. Das Paar o_{11} und o_{21} oder das Paar o_{13} und o_{23} stellen Beispiele für nebenläufige Operationen dar.

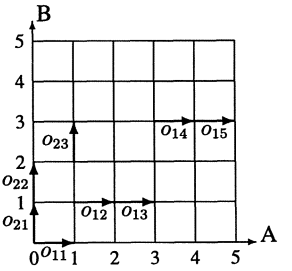

Abbildung 1: Interaktionsmodell für nebenläufige Operationen

4.1 Interpretation nebenläufiger Operationen

Da üblicherweise nur die Anwendung einzelner Operationen auf einen Anwendungszustand definiert ist, müssen nebenläufige Operation zunächst sequentialisiert werden, bevor sie bei einem Benutzer ausgeführt werden können. Dies hat zur Folge, daß höchstens ein Element einer Menge von nebenläufigen Operationen in dem Zustand ausgeführt werden kann, in dem die Eingabe stattgefunden hat, die restlichen Elemente müssen in einem anderen Zustand

ausgeführt werden. Falls die nebenläufigen Operationen aber ohne jegliche Modifikation sequentialisiert werden, treten unerwartete Resultate auf.

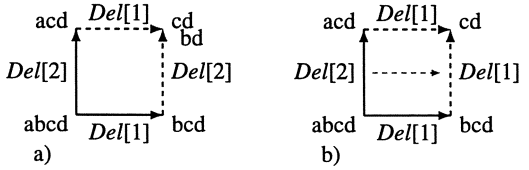

Abbildung 2: a) Falsche und b) korrigierte Sequentialisierung (Beispiel 2)

Beispiel 2: In Abb. 2a stellen die durchgängig gezeichneten Pfeile zwei nebenläufige Benutzereingaben dar: Benutzer A löscht das erste Zeichen (*Del*[1]), Benutzer B im selben Zustand das zweite Zeichen (*Del*[2]). Nur eine der beiden Operation kann im gegebenen Zustand – einem Textpuffer mit dem Inhalt *abcd* – ausgeführt werden. Wird zunächst *Del*[1] von Benutzer A ausgeführt, muß *Del*[2] von Benutzer B zwangsläufig im Zustand *bcd* ausgeführt werden; dies würde jedoch einen völlig unerwarteten Endzustand von *bd* ergeben (keiner der beiden Benutzer hatte die Absicht, den Buchstaben *c* zu löschen). Umgekehrt führt die Ausführung von *Del*[1] nach *Del*[2] – mit Zwischenzustand *acd* – zum erwarteten Ergebnis *cd*.

Glücklicherweise gibt es eine einfache Lösung dieses Problems: Falls die Operation *Del*[2] von Benutzer A als zweite ausgeführt werden sollte, ist zu berücksichtigen, daß zuvor ein Zeichen gelöscht wurde und hierdurch alle rechts davon gelegenen Positionen um eine Position nach links verschoben wurden. Deshalb sollte eigentlich die modifizierte Operation *Del*[1] ausgeführt werden. Auf diese Weise wird erreicht, daß beide Ausführungsreihenfolgen zum selben erwarteten Ergebnis führen (vgl. Abb. 2b; die notwendige Modifikation von *Del*[2] ist durch einen dünnen Pfeil im Inneren des Transformationsquadrates angedeutet).

Für die grundlegenden Operationen eines Texteditors, dem Löschen und Einfügen von Zeichen oder Zeichenketten, lassen sich die erforderlichen Transformationen leicht angeben. Die allgemeine Regel für Einzelzeichen lautet folgendermaßen: Die Lösch- bzw. Einfügeposition einer Operation ist um eins nach rechts zu verschieben, falls zuvor links davon ein Zeichen eingefügt worden ist; sie muß um eins nach links verschoben werden, falls zuvor links davon ein Zeichen gelöscht worden ist. Allgemeiner lassen sich solche Transformationsregeln auch für komplexere Operationen auf Zeichenketten definieren (Ressel 1995).

Häufig führen beliebige Ausführungsreihenfolgen bestimmter Operationen auch ohne Modifikation zum selben Ergebnis, wir sprechen dann von unabhängigen Operationen. Diese operieren typischerweise auf verschiedenen voneinander unabhängigen Objekten, oder sie modifizieren verschiedene voneinander unabhängige Aspekte eines Objekts.

