

# Computergestützte Gruppenarbeit

## 10. Late-Join

*Dr. Jürgen Vogel*

*European Media Laboratory (EML)  
Heidelberg*

*FSS 2007*

# 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

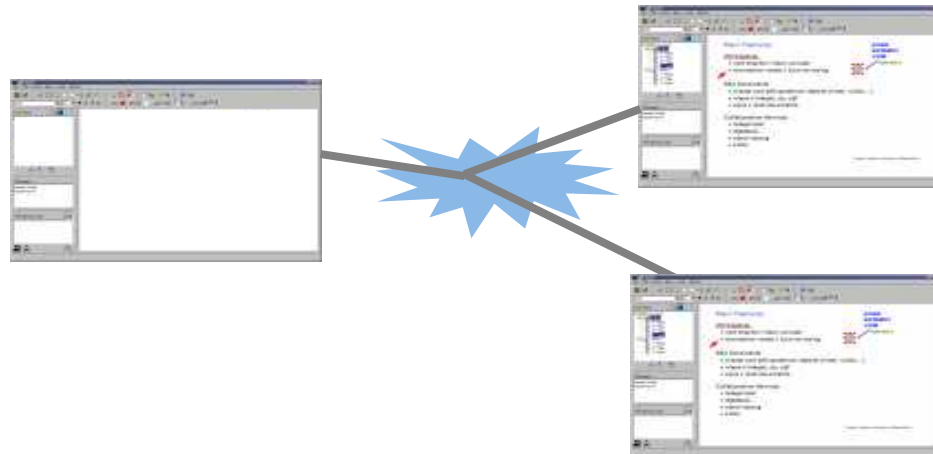
---

- Einleitung
- Design von Late-Join-Algorithmen
- Ausgewählte Verfahren
  - SRM
  - NSTP
  - Late-Join-Dienst mit Politiken
- Konsistenz der Late-Join-Daten

# Einleitung

Groupware unterstützt meist dynamische Mitgliedschaft

- Benutzer können eine Sitzung jederzeit betreten/verlassen
- neue Mitglieder haben alle Zustandsänderungen seit Beginn der Sitzung verpasst
- ➔ initialer (i.d.R. leerer) Zustand  $\neq$  aktueller Zustand
- ➔ Late-Join: Initialisierung neuer Anwendungsinstanzen
- ➔ auch im Fehlerfall (Netzwerkpartitionierung, Absturz, ...)



# Inhalt

---

- Einleitung
- Design von Late-Join-Algorithmen
- Ausgewählte Verfahren
  - SRM
  - NSTP
  - Late-Join-Dienst mit Politiken
- Konsistenz der Late-Join-Daten

# Design von Late-Join-Algorithmen

Ziel: (teilweise) Initialisierung von Instanzen

Definitionen

- Late-Join-Client = zu initialisierende Instanz
- Late-Join-Server = Datenquelle
- Late-Join-Daten = relevante Initialisierungsinformationen
- Late-Join-Anfrage = Anfrage eines LJ-Clients

Die Rollen LJ-Client und LJ-Server können dynamisch sein

# Anforderungen an LJ-Verfahren

- geringe *Initialisierungsverzögerung*
  - Zeitspanne zwischen Eintritt und Möglichkeit zur aktiven Teilnahme
  - kritisch bei umfangreichem Zustand
- geringe *Netzwerk-Last* durch LJ-Daten
- geringe *Anwendungskomplexität* (Daten und Rechenzeit)
- *Robustheit* gegen Fehler
- *Konsistenz* der LJ-Daten
  - nach Abschluss der Initialisierung
  - erfordert ggf. zusätzliche Konsistenzerhaltungs-Mechanismen

# Problembereiche

---

- Selektion eines LJ-Servers
- Umfang und Art der LJ-Daten
- Zeitpunkt der Initialisierung
- Übertragung der LJ-Daten
- Konsistenz der LJ-Daten



# Selektion eines Late-Join-Servers

## 1) Zentralisierte Architektur

- Daten-Server ist auch LJ-Server
- keine zusätzliche Verzögerung durch Selektionsverfahren
- keine zusätzlichen Konsistenz-Mechanismen erforderlich
- keine zusätzliche Anwendungs- und Netzwerklast für unbeteiligte Instanzen
- Nachteile von C/S-Architekturen: Skalierbarkeit, Serverausfall, Netzbelastung beim Server
- ggf. mehr als ein Server und/oder wechselnde Server

