# Computergestützte Gruppenarbeit

## 6. Architektur

*Dr. Jürgen Vogel*

*European Media Laboratory (EML)*
*Heidelberg*

*SS 2005*

# 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

- Verteilte Systeme
- Datenmodell
- Verteilungsarchitektur
- Eindeutige Identifizierer

# Verteilte Systeme (1)

Definition

- verteilte Systeme verbinden Ressourcen, die sich auf verschiedenen Rechnern befinden, mithilfe eines Netzwerks
- Ressourcen sind Prozesse, Daten, Benutzer, Prozessoren, Arbeitsspeicher, etc.
- verteilte Systeme bieten einen gewissen Grad von Transparenz (im Sinn von Unsichtbarkeit), z.B. in Bezug auf den Ort und den nebenläufigen Zugriff von Ressourcen
- Beispiele: WWW, SETI@Home und Groupware

Motivation

- Lastverteilung
- Skalierung
- Absicherung gegen Fehler
- Teilen wertvoller Ressourcen
- Reduktion der Antwortzeit für den einzelnen Benutzer

# Verteilte Systeme (2)

Herausforderungen

- Komplexität, z.B. bei der Fehlersuche
- Abhängigkeiten zwischen den Komponenten und Notwendigkeit der Synchronisation, z.B. beim Datenzugriff
- Netzwerkeinflüsse, z.B. durch Verzögerung des Datenaustauschs
- Sicherheit

*"A distributed system is one that stops you from getting any work done when a machine you've never heard of crashes", L. Lamport*

# Groupware als verteiltes System

Groupware

- ist spezielles verteiltes System (räumlich verteilte Benutzer)
- Transparenz ist z.T. nicht erwünscht

Wichtige Anwendungskomponenten

- Benutzerschnittstelle inkl. Endgeräte
- Anwendungslogik inkl. 3K-Funktionen (z.B. Floor Control)
- Datenhaltung (repliziert oder zentralisiert)
- Netzwerk-Kommunikation

# Inhalt

- Verteilte Systeme
- Datenmodell
- Verteilungsarchitektur
- Eindeutige Identifizierer

# Datenmodell

Motivation für ein formalisiertes Datenmodell

- verschiedene CSCW-Anwendungen benötigen ähnliche Mechanismen (z.B. Floor Control, Synchronisation, Netzwerk-Kommunikation etc.)
- formalisiertes Datenmodell erlaubt den Entwurf von anwendungsunabhängigen ("generischen") Verfahren

Anwendungszustand

= Wert aller Attribute im gemeinsamen Kontext zu einem bestimmten Zeitpunkt

- enthält
  - Daten = für den Benutzer sichtbar (z.B. Dokumente in BSCW, Foliensatz im mlb)
  - System-Daten = zur Erbringung von Systemfunktionalität (z.B. Floor Control-Attribute)
- ändert sich durch den Ablauf der Zeit oder durch Ereignisse

# Zustandsänderungen

Zeitablauf

- deterministisch: bestimmt basierend auf momentanem Objektzustand und den physikalischen Gesetzmäßigkeiten
- z.B. Bewegung eines animierten Fahrzeugs
  - Zustand: Position, Geschwindigkeit, Richtung etc.
  - Gesetzmäßigkeiten: Reibung, Gravitation etc.
- kann automatisch vom System berechnet werden
- ➡ erfordert keine zusätzliche Kommunikation bei replizierter Datenhaltung

Interaktion

- nicht deterministisch: Benutzereingaben (z.B. Richtungs-änderung) oder automatisch generierte Ereignisse (z.B. Zufallsexperimente, KI) und Eingaben (z.B. Sensoren)
- kann nicht automatisch vom System berechnet werden
- ➡ erfordert eine explizite Aktualisierung des Zustands

# Klassifikation

| *Interaktion* \ *Zeitablauf* | **diskret**<br>keine zeitbasierten Änderungen | **kontinuierlich**<br>zeitbasierte Änderungen |
|---|---|---|
| **interaktiv**<br>Ereignisse | Shared Whiteboard<br>Gruppen-Editor<br>Instant Messaging | Animation<br>Virtuelle Realität<br>Multiplayer-Spiel |
| **nicht interaktiv**<br>keine Ereignisse | Shared Browsing | Audio- und Video-konferenzen |