In manchen Fällen können verschiedene Transformationen sinnvoll sein; eine eindeutige Interpretation nebenläufiger Operationen gibt es dann im Grunde nicht. Da ein optimistischer Algorithmus die Transformationen bei jedem Teilnehmer ohne Kommunikation mit den anderen Teilnehmern ausführt, muß gewährleistet sein, daß jeder Teilnehmerprozeß dieselbe Interpretation wählt.

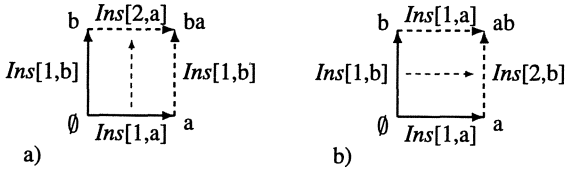

Abbildung 3: Mehrdeutigkeit bei Einfügeoperationen an derselben Position (Beispiel 3)

Beispiel 3:Von den beiden nebenläufigen Operationen *Ins*[1,*a*] und *Ins*[1,*b*], kann entweder die eine oder die andere nach rechts verschoben werden, falls sie als zweite ausgeführt wird. Die eine Alternative führt zum Endzustand *ba*, die andere zu *ab* (s. Abb. 3a–b, ∅ bezeichnet einen leeren Textpuffer). Durch die Vergabe einfacher, numerischer Benutzerprioritäten kann nun bestimmt werden, daß z. B. bei niedrigerer Priorität von Benutzer A Version a) gewählt – hier also seine Operation modifiziert – und bei höherer Priorität Version b) gewählt wird.

5 Allgemeine Methode zur Koordination nebenläufiger Operationen

Die bisherigen Beispiele berücksichtigten nur einfache Paare nebenläufiger Operationen. Die Interpretation komplexerer Mengen nebenläufiger Operation erfolgt mit Hilfe des Interaktionsmodells wie folgt. Jede Benutzeraktion wird zunächst am Gitterpunkt, der dem Anwendungszustand entspricht, eingefügt (vgl. Abb. 1). Falls darüber hinaus bereits weitere Operationen ausgeführt worden sind, muß die Benutzeraktion in den aktuellen Zustand transformiert werden. Abhängig von der Anzahl bereits ausgeführter nebenläufiger Operationen, kann dies ebenso viele Transformationsschritte erfordern. Eventuell für die Transformation benötigte, im Interaktionsmodell aber noch fehlende Operationen, können durch rekursive Anwendung weiterer Transformationsschritte berechnet werden. Durch diesen Transformationsprozeß wird

schließlich ein *erweitertes Interaktionsmodell* aufgebaut. Jedem Weg durch dieses gitterartige Modell entspricht eine gültige Ausführungsreihenfolge von Benutzeraktionen.

Beispiel 4: Abb. 4a zeigt das erweiterte Interaktionsmodell, nachdem alle möglichen Transformationen auf die Menge der in Abb. 1 dargestellten Benutzeraktionen ausgeführt wurden (Labels wurden hier der Übersichtlichkeit wegen weggelassen). Alle Pfeile mit Ausnahme der jeweils untersten jeder Spalte und der jeweils am weitesten links stehenden jeder Zeile sind das Ergebnis eines Transformationsschrittes. Abb. 4b zeigt eine mögliche Ausführungsfolge. Sie beginnt mit der Ausführung einer untransformierten Originaloperation von Benutzer A, gefolgt von zwei transformierten Operationen von Benutzer B usw.

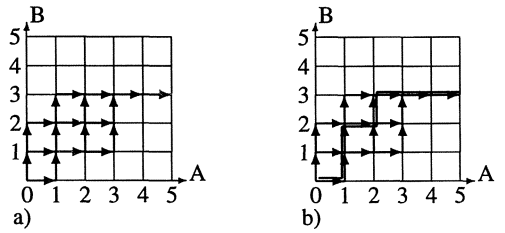

Abbildung 4: a) Erweitertes Interaktionsmodell, b) Beispiel einer gültigen Ausführungsfolge

Bei drei Teilnehmern nimmt das resultierende Gittermodell eine räumliche Struktur, bei n Teilnehmern eine n-dimensionale Struktur an. Das erweiterte Interaktionsmodell von zwei einzelnen, nebenläufigen Operationen hat – wie gesehen – die Form eines Quadrates, jenes von drei die Form eines Würfels usw.