## 2) Verteilte Architektur

- replizierte Datenhaltung → mehrere potentielle LJ-Server
- erfordert Verfahren zur Selektion des LJ-Servers
- zusätzliche Verzögerung
- Belastung aller Instanzen, aber auch Lastverteilung
- Konsistenz der LJ-Daten muss sichergestellt werden

# Problembereiche

---

- Selektion eines LJ-Servers
- Umfang und Art der LJ-Daten
- Zeitpunkt der Initialisierung
- Übertragung der LJ-Daten
- Konsistenz der LJ-Daten

# Umfang der Late-Join-Daten

## 1) Komplette Initialisierung

- LJ-Client erhält Daten zur Berechnung des kompletten Zustands
- + geringe Komplexität
- hohe Initialisierungsverzögerung, Netzwerk- und Anwendungsbelastung bei umfangreichem Zustand

## 2) Teilweise Initialisierung

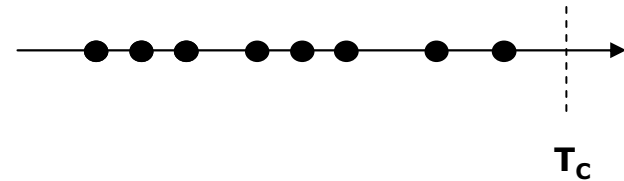
- LJ-Client erhält nur Teile des aktuellen Zustands (z.B. alle aktiven Objekte)
- ➔ zusätzliche Logik: wann sollen welche Teile übertragen werden?
- ➔ *LJ-Politik*

# Art der Late-Join-Daten (1)

## 1) Replay der Operations-Historie

- LJ-Client berechnet aktuellen Zustand aus kompletter Historie

- Übertragung beginnt mit wohlbekanntem initialen Zustand



## Bewertung

- + einfache Implementierung
- + keine zusätzliche Konsistenzerhaltung
- ineffizient, da die Historie irrelevante Informationen enthält:
  - mittlerweile gelöschte Objekte
  - sich überschreibende Änderungen, z.B. mehrfaches Verschieben eines Objekts
- erfordert unbegrenzte Speicherung der Historie
- Rekonstruktion aufwendig, z.B. per Timewarp für kontinuierliche Anwendungen

## Art der Late-Join-Daten (2)

2) Kopie des aktuellen Zustands

- LJ-Client erhält Zustand  $S_{T_C}$  (oder Teile hiervon)



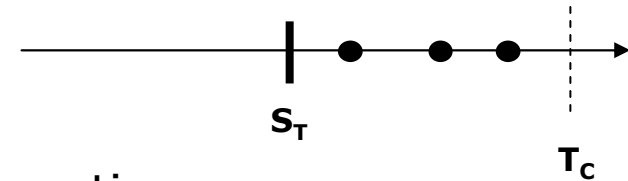
Bewertung

- + effizienteste Form der Übertragung in Bezug auf Initialisierungsverzögerung und Netzwerklast
- Snapshot ggf. aufwendig für LJ-Server
- Konsistenz von  $S_{T_C}$

## Art der Late-Join-Daten (3)

3) Sequenz aus Zustand und Historie

- Kombination aus (1) und (2): übertrage älteren Zustand  $S_T$  und alle folgenden Operationen



Bewertung

- + gegenüber (2) vereinfachte Konsistenzerhaltung
- + Timewarp: älterer Zustand bereits in der Historie des LJ-Servers
- weniger effizient als (2), aber besser als (1) (hängt von  $T_C - T$  ab)

# Problembereiche

---

- Selektion eines LJ-Servers
- Umfang und Art der LJ-Daten
- Zeitpunkt der Initialisierung
- Übertragung der LJ-Daten
- Konsistenz der LJ-Daten

# Übertragung der Late-Join-Daten

## 1) Punkt-zu-Punkt (Unicast)

- direkte Verbindung zwischen LJ-Client und LJ-Server
- + optimiert Netz- und Anwendungsbelastung bezogen auf einzelne Initialisierung
- limitierte Verbindungsanzahl schränkt Skalierbarkeit ein
- Wartezeit erhöht Initialisierungsverzögerung