# Partionierter Anwendungszustand

Partitionierung

- Aufteilung des Anwendungszustands in unabhängige Objekte (z.B. graphische Objekte, Container, Verkörperung von Benutzern etc.)
- unabhängig bedeutet, dass jede Zustandsänderung genau ein Zielobjekt besitzt
- erfordert eindeutige und beständige Objekt-Bezeichner (ID)
- Vorteile: einfachere Handhabung, partielle Zustandsverfolgung
- aktive vs. passive Objekte
  - *aktiv:* für einen bestimmten Benutzer momentan sichtbar
  - *passiv*: für einen bestimmten Benutzer momentan unsichtbar

Ob eine Partitionierung möglich und sinnvoll ist, hängt von der Anwendung ab.

# Kodierung von Zustandsänderungen (1)

Kodierung als Zustand, Ereignis, Delta-Zustand oder Hinweis.

1) Zustand ("State")

- enthält alle Attribut-Werte eines Objekts zu einem bestimmten Zeitpunkt
- z.B. Typ, Farbe, Geschwindigkeit und Richtung eines animierten Fahrzeugs

2) Ereignis ("Event")

- kodiert die Änderung bestimmter Attribute
- ist nur mit dem dazugehörigen State interpretierbar
- ist bei kontinuierlichen Anwendungen nur zu einem bestimmten (Ausführungs-)Zeitpunkt gültig (z.B. Richtungsänderung eines animierten Objekts)
- kann absolut oder relativ sein: ändere die Position eines Objekts auf Position (x,y) vs. um x Pixel nach rechts

# Kodierung von Zustandsänderungen (2)

3) Delta-Zustand ("Delta-State")

- enthält alle Änderungen in Bezug auf einen bestimmten State
- z.B. ändere die Richtung eines animierten Objekts zum Zeitpunkt $t_1$ nach x und zum Zeitpunkt $t_2$ nach y
- ist nur mit dem dazugehörigen State interpretierbar
- absolute oder relative Kodierung

4) Hinweis ("Cue")

- temporäre Zustandsänderung, die mit einem Event manifestiert werden muss (oder: nicht zustandsändernde Information)
- tritt meist als Folge auf (mit abschließendem Event)
- z.B. Zwischenpositionen eines Objekts, das von einem Benutzer verschoben wird (Endposition als Ereignis)
- ist bei kontinuierlichen Anwendungen nur zu einem bestimmten (Ausführungs-)Zeitpunkt gültig
- ist nur mit dem dazugehörigen Zustand interpretierbar
- absolute oder relative Kodierung

# Kodierung von Zustandsänderungen (3)

Unterscheidung Event und Cue

- erlaubt unterschiedliche Behandlung, z.B. bei der Übertragung über ein Netzwerk: Cue unzuverlässig, Ereignis zuverlässig (→ Zwischenposition vs. Endposition)

Auswahlkriterien

- Umfang: State ≥ Delta-State ≥ Event
- Robustheit: State ≥ Delta-State ≥ Event

# Zusammenfassung

## Definitionen

State, Delta-State und Ereignis bezeichnet man als *Operationen*, da sie den Zustand einer Anwendung dauerhaft ändern.

Der initiale Zustand des gemeinsamen Arbeitsbereichs wird durch den Ablauf der Zeit und durch Operationen geändert. Die Folge der Operationen bildet eine *Historie.*

# Inhalt

- Verteilte Systeme
- Datenmodell
- Verteilungsarchitektur
- Eindeutige Identifizierer

# Verteilungsarchitektur

Die einzelnen Anwendungskomponenten können

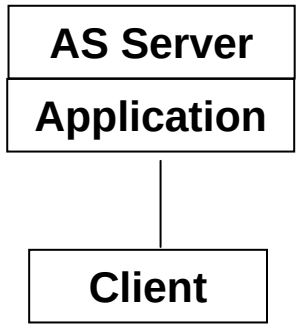
- zentralisiert,
- verteilt oder
- hybrid

realisiert werden.