5.1 Korrektheit

Es stellt sich die Frage, ob verschiedene Pfade durch das erweiterte Interaktionsmodell und damit verschiedenen Ausführungsreihenfolgen auf denselben Anwendungszustand führen. Falls dies nicht der Fall wäre, würden die Teilnehmer bei dem vorgestellten optimistischen Koordinationsverfahren nach einiger Zeit divergierende Anwendungszustände haben.

Tatsächlich konnten wir beweisen, daß es für die Korrektheit im Falle von zwei Teilnehmern ausreicht, daß die Transformation jedes Paares von Benutzeraktionen eindeutig definiert ist und daß die Ausführung der beiden resultierenden Operationssequenzen denselben Effekt auf den Anwendungszustand ausüben (Ressel/Nitsche-Ruhland/Gunzenhäuser 1996). Im Falle

von mehr als zwei Teilnehmern konnten wir eine weitere Bedingung identifizieren, mit deren Hilfe die Korrektheit auch hier bewiesen werden kann. Diese zusätzliche Bedingung stellt sicher, daß alle im Interaktionsmodell enthaltenden – eventuell auf verschiedenen Wegen zu berechnenden – Labels eindeutig sind.

6 Gruppen-Undo

Ein mit der Koordination nebenläufiger Operationen verwandtes Problem ist Gruppen-Undo. Ein oder mehrere Operationen zu stornieren, ist eine wichtige Benutzerintention, die jeder Editor unterstützten sollte (Abowd/Dix 1992). In Groupware werden zwei Arten von Undo unterschieden: lokales Gruppen-Undo und globales Gruppen-Undo. Globales Gruppen-Undo kehrt den Effekt der von der Anwendung zuletzt ausgeführten Operation um, unabhängig davon, wer diese veranlaßt hatte. Lokales Gruppen-Undo hat dagegen nur Auswirkungen auf eigene Operationen, i. d. R. also die letzte eigene Eingabe. Eine allgemeine Methode, um Operationen zu stornieren, nach denen bereits weitere Operationen ausgeführt wurden, wird selektives Undo genannt. Lokales Gruppen-Undo ist somit ein Spezialfall von selektivem Undo, da andere Benutzer nach der letzten eigenen Eingabe bereits weitere Operationen ausgeführt haben können. Im Gegensatz zum allgemeineren selektiven Undo bezieht sich lokales Gruppen-Undo aber immer nur auf die letzte noch nicht stornierte eigene Eingabe oder Operation. Bei Gruppeneditoren, bei denen alle Teilnehmer unabhängig voneinander ihren eigene Einfügeposition besitzen, sollte lokales Gruppen-Undo angeboten werden.

Die meisten bekannten Undo-Verfahren setzen voraus, daß sich die Applikation bei der Ausführung des Undo im selben Zustand befindet, in den die zu stornierende Operation führte. Andernfalls wäre es z.B. nicht möglich, die inverse Operation auf den Anwendungszustand anzuwenden, um die Effekte rückgängig zu machen.

Beispiel 5: Eine Operation *Ins*[2,*a*] kann mittels einer Operation *Del*[2] rückgängig gemacht werden. Angenommen jedoch, zwischenzeitlich wäre eine Operation *Ins*[2,*H*] ausgeführt worden, dann würde *Del*[2] ohne Transformation das falsche Zeichen löschen, nämlich den Buchstaben *H* anstatt *a* wie gewünscht. Um das richtige Zeichen zu löschen, muß *Del*[2] unter Berücksichtigung der Einfügeoperation in *Del*[3] transformiert werden.

6.1 Gruppen-Undo durch Transformation

Wie Beispiel 5 zeigt, läßt sich das Problem des lokalen Gruppen-Undo auf Transformationen zurückführen. Um eine Benutzereingabe rückgängig zu machen, wird im geeigneten Zustand

die inverse Operation erzeugt und mit Hilfe des Interaktionsmodells in den momentanen Anwendungszustand transformiert und dort zur Ausführung gebracht.