## 2) Gruppenkommunikation (Multicast)

- Übertragung von LJ-Daten per Multicast
  - alle oder eine Teilmenge der Instanzen empfangen Daten
- + Übertragung kann mehr als einen LJ-Client initialisieren
  - optimiert Netz- und Anwendungsbelastung bezogen auf gesamte Sitzung
- Belastung unbeteiligter Instanzen



# Inhalt

---

- Einleitung
- Design von Late-Join-Algorithmen
- Ausgewählte Verfahren
  - SRM
  - NSTP
  - Late-Join-Dienst mit Politiken
- Konsistenz der Late-Join-Daten

# Scalable Reliable Multicast (SRM)

SRM: Zuverlässiges Multicast-Protokoll, das auf UDP aufsetzt

➔ Reparatur von Paketverlusten mit Feedback Raise-Algorithmus

LJ-Algorithmus von SRM

- LJ-Client hat den Datenverkehr seit Beginn der Sitzung verpasst
- aus Sicht von SRM: kompletter Paketverlust
- ➔ repariere Verluste zur Auslieferung der Paket-Historie

Klassifikation

- verteiltes Verfahren mit dynamischer Selektion des LJ-Servers
- vollständige Initialisierung
- Replay der kompletten Operations-Historie
- sofortige Initialisierung
- Übertragung per Multicast
- anwendungsunabhängiger Ansatz

# Notification Service Transport Protocol (NSTP)

NSTP: C/S-Architektur mit generischem Server

- Verwaltung von Objekten in Interessensbereichen
- abstraktes Datenmodell auf Basis von States und Events

LJ-Algorithmus von NSTP

- LJ-Client kontaktiert NSTP-Server und erhält die Zustände des gewählten Interessensbereichs

Klassifikation

- zentralisiertes Verfahren mit wohlbekanntem LJ-Server
- teilweise Initialisierung
- Initialisierung mit aktuellem Zustand
- sofortige Initialisierung
- Übertragung per Unicast
- anwendungsunabhängiger Ansatz

# Late-Join-Dienst mit Politiken (1)

## Überblick

- generischer Dienst für Anwendungen mit P2P-Architektur auf Basis des allgemeinen Datenmodells
- Ablauf wird vom LJ-Client gesteuert
- dynamische Selektion von Instanzen per EFR
- Anwendung kann Umfang und Zeitpunkt der Initialisierung individuell festlegen
- Initialisierung mit älterem Zustand und folgenden Operationen
- Übertragung per Multicast

# Late-Join-Politiken (1)

Anwendung mit partitioniertem Zustand

Idee

- teilweise Initialisierung
- Berücksichtigung des aktuellen Bedarfs der Anwendung
- ➔ objektspezifische Initialisierungs-Strategie → LJ-Politik

Beispiel: Shared Whiteboard

- LJ-Client benötigt aktive Objekte sofort
- passive Objekte erst, wenn ältere Seiten reaktiviert werden
- ➔ Reduktion der Initialisierungsverzögerung und der Netzlast
- ➔ Voraussetzung: LJ-Client hat Meta-Informationen über den Anwendungszustand (z.B. aktive/passive Objekte)

