Computergestützte Gruppenarbeit

8. Undo von Operationen

Dr. Jürgen Vogel

European Media Laboratory (EML)
Heidelberg

SS 2006

Inhalt der Vorlesung

1. Einführung
2. Grundlagen von CSCW
3. Gruppenprozesse
4. Benutzerschnittstelle
5. Zugriffsrechte und Sitzungskontrolle
6. Architektur
7. Konsistenz
8. **Undo von Operationen**
9. Visualisierung semantischer Konflikte
10. Late-Join
11. Netzwerk-Protokolle
12. Entwicklung von Groupware
13. Ausgewählte Groupware

Inhalt

- Einführung
- Interpretation von Undo
- Design-Optionen
- Undo für verschiedene Konsistenzerhaltungs-Mechanismen
  - Operations-Transformation
  - Objekt-Duplikation
  - Timewarp

Einführung (1)

Undo O_i^{-1}

- mache fehlerhafte/unbeabsichtigte Operation rückgängig
- erlaubt eine Arbeitsweise nach "Try and Error"
- wichtig für Groupware: unvorhersehbare (nebenläufige) oder konfliktäre Operationen anderer Benutzer
- Redo $(O_i^{-1})^{-1}$ = Undo Undo (nicht notwendigerweise $(O_i^{-1})^{-1} = O_i$)

Herausforderung Nebenläufigkeit

- Beispiel: Objekt mit Zustand S, O_i = "lösche Objekt", O_j = "ändere Farbe", $O_i \parallel O_j$ mit Serialisierung $O_i < O_j$
- O_i^{-1} = "stelle Objekt wieder her", aber in welchen Zustand: S oder $O_j(S)$?

➡ die lokal letzte Operation ist nicht immer auch die global letzte

Einführung (2)

Herausforderung kontinuierliche Anwendungen

- zwischen einer Operation O_i und O_i^{-1} / $(O_i^{-1})^{-1}$ hat sich der Anwendungszustand u.U. verändert
- welchen Zustand soll das Objekt haben? Berücksichtigung von Wechselwirkungen zwischen Operationen und bewegten Objekten?

Zusammenfassung

- ➡ die semantische Bedeutung von Undo/Redo kann nicht automatisch erschlossen werden
- ➡ dennoch erfordert benutzerfreundliche Groupware Undo/Redo-Funktionalität

Inhalt

- Einführung
- Interpretation von Undo
- Design-Optionen
- Undo für verschiedene Konsistenzerhaltungs-Mechanismen
  - Operations-Transformation
  - Objekt-Duplikation
  - Timewarp

Interpretation von Undo (1)

Beispiel:
- Objekt mit Zustand S
- O_i = "lösche Objekt",
- O_j = "ändere Farbe"
- $O_i \parallel O_j$ mit Serialisierung $O_i < O_j$

1) **Strikte** Interpretation
- O_i^{-1}: $O_i(S) \mapsto S'$ mit $S' = O_{j1} \circ \ldots \circ O_{jn}(S)$ $\forall\, O_j \parallel O_i$
- lösche O_i und seine Wechselwirkungen aus der Historie
- d.h. im Beispiel oben hat das wiederhergestellte Objekt den Zustand $O_j(S)$
- $(O_i^{-1})^{-1}$ erfordert die Wiederherstellung der Historie / des Zustands nach der ursprünglichen Wirkung von O_i
- orientiert sich an den syntaktischen Konsistenzkriterien

2) **Relaxierte** Interpretation
- O_i^{-1}: $O_i(S) \mapsto S$
- stelle den Objektzustand wieder her, der bei der Erzeugung von O_i gültig war
- alle Operationen $O_j \parallel O_i$ werden ignoriert
- d.h. im Beispiel oben: S → i war sich O_j nicht bewusst; wenn i O_i rückgängig macht, erwartet i also auch nicht den Effekt von O_j
- orientiert sich an der Intentionserhaltung

Interpretation von Undo (2)

Vergleich

- die relaxierte Interpretation ist einfacher zu implementieren
- in der Literatur wird die strikte Interpretation präferiert
- ➡ aber: semantische Richtigkeit kann von keinem Verfahren garantiert werden

Inhalt

- Einführung
- Interpretation von Undo
- Design-Optionen
- Undo für verschiedene Konsistenzerhaltungs-Mechanismen
  - Operations-Transformation
  - Objekt-Duplikation
  - Timewarp