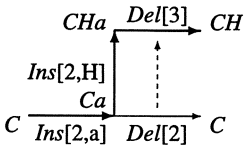

Abbildung 5: Lokales Gruppen-Undo mittels Transformation

Beispiel 6: Abb. 5 zeigt das zu Beispiel 5 gehörige Interaktionsmodell. Die inverse Operation *Del*[2] wird im unmittelbar auf die zu stornierende Operation *Ins*[2,*a*] folgenden Anwendungszustand eingefügt. Eine Transformation dieser Löschoperation in den aktuellen Zustand – Textpufferinhalt *CHa* nach zwischenzeitlicher Ausführung von *Ins*[2,*H*] – ergibt die korrekte Operation *Del*[3].

6.2 Ordnungsproblem bei Gruppen-Undo

Beim Stornieren mehrere Löschoperationen ist – ohne besondere Maßnahmen – häufig ein unangenehmer Effekt anzutreffen, der durch folgendes Beispiel illustriert wird.

Beispiel 7: Ausgehend von einem Textpuffer mit Inhalt *ab* löscht Benutzer A das Zeichen *a* und Benutzer B daraufhin das Zeichen *b*. In diesem Zustand – mit leerem Textpuffer – stornieren beide jeweils ihre eigene Löschoperation. Wie in Abb. 6 dargestellt, ergibt sich hierdurch ein Interaktionsmodell, in dem zwei Undelete-Operationen mit identischer Einfügeposition gegeneinander transformiert werden müssen. Prinzipiell können Undelete-Operationen (also das Stornieren von Löschoperationen) wie Einfügeoperationen behandelt werden. In diesem Fall würden also die Prioritäten der beiden Benutzer hinzugezogen. Angenommen Benutzer A hat die niedrigere Priorität, dann würde der Positionsparameter seiner Operation um eins erhöht werden. Die resultierende Operation *UnDel*[2,a] ergäbe dann den Textpufferinhalt *ba* – im Widerspruch zum Ausgangsinhalt und im Gegensatz zu den Benutzererwartungen.

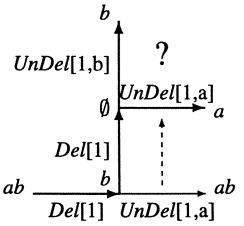

Abbildung 6: Ordnungsproblem bei Gruppen-Undo

Eine Änderung der Benutzerprioritäten nützt nichts, da sich dann eine andere Sequenz von Löschoperationen gefolgt von entsprechenden Undelete-Operationen finden läßt, die zu einem gleichermaßen unerwarteten Resultat führt. Unser Lösungsansatz besteht darin, nach einer inhärenten räumlichen Ordnungsbeziehung der beteiligten Undelete-Operationen zu suchen. Tatsächlich kann die hierzu nötige Information aus dem bereits vorhandenen Interaktionsmodell herausgelesen werden (hier z. B.: *UnDel*[1,a] links von *UnDel*[1,b]).

7 Mehrdeutigkeiten

Bereits oben haben wir bemerkt, daß es für Paare von nebenläufigen Benutzeraktionen durchaus verschiedene sinnvolle Interpretationen und damit Transformationen geben kann. Unser Ansatz geht auf diese Mehrdeutigkeiten dadurch explizit ein, daß er es ermöglicht, mehrere auf bestimmte Paare von Benutzeraktionen anwendbare Transformationsregeln zu definieren. In solchen Fällen wird normalerweise eine vordefinierte Standardregel zur Transformation ausgewählt. Es ist dann nicht erforderlich, daß das Computersystem die Bearbeitung unterbricht, um zwischen Benutzern oder zwischen Benutzer und Computer einen Dialog anzustoßen, welche Transformationsregel auszuwählen ist. Die betroffenen Benutzer werden allerdings informiert und es wird ihnen ermöglicht, nachträglich eine andere Alternative auszuwählen. In letzterem Fall werden die unerwünschten Operationen storniert und die gewünschten Alternativoperationen nachträglich ausgeführt.

8 Gruppeneditor „Joint Emacs"

Um unseren Ansatz zur Koordination nebenläufiger Benutzeraktionen und für Gruppen-Undo zu testen, haben wir einen prototypischen Gruppentexteditor *Joint Emacs* (s. Abb. 7) entworfen und implementiert. Er basiert auf dem im UNIX-Bereich weit verbreiteten Texteditor Emacs (Stallman 1981) und erleichtert den damit erfahrenen Benutzern den Übergang von einem Einbenutzer- auf einen Mehrbenutzereditor.