## Late-Join-Politiken (2)

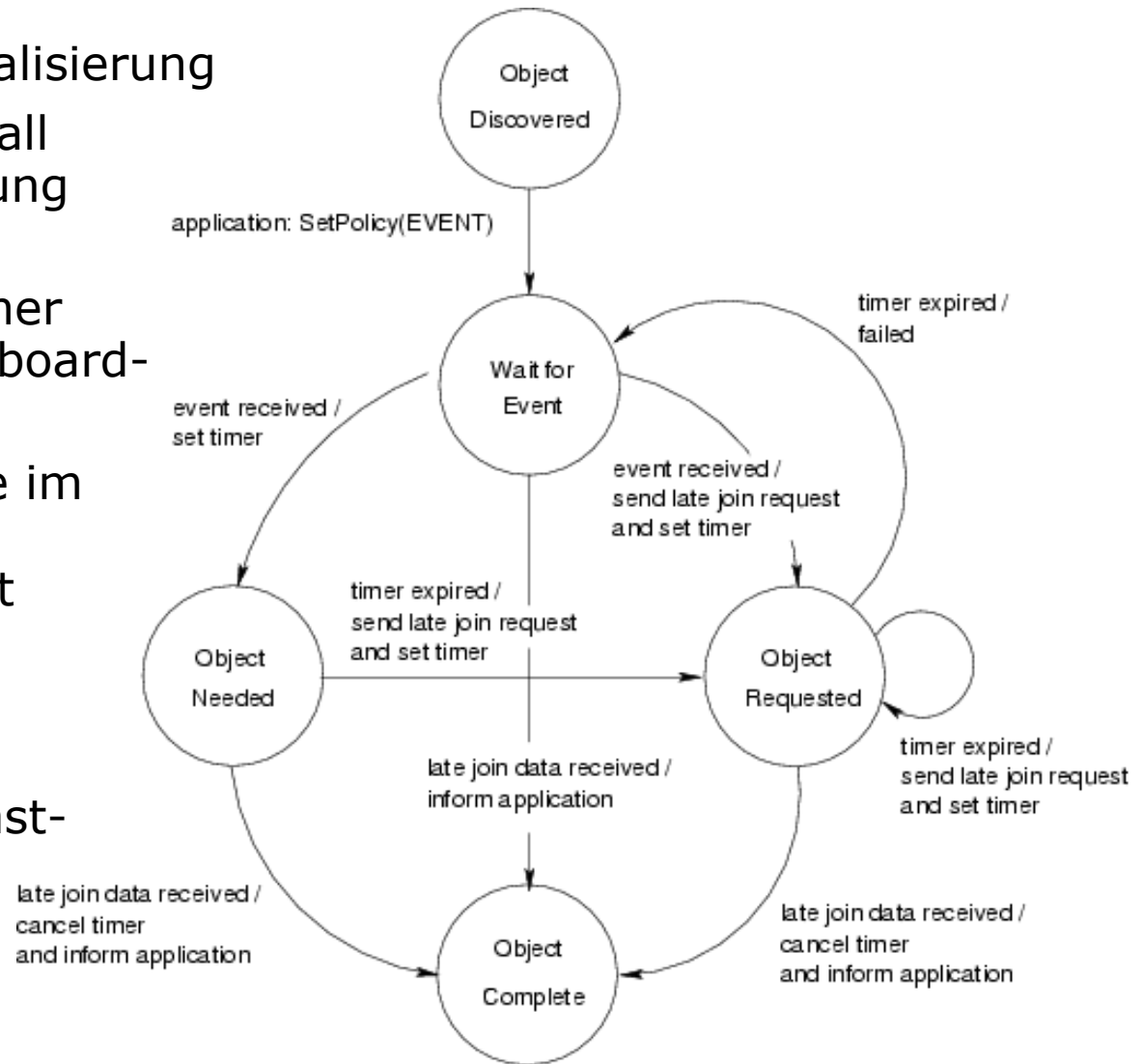
### LJ-Politiken

- Anwendung steuert objektspezifische Initialisierungs-Strategie
- statische oder dynamische Auswahl
- mögliche Politiken:
  - *keine Initialisierung*: Anwendung hat kein Interesse
  - *sofortige Initialisierung*: Objekt wird schnellstmöglich benötigt (z.B. aktive Objekte)
  - *ereignisgesteuerte Initialisierung*: Objekt wird erst im Fall einer Zustandsänderung benötigt
  - *kapazitätsgesteuerte Initialisierung*: Initialisierung bei freien (Netzwerk-)Ressourcen

# Late-Join-Politiken (3)

## Ereignisgesteuerte Initialisierung

- Objekt wird erst im Fall einer Zustandsänderung benötigt
- z.B. Reaktivierung einer älteren Shared Whiteboard-Seite
- ➔ ignoriere Objekte, die im weiteren Verlauf der Sitzung nicht benötigt werden
- ➔ erhöhe Wahrscheinlichkeit für Mehrfachnutzung einer Multicast-Übertragung durch mehrere LJ-Clients



## Late-Join-Politiken (4)

### Bewertung

- + Einsatz von LJ-Politiken kann Initialisierungsverzögerung, sowie Netzwerk- und Anwendungsbelastung stark reduzieren (siehe Simulationsergebnisse unten)
- + anwendungsunabhängige Realisierung auf Basis des Datenmodells
- Verzögerung der Initialisierung kann zu Datenverlust führen: Austritt der letzten besitzenden Instanz



# Übertragung von Late-Join-Daten

Übertragung: (1) Anwendungsdaten, (2) LJ-Anfragen, (3) LJ-Daten

Multicast-Gruppen

<b>1</b>	Anwendungsdaten LJ-Anfragen LJ-Daten	Anwendungsdaten LJ-Anfragen	Anwendungsdaten
<b>2</b>		LJ-Daten	LJ-Daten
<b>3</b>			LJ-Anfragen