# Zentralisierte Architektur (1)

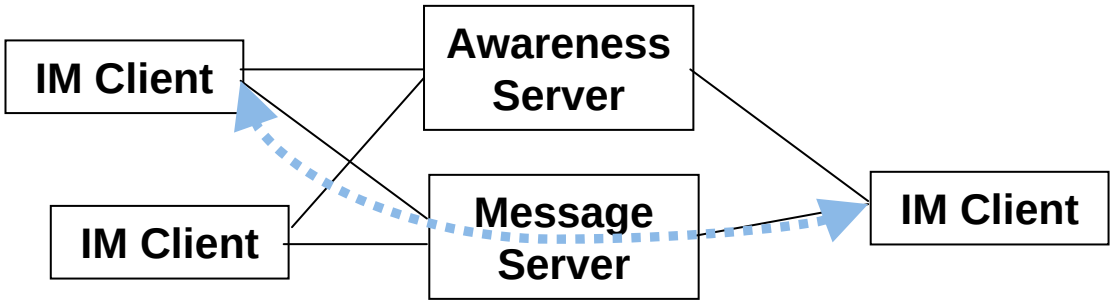
1) Zentralisiert (Client-Sever, C/S)

- Funktionalität und Daten einer Komponente sind auf einem System (Server) konzentriert, das
  - den Zugriff auf den Zustand steuert und für dessen Fort-schreibung verantwortlich ist
  - bestimmte Dienste erbringt
  - den anderen Komponenten wohlbekannt ist
- jeder Benutzer ist über einen Client mit den Server-Komponenten des CSCW-Systems verbunden (GUI + Δ)
- der Server kann auch auf dem System eines Benutzers sein
- Beispiel: Application Sharing
  - Server: meist auf dem Rechner eines Benutzers (z.B. NetMeeting), Übertragung des Bildschirm-inhalts und Entgegennehmen von entfernten Benutzereingaben
  - Client: Anzeigen des Bildschirminhalts und Weiter-leiten von Benutzereingaben bei entfernter Kontrolle



# Zentralisierte Architektur (2)

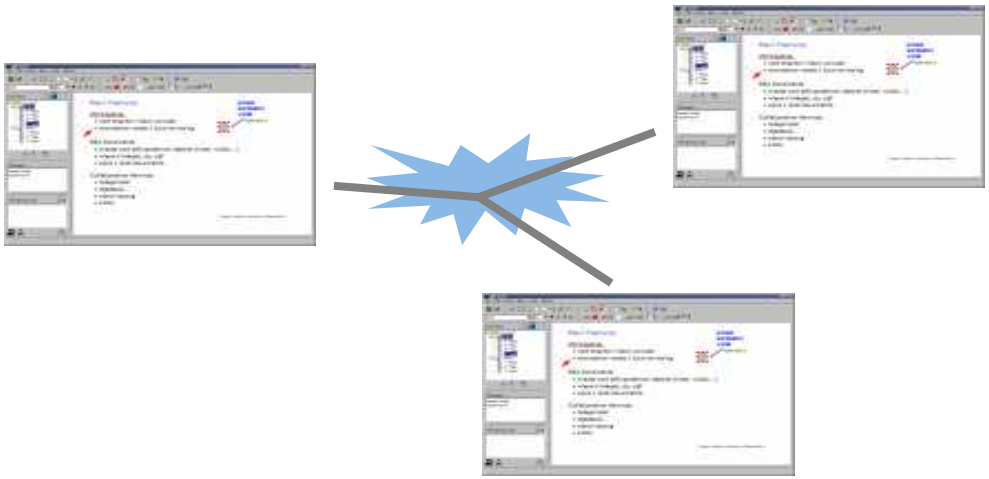
- Beispiel: Instant Messaging
  - Message-Server: Entgegennehmen und Weiterleiten von Nachrichten
  - Awareness-Server: Protokoll der Schreibaktivitäten der Benutzer und Schnittstelle für Statusabfragen
  - IM Client: Eingeben von Nachrichten und Anzeigen von Nachrichten und Awareness-Informationen



# Verteilte Architektur

2) Verteilt (Peer-to-Peer, P2P)