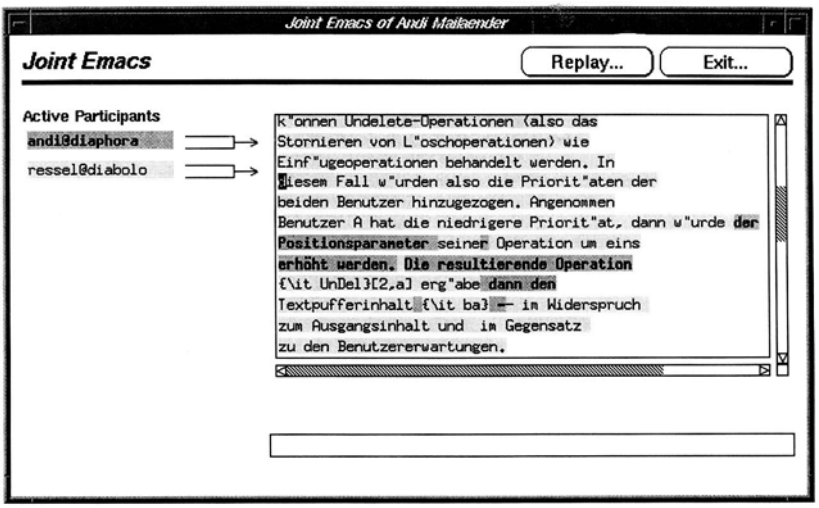

Abbildung 7: Benutzungsoberfläche des Gruppeneditors Joint Emacs

Über die Funktionalität von Emacs hinaus speichert *Joint Emacs* zu jedem Text den zugehörigen Autor. Zur Unterstützung des Gewahrseins wird diese Information durch farbliche Kennzeichnung oder – bei Monochrom-Bildschirmen – durch unterschiedliche Schriftarten visualisiert.

Joint Emacs stellt einen Historie-Mechanismus ähnlich dem GINA Interaction Recorder (Berlage/Spenke 1992) zur Verfügung, mit dem es möglich ist, vergangene Zustände wiederherzustellen bzw den Fortgang des Editierprozesses nachträglich dynamisch anzuzeigen. Der Inhalt des Textpuffers und das aktuelle Interaktionsmodell können darüber hinaus jederzeit zusammen abgespeichert und später erneut geladen werden. Eine Inspektion vergangener Zustände ist also auch sitzungsübergreifend möglich.

Zu Analysezwecken kann die Gitterstruktur des Interaktionsmodells visualisiert werden, um etwa nach Interaktionsmustern zu suchen. Abb. 8 zeigt das Interaktionsgitter einer Sitzung zweier Teilnehmer, in der das Textsystem WORD diskutiert wurde. In einer kontroversen Phase der Diskussion kam es zu etlichen überlappenden Benutzereingaben, was an dem Schachbrettmuster erkennbar ist, das durch die notwendig gewordenen Transformationsschritte gebildet wird (vgl. den hervorgehobenen Bereich in Abb. 8).

Abbildung 8: Struktur eines Interaktionsmodells mit einer typischen Dialogstruktur und Regionen überlappender Kommunikation

9 Einsatz von Joint Emacs

Typische Sitzungen mit Joint Emacs bestehen aus zwei bis drei Teilnehmern. Sitzungen mit bis zu sechs realen Teilnehmern liefen bereits erfolgreich. Die implementierungsbedingte Obergrenze beträgt zur Zeit neun Teilnehmer.

Joint Emacs ist experimentell insofern, als er vor allem zeigen soll, daß unser Ansatz zum einen technisch realisierbar ist und zum anderen kooperatives Editieren prinzipiell unterstützen kann. So fehlt etliches an Funktionalität – Verwaltung mehrerer Puffer, Fenster usw. –, was für ein ausgereiftes Groupware-Produkt notwendig wäre.