- + gleichzeitige Initialisierung mehrerer LJ-Clients
- + geringe Komplexität
- alle Instanzen empfangen LJ-Daten und Anfragen

- + reduzierte Netzbelastung
- alle Instanzen empfangen LJ-Anfragen
- Management-Overhead für 2te Gruppe

- + sehr geringe Netzbelastung durch LJ
- Management-Overhead für 2te und 3te Gruppe

# Late-Join mit 3 Multicast-Gruppen (1)

Gruppen: (1) alle Instanzen, (2) Server-Gruppe, (3) Client-Gruppe

## (1) LJ-Server-Gruppe

- sende LJ-Anfragen zunächst an Server-Gruppe
- Selektion eines bestimmten Servers per EFR
- Antwort an Client-Gruppe (→ Daten) und Server-Gruppe (→ ACK)
- Teilnehmer sind einige wenige Instanzen:
  - günstig platziert → Verzögerung
  - mächtig → Antwortfähigkeit
  - Reduktion der gesamten Netz- und Instanz-Belastung
- Fragestellungen:
  1. Bestimmung der Mitglieder
  2. Fehlerfall: kein passender Server für Anfrage

## Fehlerfall

- zweite Anfragerunde: sende Anfrage an Anwendungsgruppe

## Late-Join mit 3 Multicast-Gruppen (2)

Bestimmung der Mitglieder

- Kriterien
  - Abdeckung aller Objekte → verhindert zweite Anfrage-Runde
  - geringe Teilnehmer-Anzahl Server-Gruppe  $N^{SG}$  → Netzverkehr, doppelte Selektion, beteiligte Instanzen
  - Stabilität der Gruppe (Anzahl Ein- und Austritte) → Overhead
  - Fairness → gleichmäßige Belastung aller Instanzen
  - Komplexität → Overhead
- Algorithmen
  - Verteilt: Informationsaustausch zur Bestimmung der optimalen Zusammensetzung, z.B. Objektbesitz  
→ Komplexität und zusätzliche Netzbelastung
  - Isoliert: individuelle Entscheidung über Eintritt und Austritt
  - Anwendungsbestimmt: Anwendung entscheidet, z.B. wenn Floor Control zum Einsatz kommt

## Late-Join mit 3 Multicast-Gruppen (3)

Isolierte Teilnehmer-Bestimmung

- Austritt: Timeout, wenn keine Anfrage mehr beantwortet wurde

$$t_l = T_m (\gamma O_p/O + (1-\gamma)R_a/R )$$

$T_m$  durchschnittliche Mitgliedschafts-Zeitdauer

$O_p$  Anzahl verfügbarer Objekte

$O$  Anzahl aller Objekte

$R_a$  Anzahl beantworteter Anfragen

$R$  Anzahl aller Anfragen

$\gamma$  Gewichtung

→ Ziel: geringe Anzahl potenter Server

→ minimale Gruppengröße  $N_{\min}^{SG}$

- Eintritt: angepasster EFR-Algorithmus

$$t = T_{\max} (\gamma(1-O_p/O) + (1-\gamma)(1+\log_N x))$$

$T_{\max}$  maximale Wartezeit

$x$  Zufallsvariable

→ je mehr Objekte, desto kürzer  $t$

## Late-Join mit 3 Multicast-Gruppen (4)

### (2) LJ-Client-Gruppe

- Verteilung der LJ-Daten → im Idealfall (Politik!) Mehrfachnutzung
- Instanz wird Mitglied bei Eintritt in die Sitzung
- Dauer der Mitgliedschaft:
  - (1) Empfang aller Daten
  - (2) Timeout (mit Wiedereintritt bei Bedarf)