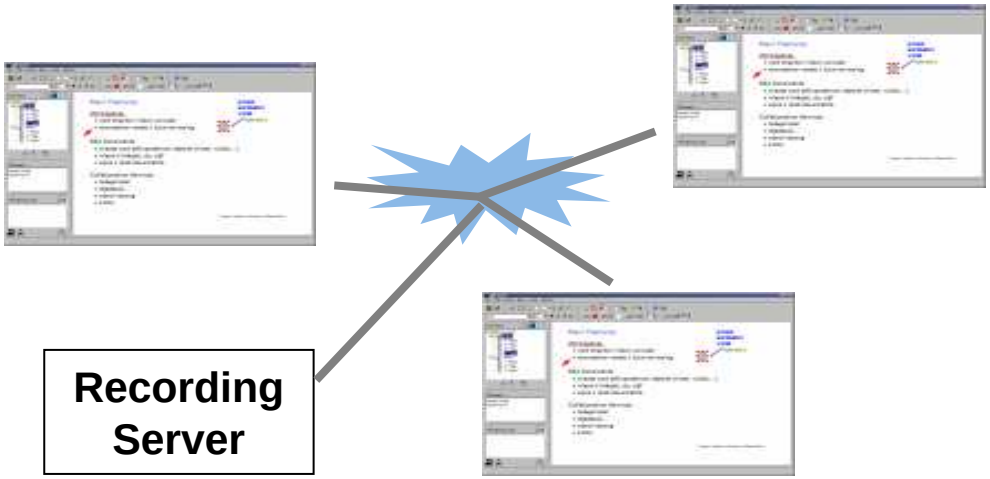
- jeder Benutzer verfügt über eine gleichwertige Anwendungs-instanz (Peer), die
  - alle Systemkomponenten enthält
  - die komplette Funktionalität besitzt
  - den gesamten Anwendungszustand verwaltet
- Anwendungsinstanzen sind voneinander unabhängig
- Beispiel: Shared Whiteboard mlb



# Hybride Architektur

3) Hybrid

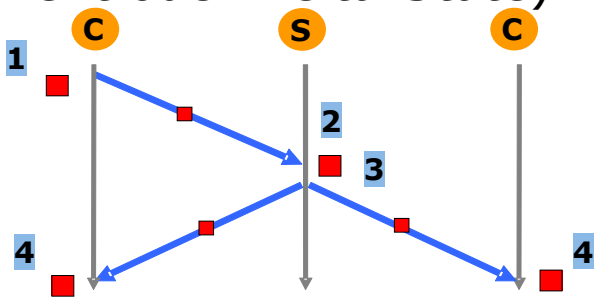
- P2P-System, das durch einzelne Server-Komponenten unterstützt wird
- Beispiel: Shared Whiteboard mlb mit Recording-Server
  - Server zeichnet Datenströme auf bzw. gibt Datenströme wider



# Datenhaltung: zentralisiert vs. repliziert (1)

Zentralisierte Datenhaltung

- der Server verwaltet die Master-Kopie des Zustands
- Ablauf bei Zustandsänderungen
  1. Client sendet Änderung an Server (i.d.R. als Event)
  2. Server sammelt und serialisiert Änderungen aller Clients
  3. Server führt Zustandsänderung aus
  4. Server übermittelt Zustandsänderung an alle Clients (als State, Event oder Delta-State)

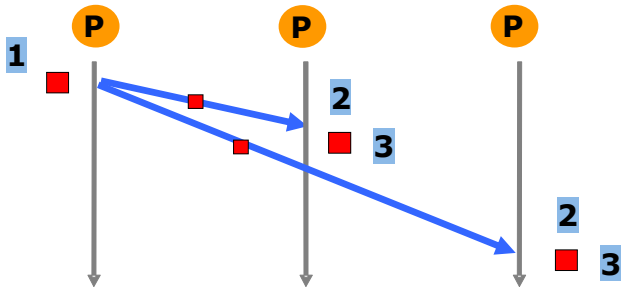



- streng genommen sind Response und Notification Time für alle Benutzer gleich (bei gleichen Netzverzögerungszeiten)
- zur Reduktion der Response Time kann der Client die Zustandsänderung aber auch als vorläufig anzeigen (1)