Trotz des experimentellen Charakters wird *Joint Emacs* inzwischen tagtäglich als Kommunikationsmedium – in Ergänzung zu persönlicher Kommunikation, E-Mail und Telefon – zwischen räumlich getrennten Mitarbeitern genutzt. Im Vergleich zu persönlicher Kommunikation und Telefon bietet Joint Emacs den Vorteil, auch asynchrone, permanente Kommunikation zu unterstützen: Der Empfänger einer Information muß nicht anwesend sein und kann den Inhalt seines Editors auch zu einem späteren Zeitpunkt lesen. Im Vergleich zu E-Mail oder einem Anrufbeantworter hat Joint Emacs den Vorteil, daß eine Mitteilung auch nachträglich noch modifiziert werden kann. So kann eine Information, falls sie für den Empfänger nicht mehr relevant ist, gelöscht werden. Darüber hinaus bietet die Möglichkeit, uneingeschränkt eigene und fremde Beiträge zu editieren, neuartige Interaktionsmöglichkeiten. Es ist z. B. möglich, daß ein Teilnehmer Text eintippt, ein zweiter diesen unmittelbar umbricht und

ein dritter parallel dazu die Schreibfehler korrigiert. Bei vergleichbaren dezidierten Kommunikationsanwendungen wie *talk* oder *irc* (Internet Relay Chat) ist so etwas nicht möglich.

Eine interessante, neuartige Anwendung von Joint Emacs besteht darin, kooperativ E-Mail-Nachrichten zu schreiben und als Gruppe zu versenden. Letzteres wird dadurch erreicht, daß eine solche Gruppen-E-Mail mehrere Absenderadressen enthält, was u. a. zur Folge hat, daß eine Antwort auf eine solche E-Mail an alle Absender geschickt wird. Letzteres sollte nicht mit einer sogenannten Gruppen-Antwort (*group reply*) verwechselt werden, bei der die Antwort an einen einzelnen Absender und alle ursprünglichen Adressaten verschickt wird. Bei der Erstellung der E-Mail können, wie beim Einsatz als Kommunikationsmedium beschrieben, vielfältige, vor allem parallele Interaktionsmöglichkeiten genutzt werden. Es ist nicht nötig, eine E-Mail zur Fertigstellung zeitaufwendig mehrmals im Kreis zu schicken. E-Mails, auf deren Inhalt mehrere Leute Einfluß nehmen wollen, können auf diese Weise sehr rasch und elegant erstellt und versendet werden. Da es sich beim Versenden der Gruppen-E-Mail um eine nicht stornierbare Operation handelt, ist die Entscheidung, diese Operation auszulösen, zwischen den Teilnehmern abzusprechen. Dies kann entweder durch Abstimmung über einen Audio-Kanal erfolgen oder über ein integrierten Votierungsmechanismus ausgehandelt werden (Herrmann 1994).

10 Erfahrungen beim Einsatz von Joint Emacs

Der Gruppeneditor Joint Emacs zeigte in der Anfangsphase Akzeptanzprobleme, da er im Vergleich zu herkömmlichen Einbenutzereditoren relativ langsam im Feedback und Bildschirmneuaufbau war, was jedoch nicht auf den verwendeten Koordinationsalgorithmus, sondern auf eine ineffiziente, prototypische Implementierung der Editorfunktionen zurückzuführen war. Die hierdurch bei den Benutzern hervorgerufenen Vorurteile gegen einen Gruppeneditor konnten durch eine effizientere Implementierung ausgeräumt werden.

Besonders beim Einsatz als Kommunikationsmedium vermißten die Benutzer eine Benachrichtigung, wenn Kommunikationspartner etwas Neues eingefügt hatten. Aus diesem Grund wurde ein animiertes Piktogramm eingeführt, das sich in einem solchen Fall in einen geöffneten Mund verwandelt. Optional ertönt dazu über Lautsprecher ein durch die Benutzer spezifizierbares Signal.

Einige Benutzer von Joint Emacs hatten wenig Erfahrung mit der Bedienung von Emacs. Die replizierte Architektur erleichterte es, ihre lokale Benutzungsschnittstelle entsprechend ihren Wünschen anzupassen.