Design-Optionen für Undo (1)

1) Lokales vs. globales Undo

- lokal: Undo nur für die eigenen Operationen erlaubt
- global: Undo auch für empfangene Operationen
- ➡ globales Undo ist universeller, kann aber aus Sicht der entfernten Benutzer unerwünscht sein

2) Lineares vs. selektives Undo

- linear: Reihenfolge beim Undo folgt der umgekehrten Reihenfolge der Ursprungs-Operationen in der Historie
- selektiv: beliebige Auswahl der Ursprungs-Operation
- ➡ selektiv ist universeller, ist aber wegen der möglichen Wechselwirkungen zwischen Operationen schwieriger
- ➡ z.B. Undo "erzeuge Objekt" → gleichzeitiges Undo aller Folge-Operationen

Kombination von global und linear

- ➡ nächste Undo-Operation ist abhängig von lokaler Historie

Design-Optionen für Undo (2)

3) Semantisches vs. syntaktisches Undo

- semantische Kodierung
  - O_i^{-1} enthält alle semantischen Informationen, um den Effekt von O_i rückgängig zu machen
  - z.B. S = "grünes Rechteck", O_i = "ändere Farbe auf blau" → O_i^{-1} = "ändere Farbe auf grün"
  - O_i^{-1} kann wie jede andere Operation auf den aktuellen Zustand angewendet werden
- syntaktische Kodierung
  - O_i^{-1} identifiziert O_i (z.B. über (i,SN_i))
  - lokale Berechnung der notwendigen Aktionen beim Empfang von O_i^{-1}
  - O_i^{-1} ist spezielle Operation im Datenmodell
  - erlaubt abweichende lokale Historien
  - erzeugt Abhängigkeiten innerhalb einer Historie

Design-Optionen für Undo (3)

Konkretes Design ist anwendungsspezifisch

- abhängig von Konsistenzerhaltung und Anwendungsszenario
- ggf. Unterstützung mehrerer Undo-Modi

Achtung bei relativen Operationen (z.B. Text-Editoren)

- O_i^{-1} und $O_j > O_i$ beeinflussen sich
- Beispiel: S = "AB", O_i = "füge '1' bei Index 2 ein", O_j = "füge '2' bei Index 1 ein" → $O_j \circ O_i$ (S) = "A2B1"
- naives Undo mit O_i^{-1} = "lösche Zeichen bei Index 2" führt zu "A21"

Inhalt

- Einführung
- Interpretation von Undo
- Design-Optionen
- Undo für verschiedene Konsistenzerhaltungs-Mechanismen
  - Operations-Transformation
  - Objekt-Duplikation
  - Timewarp

Undo und Operations-Transformation

- Klassifikation: strikt, lokal oder global, linear oder selektiv
- speziell für relative Operationen
- syntaktisches Undo; bei Empfang:
  - erzeuge O_i^{-1} als semantisches Undo von O_i
  - z.B. O_i = "füge Zeichen x bei Index y ein"
    → O_i^{-1} = "lösche Zeichen bei Index y"
  - transformiere O_i^{-1} gegen alle Operationen $O_j > O_i$
  - führe O_i^{-1}' aus

➡ unterschiedliche lokale Historien erfordern syntaktisches Undo

➡ Undo/Redo für Operations-Transformation ist sehr komplex

Undo und Objekt-Duplikation

- Klassifikation: strikt, lokal oder global, linear oder selektiv
- Undo: entferne O_i aus der Historie und berechne neue MCGS

➡ führt u.U. zum Löschen von Objekt-Versionen (wieder erzeugt mit Redo)

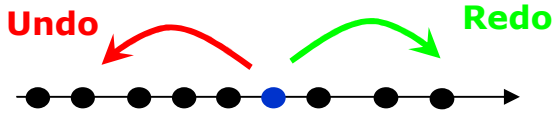
- syntaktisches Undo aufgrund der lokalen Löschung / Erzeugung von Objekten

➡ Undo und Redo haben komplexe Seiteneffekte

➡ schwierige Interpretation für den Benutzer

Undo und Timewarp (1)

- Klassifikation: relaxiert, lokal oder global, linear oder selektiv
- O_i^{-1} ist unabhängig von O_j mit $O_i < O_j < O_i^{-1}$