# Datenhaltung: zentralisiert vs. repliziert (2)

Replizierte Datenhaltung

- jede Anwendungsinstanz verwaltet gleichberechtigt den Zustand (bzw. Teile, z.B. alle aktiven Objekte)
- Ablauf bei Zustandsänderungen
  1. Urheber führt Zustandsänderung aus und sendet diese an alle anderen Anwendungsinstanzen (als State, Event oder Delta-State)
  2. Jede Anwendungsinstanz sammelt und serialisiert alle Zustandsänderungen
  3. Jede Anwendungsinstanz schreibt den Zustand fort




- die Response Time ist wesentlich geringer als die Notification Time für alle Benutzer (bei gleichen Netzverzögerungszeiten)

# Zusammenfassende Diskussion (1)

Zentralisierte Architektur

- Vorteile
  - einfache Koordination der Zustandsänderungen
  - Fehlersuche und –behandlung (z.B. zentrales Backup)
  - Server übernimmt zeit- und speicherintensive Aufgaben und entlastet so die Clients
  - fair in Bezug auf Response und Notification Time (→Spiele)
- Nachteile
  - Server ist potentielles Bottleneck bei vielen Clients, Daten oder Prozessen
  - ein Ausfall des Servers führt zum Systemausfall
  - Netzverkehr konzentriert sich beim Server
  - Betrieb eines Servers bedeutet administrativen Overhead und Verantwortung

# Zusammenfassende Diskussion (2)

Verteilte Architektur

- Vorteile
  - robust gegenüber dem Ausfall einzelner Systeme
  - gute Skalierbarkeit
  - faire Lastverteilung (Speicher und Rechenzeit)
  - geringe Response Time
- Nachteile
  - oft komplexe Implementierung und Fehlersuche, insbesondere aufgrund der replizierten Datenhaltung
  - gleich hoher Speicher- und Rechenbedarf in jeder Anwendungsinstanz
  - hoher Netzverkehr bei allen Systemen
  - Response Time << Notification Time ist evtl. unfair

Hybride Architektur

- als Kompromiss aus obigen Vor- und Nachteilen

# Inhalt

- Verteilte Systeme
- Datenmodell
- Verteilungsarchitektur
- Eindeutige Identifizierer

# Eindeutige Identifikation von Objekten (1)

- Referenzierung von Objekten durch Identifizierer (ID), z.B. zum Zuordnen von Zustandsänderungen
- ID ist Teil des Objektzustands
- ID-Anforderungen
  - global eindeutig (→ auch bei nebenläufiger Erzeugung einer ID) und persistent während der Sitzung
  - ausreichend großer Namensbereich
  - möglichst kleiner Namensbereich → Speicher-Overhead

Mechanismen

- isolierte Auswahl einer zufälligen ID aus Namensbereich
  - Verwendung in vielen Netz-Protokollen, z.B. RTP, SRM, …
  + einfach umzusetzen
  - Kollisionen möglich und Kollisionsbehandlung erforderlich

# Eindeutige Identifikation von Objekten (2)

- isolierte Auswahl aus einem Teilnehmer-spezifischem Namensbereich
  - \- z.B. zusammengesetzte ID aus eindeutiger Teilnehmer-ID (z.B. MAC-Adresse) und lokaler Objekt-ID
  - \+ keine Kollisionen
  - \- ggf. große IDs
- koordinierte Auswahl aus einem gemeinsamen Namensbereich
  - \- z.B. zentralisierte ID-Vergabe durch einen Server (→ hybride Architektur)
  - \+ keine Kollisionen, Persistenz, kleiner Namensbereich
  - \- Nachteile eines Server-Ansatzes

# Literaturhinweise


- A. S. Tanenbaum und M. v. Stehen: verteilte Systeme, Pearson Studium, 2003
- M. Mauve, Distributed Interactive Media, PhD Thesis, University of Mannheim, 2000
- J. Vogel, Consistency Algorithms and Protocols for Distributed Interactive Applications, PhD Thesis, University of Mannheim, 2004
- Mauve, M., Hilt, V., Kuhmünch, C., and Effelsberg, W. *RTP/I - Toward a Common Application-Level Protocol for Distributed Interactive Media*. In: IEEE Transactions on Multimedia, Vol. 3, No. 1, pages 152–161, 2001.