In frühen Versionen des Editors traten beim gemeinsamen Editieren einige unvorhergesehene Probleme auf. Wenn die Cursor zweier Teilnehmer sich an identischen Positionen befanden und einer dieser Teilnehmer Text eintippte, wurden beide Cursor entsprechend nach rechts gerückt, befanden sich also weiterhin an identischen Positionen; gaben beide gleichzeitig Text ein, wurden diese Texte miteinander vermischt. Der Koordinationsalgorithmus arbeitete korrekt insofern, als alle Kopien des Editors dasselbe Resultat anzeigten. Wir erkannten, daß in einem solchen Fall nicht alle Cursor mitbewegt werden dürfen. Vielmehr ist hierzu die Priorität der beteiligten Benutzer zu berücksichtigen, wobei eine niedrigere Priorität z. B. bedeutet, daß der Cursor dieses Benutzers mitverschoben werden soll.

Hierdurch trat allerdings ein anderes störendes Problem auf: Manchen Benutzern war es nicht mehr möglich, ganz am Ende des Textpuffers etwas einzugeben, falls dort bereits ein anderer Teilnehmer Text einfügte – eine sehr häufige Situation beim Einsatz von Joint Emacs als Kommunikationsmedium. Tatsächlich war es nach obigem Verfahren einem Teilnehmer mit höherer Priorität nicht möglich, seinen Cursor an dem Cursor des Teilnehmers mit niedrigerer Priorität „vorbeizuschieben". Deshalb wurden *dynamische* Benutzerprioritäten eingeführt, die explizit durch Benutzeraktionen geändert werden können. Springen mit dem Cursor ans Ende des Textes erniedrigt die Priorität z. B. derart, daß der Cursor bei weiteren Einfügeoperationen – bis zur nächsten Prioritätsänderung – am Ende bleibt.

Unsere Erfahrungen zeigen, daß die geringen Einschränkungen, die unser Gruppeneditor den Benutzern beim kooperativen Editieren auferlegt, zu einer größeren Flexibilität der sozialen Interaktion – auch über zeitliche und räumliche Distanzen hinweg – führt. Immer wieder entdecken Benutzer neue produktive Möglichkeiten des Zusammenarbeitens, wie etwa oben beim Einsatz des Editors als Kommunikationsmedium und als Gruppen-E-Mail-Editor beschrieben.

11 Verwandte Forschungsarbeiten

Operationstransformationen wurden zum ersten Mal im dOPT-Algorithmus (distributed OPeration Transformations) zur Koordination nebenläufiger Operationen in verteilten Umgebungen verwendet (Ellis/Gibbs 1989). Dieser Algorithmus produziert allerdings nur dann konsistente Kopien, wenn ein Teilnehmer höchstens jeweils eine Eingabe parallel zu Eingaben anderer Teilnehmer macht. Für allgemeinere Fälle konnte die Korrektheit nicht nachgewiesen werden. Da wir im Gegensatz zum einfachen linearen Interaktionsmodells des dOPT-Algorithmus ein mehrdimensionales Interaktionsgitter verwenden, konnten wir die Korrektheit unseres Ansatzes nachweisen. Besonders für Texteditieren ist dies von großer

Bedeutung, da dort selbst kleinste Inkonsistenzen zwischen verschiedenen Textpuffern nicht toleriert werden können.

In GINA (Berlage/Genau 1993) werden nebenläufige Operationen als Historiebaum modelliert. Das Kombinieren nebenläufiger Operationen erfolgt durch Verschieben der Äste. Um Inkonsistenzen zu vermeiden, muß dies bei jedem Teilnehmer in gleicher, eindeutiger Weise erfolgen. Hierzu werden u. U. zusätzliche Undo-Operationen notwendig, um einen bereits erfolgten Umbau eines Baumes rückgängig zu machen. Dieser Ansatz setzt zudem voraus, daß Operationen nicht modifiziert werden müssen, bevor sie in einem anderen Anwendungszustand ausgeführt werden. Er ist daher für Texteditoren, wo Einfüge- und Löschoperationen angepaßt werden müssen, weniger geeignet.