1) semantisches Undo

- Undo-Operation wird behandelt wie reguläre Operation
- verhindert Abhängigkeiten in der Historie
- Do: gleichzeitiges Erzeugen von O_i und O_i^{-1}
  - O_i^{-1} stellt den vor O_i gültigen Zustand wieder her
  - z.B. O_i = "lösche Objekt" → O_i^{-1} = "erzeuge Objekt"
  - erfordert Verwaltung einer separaten Undo-Historie
- Undo: gleichzeitiges Erzeugen von O_i^{-1} und $(O_i^{-1})^{-1}$
  - $(O_i^{-1})^{-1}$ stellt den vor O_i^{-1} gültigen Zustand wieder her
  - erfordert Verwaltung einer separaten Redo-Historie
  - lösche Redo-Historie nach dem letzten Undo-Kommando
  - erzeuge Undo für jedes Redo

Undo und Timewarp (2)

- funktioniert am Besten für lokales und lineares Undo
- ➡ erlaubt einfache Benutzerschnittstelle mit vorhersehbarem Verhalten

Analyse von Beispielen (Undo-Puzzel)

- $O_i \parallel O_j$ und $O_i \otimes O_j$ (O_i und O_j ändern das selbe Attribut eines Objekts)
- $O_i^{\varnothing}$ und $O_j^{\varnothing}$ löschen das Objekt

1) $O_i < O_j < O_i^{-1}$

- ignoriert O_j beim Undo
- aus der Sicht von i erwartetes Verhalten

2) $O_i^{\varnothing} < O_j < O_i^{\varnothing -1}$

- ignoriert O_j beim Undo
- i hat O_j nie gesehen
- aus der Sicht von j potentiell verwirrend (O_j war hier sichtbar)

Undo und Timewarp (3)

3) $O_i < O_j^{\varnothing} < O_i^{-1}$

- O_i^{-1} kann nicht ausgeführt werden, da es nicht genug Informationen zum Wiederherstellen des Objekts kodiert
- ➡ O_i^{-1} müsste immer (!) den kompletten Zustand enthalten
- statt dessen, ignoriere O_i^{-1}
- ist potentiell verwirrend für i, wenn $O_j^{\varnothing}$ noch nicht sichtbar war

4) $O_i^{\varnothing} < O_j^{\varnothing} < O_i^{\varnothing-1}$

- erzeuge Objekt trotz doppelter Löschaktion
- von i erwartetes Verhalten, da $O_j^{\varnothing}$ nicht sichtbar
- OT würde statt dessen Objekt nur erzeugen, wenn alle Lösch-Operationen rückgängig gemacht würden

Undo und Timewarp (4)

2) syntaktisches Undo

- speziell für kontinuierliche Anwendungen
- O_i^{-1} löscht (oder neutralisiert) O_i aus der Historie und löst einen Timewarp zur Berechnung des neuen Zustands aus
- der Timewarp ist bei allen Instanzen erforderlich → hoher Ressourcenverbrauch
- aufgrund der Wechselwirkung zwischen bewegten Objekten sind starke Seiteneffekte wahrscheinlich
- O_i^{-1} erzeugt Abhängigkeiten in der Historie

Zusammenfassung

- Undo und Redo sind wichtige Funktionen für Groupware
- aufgrund der Abhängigkeiten zwischen Operationen (und Wechselwirkungen zwischen Objekten) ist das Ergebnis von automatischem Undo semantisch fragwürdig
- die Implementierung von Undo / Redo wird stark von den verwendeten Konsistenzerhaltungs-Mechanismen beeinflusst (und kann sehr komplex sein)

Literaturhinweise

- U.M. Borghoff, J.H. Schlichter, Computer-Supported Cooperative Work – Introduction to Distributed Applications, Springer Verlag, Berlin, Heidelberg, New York, 2000, Kapitel 8.3
- Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D. *Achieving Convergence, Causality Preservation and Intention Preservation in Real-Time Cooperative Editing Systems*. In: ACM Transactions on Computer-Human Interaction, Vol. 5, No. 1, pages 63–108, 1998
- J. Vogel, Consistency Algorithms and Protocols for Distributed Interactive Applications, PhD Thesis, University of Mannheim, 2004