# Simulationsergebnisse

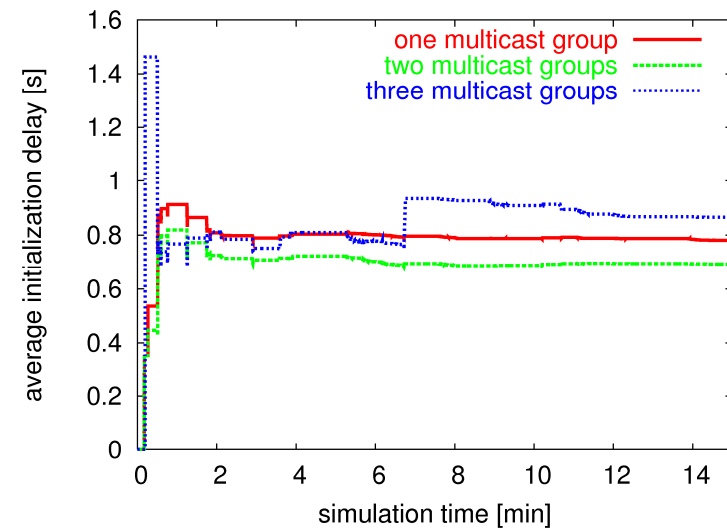
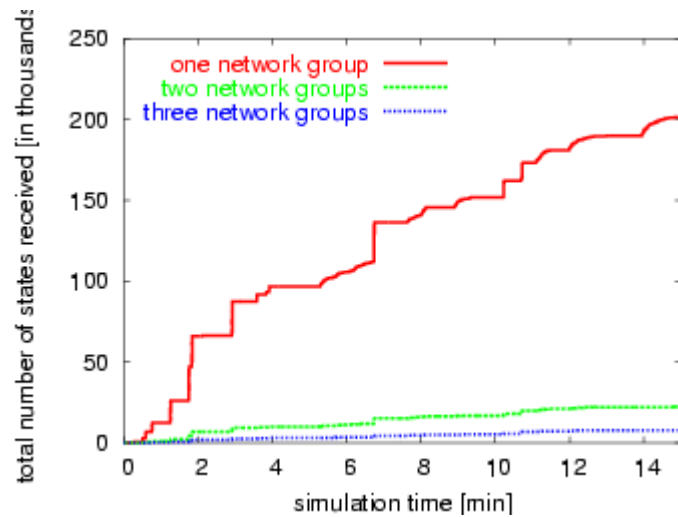
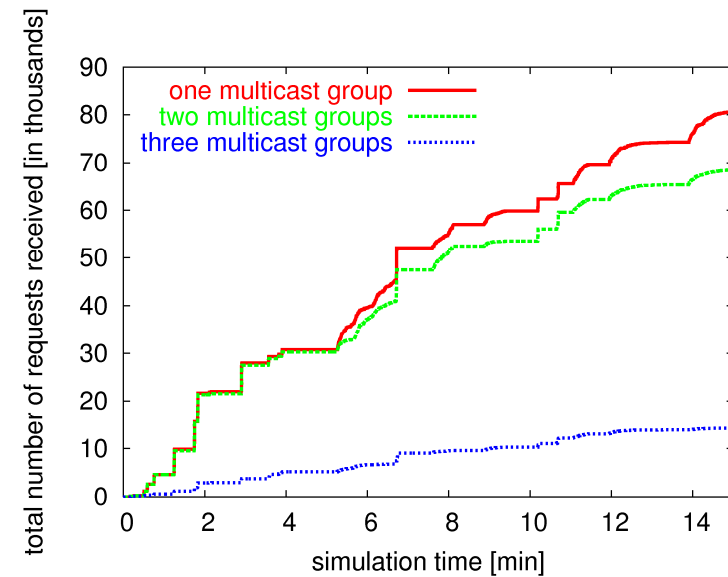
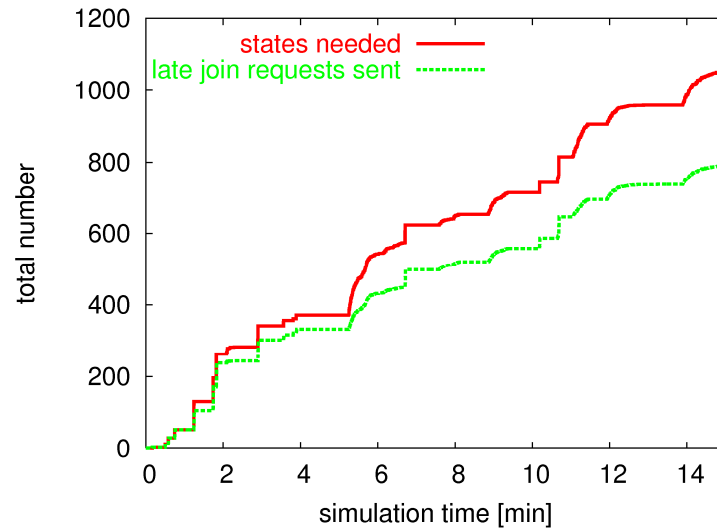
## Simulation von LJ-Algorithmen