Eine starke Verwandtschaft zu unserem Ansatz weist der Transformationsansatz von Prakash/ Knister (1994) zur Implementierung von Gruppen-Undo auf. Das von ihnen hierzu eingesetzte Transponieren von Operationen in der linearen Historie kann als äquivalent dazu angesehen werden, in unserem erweiterten Interaktionsmodell eine alternative Ausführungsreihenfolge auszuwählen. Nebenläufige Operationen können bei ihnen aber nicht modelliert werden. Ihr Verfahren setzt einen sequentiell geordneten Eingabestrom voraus, der durch einen zentral verwalteten Sperrmechanismus erzielt wird. Prakash und Knister beschreiben auch einen weiteren Lösungsansatz für das Ordnungsproblem bei Gruppen-Undo: Dabei werden Zeiger auf Einfüge- und Löschpositionen verwaltet, die bei *jeder* Einfüge- oder Löschoperation entsprechend mitverschoben werden. Unser Ansatz hat den Vorteil, daß solche Positionsanpassungen nur durchgeführt werden müssen, wenn tatsächlich eine Undo-Operation ausgeführt wird.

12 Ergebnisse

Wir haben einen neuartigen transformationsorientierten Lösungsansatz zur optimistischen Koordination von nebenläufigen Benutzeraktionen in Groupware entwickelt, der auf einem mehrdimensionalen, gitterartigen Interaktionsmodell basiert. Diese Methode wurde beim Bau des prototypischen Gruppentexteditors „Joint Emacs“ verwendet. Durch ausgiebige Tests mit Benutzern wurde der Editor ständig verbessert und wird inzwischen u. a. als innovatives Kommunikationsmedium und zum gemeinsamen Verfassen und Versenden von E-Mail in der täglichen Arbeit erfolgreich eingesetzt. Es zeigte sich, daß es möglich ist, einen funktionsfähigen Gruppeneditor zu entwickeln, der es räumlich verteilten Benutzern erlaubt, gemeinsam ein Textdokument zu bearbeiten, der die Benutzer möglichst wenig einschränkt oder behindert, von Netzverzögerungen unabhängige, kurze Antwortzeiten garantiert, Gewahrsein –

auch über Sitzungsgrenzen hinweg – unterstützt, konsistente, erwartungskonforme Ergebnisse liefert und Gruppen-Undo-Funktionalität bietet.

13 Literatur

Abowd, Gregory D.; Dix, Alan J. (1992): Giving Undo Attention. In: Interacting with Computers, Vol. 4 (1992) Nr. 3, S. 317–342.

Berlage, Thomas; Genau, Andreas (1993): A Framework for Shared Applications with a Replicated Architecture. In: Proceedings of the UIST '93. ACM Press, New York 1993, S. 249–257.

Berlage, Thomas; Spenke, Michael (1992): The GINA Interaction Recorder. In: Engineering for Human-Computer Interaction. Hrsg.: Larson, J., Unger, C. North-Holland, Amsterdam u. a. 1992, S. 69-78.

Dourish, Paul; Bellotti, Victoria (1992): Awareness and Coordination in Shared Workspaces. In: Proceedings of the CSCW '92. Hrsg.: Turner, Jon, Kraut, Robert. ACM Press, New York 1992. S. 107–114.

Ellis, Clarence A.; Gibbs, Simon J. (1989): Concurrency control in groupware systems. In Proceedings of the ACM SIGMOD '89 Conference on the Management of Data. ACM Press, New York 1989, S. 399–407.

Greenberg, Saul; Marwood, David (1994): Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the interface. In: Proceedings of the CSCW '94. Hrsg.: Furuta, Richard, Christine Neuwirth. ACM Press, New York 1994, S. 207–217.

Herrmann, Thomas (1994): Software-ergonomische Grundsätze für die Gestaltung von Groupware. In: Ergonomie & Informatik Nr. 23, März 1994, S. 7–12.

Prakash, Atul; Knister, Michael J. (1994): A Framework for Undoing Actions in Collaborative Systems. In: ACM Transactions on Computer-Human Interaction, Vol. 1 (1994) Nr. 4, S. 295–330.

Ressel, Matthias (1995): Kooperative Interaktionsunterstützung in Groupware. In Software-Ergonomie '95. Hrsg.: Böcker, Heinz-Dieter. Teubner, Stuttgart 1995, S. 311-329.

Ressel, Matthias; Nitsche-Ruhland, Doris; Gunzenhäuser, Rul (1996): An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors. In: Proceedings of the CSCW '96 (in Vorbereitung).

Stallman, Richard M. (1981): EMACS, the Extensible, Customizable, Self-documenting Display Editor. ACM SIGOA Newsletter, Vol. 2 (1981) Nr. 1/2, S. 147–156.