- diskrete Ereignis-Simulation mit einfachem Netzwerk-Modell
- Analyse verschiedener Politiken und Übertragungsmethoden
- verschiedene Szenarien mit simuliertem Benutzerverhalten
  - Ein- oder Austritt in die Sitzung
  - erzeuge Objekte oder ändere Objekte
  - aktiviere oder deaktiviere Objekte

## Szenario 1: Shared Whiteboard-Sitzung

- Analyse der Datenübertragung: 1,2 oder 3 Gruppen
- maximal 100 Teilnehmer (Start mit 80)
- 15 Minuten, 14 Seiten, 570 Objekte, 3.300 Events
- Ereignis-Gesteuerte Politik
- Ende-zu-Ende Netzverzögerung  $\in [3\text{ms}, 480\text{ms}]$ ,  $\emptyset$  230ms
- $T_{\max} = 500\text{ms}$
- $N_{\min}^{\text{SG}} = 3$

# Simulationsergebnisse (2)



# Inhalt

---

- Einleitung
- Design von Late-Join-Algorithmen
- Ausgewählte Verfahren
  - SRM
  - NSTP
  - Late-Join-Dienst mit Politiken
- Konsistenz der Late-Join-Daten



# Konsistenz der Late-Join-Daten

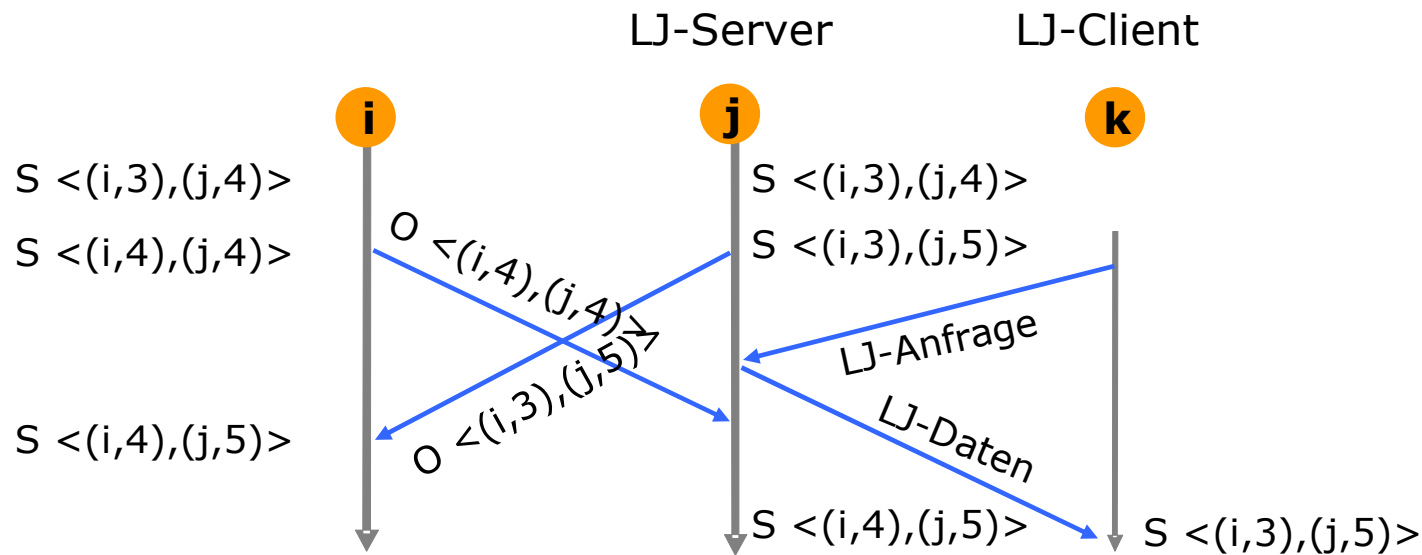
Ziel: LJ-Client hat konsistenten Zustand nach der Initialisierung

## 1) Zentralisierte Architektur

- LJ-Server stellt die vollständige Initialisierung sicher
- keine zusätzlichen Konsistenz-Mechanismen erforderlich

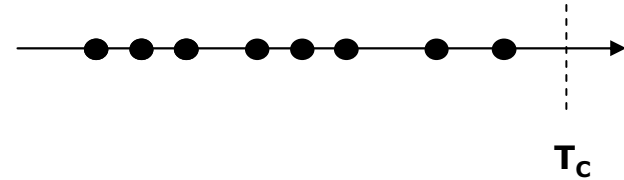
## 2) Verteilte Architektur

- zentrales Problem: keine Konsistenz-Garantie für lokale Zustände



# Replay der Operations-Historie

- LJ-Client berechnet aktuellen Zustand
- Historie muss vollständig sein: Verluste können per SV/SN erkannt werden
- diskrete Anwendungen: Reihenfolge beachten
- kontinuierliche Anwendungen: Ausführungszeitpunkte beachten
- Übertragungsmethode hat keinen Einfluss



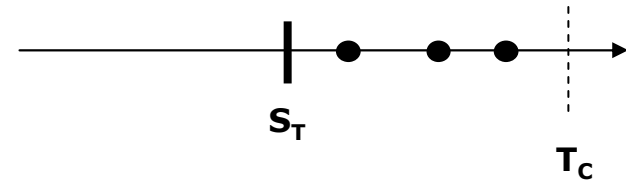
# Kopie des aktuellen Zustands

- nur Instanzen ohne bewusste temporäre Inkonsistenz oder Unvollständigkeit sind potentielle LJ-Server
  - mangelnde Konsistenzgarantie erfordert zusätzliche Maßnahmen
1. iterative Zustandsübertragung
    - empfangende Instanzen überprüfen Konsistenz/Vollständigkeit
    - ggf. erneute Zustandsübertragung
    - bei Verwendung einer Multicast-Gruppe
      - schnelle Reparatur
  2. iterative Zustandsanfrage
    - LJ-Client überprüft anhand periodischer Sitzungsnachrichten Konsistenz/Vollständigkeit
    - ggf. erneute LJ-Anfrage
    - bei Verwendung von zwei oder drei Multicast-Gruppen
      - konzentriert Belastung auf LJ-Client



# Sequenz aus Zustand und Historie

- Voraussetzung vieler Konsistenz-  
erhaltungs-Mechanismen ist eine lokale  
Operations-Historie  
→ Reparatur auf Basis lokaler Informationen
- Initialisierung mit der Historie würde dem LJ-Client die lokale  
Reparatur von temporären Inkonsistenzen ermöglichen  
→ Übertragung der kompletten Historie ist ineffizient
- Idee: Übertrage älteren Zustand plus Teil-Historie
- größtenteils lokale Reparatur möglich:  $\forall O_{j,t^0,t^*}$  mit  $t^* > T$
- in den meisten Szenarien sollte  $T_C - T \approx 1s$  ausreichen



# Zusammenfassung

- Late-Join: Initialisierung neue Sitzungsteilnehmer
- Art und Umfang der Initialisierung: LJ-Politiken
- Übertragung von LJ-Anfragen und -Daten
- "schlechter" LJ-Algorithmus kann hohe Initialisierungsverzögerung, Netzwerk- und Anwendungsbelastung verursachen
- LJ erfordert ggf. zusätzliche Konsistenzerhaltungs-Verfahren

# Literaturhinweise

- Vogel, J., Mauve, M., Hilt, V., and Effelsberg, W. *Late Join Algorithms for Distributed Interactive Applications*. In: ACM/Springer Multimedia Systems, Vol. 9, No. 4, pages 327–336, 2003.
- J. Vogel, Consistency Algorithms and Protocols for Distributed Interactive Applications, PhD Thesis, University of Mannheim, 2004.
- Geyer, W., Vogel, J., and Mauve, M. *An Efficient and Flexible Late Join Algorithm for Interactive Shared Whiteboards*. In: Proc. ISCC, Antibes, France, pages 404–409, July 2000.
- Floyd, S., Jacobson, V., Liu, C., McCanne, S., and Zhang, L. *A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing*. In: IEEE/ACM Transactions on Networking, Vol. 5, No. 6, pages 784 – 803, 1997.
- Patterson, J. F., Day, M., and Kucan, J. *Notification Servers for Synchronous Groupware*. In: Proc. ACM CSCW, Cambridge, MA, USA, pages 122–129, November 1996.