Computergestützte Gruppenarbeit

7. Konsistenz

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

- Einführung in die Synchronisation replizierter Daten
- Konsistenzkriterien
- Klassifikation von Konsistenzerhaltungs-Verfahren
- Ausgewählte Verfahren

Replizierte Datenhaltung

Wiederholung: Groupware mit replizierter Datenhaltung

- jede Anwendungsinstanz verwaltet einen bestimmten Teil des Anwendungszustands
- Zustandsänderungen (Operationen) werden vom Urheber zu allen betroffenen Instanzen übermittelt und führen dort zur Aktualisierung des Zustands

Zentrale Herausforderungen

- **Synchronisation**: der Zustand eines bestimmten Objekts soll bei allen Anwendungsinstanzen identisch sein
  - ➡ eine Operation führt bei allen Objektkopien zu einem identischen Zustand
  - ➡ grundlegende Voraussetzung für gemeinsame Gruppenarbeit
  - ➡ auch von WYSIWIS gefordert
- **Replikations-Transparenz**: dem Benutzer soll (zu einem gewissen Grad) verborgen bleiben, dass er auf einer lokalen Kopie arbeitet

Synchronisations-Mechanismen

1) Konsistenzerhaltung ("Consistency Control")

- stellt sicher, dass Operationen bei allen Instanzen zu einem identischen Zustand führen
- ist kritisch bei gleichzeitigen Änderungen

2) Initialisierung von Anwendungsinstanzen ("Late-Join")

- Übergabe des aktuellen Zustands an Instanzen, die neu in eine bestehende Sitzung eintreten

Anmerkung: Synchronisations-Verfahren sind allgemein in verteilten Systemen erforderlich

Synchronisations-Arten

1) Eng gekoppelte Synchronisation

- jede Operation wird sofort propagiert
- minimale Notification Time
- direkte Zusammenarbeit zwischen Benutzern
- häufig bei synchroner Groupware

2) Lose gekoppelte Synchronisation

- Operationen werden gebündelt übertragen
- vorübergehende unbeeinflusste Arbeit einzelner Benutzer
- entspricht dem Zusammenführen ("Mergen") unterschiedlicher Objekt-Versionen
- häufig bei asynchroner Groupware

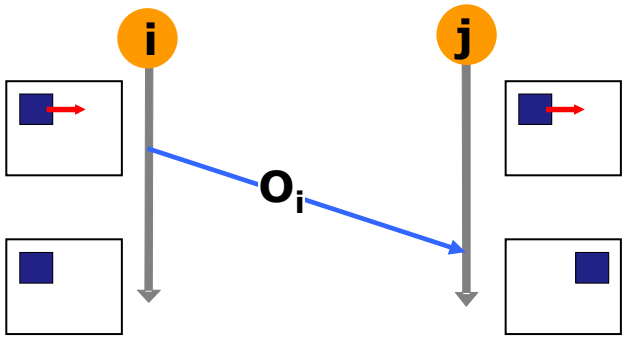
Zunächst betrachten wir ausschließlich (1)

Konsistenzerhaltung (1)

Gefährdung der Konsistenz

1) Szenario: synchrone kontinuierliche Anwendung

- Operation O_i unterliegt einer gewissen Netzverzögerung, so dass Notification Time > Response Time ≥ 0
- Ausführung von O_i bei j führt zu verschiedenen Zuständen
  → **Inkonsistenz**



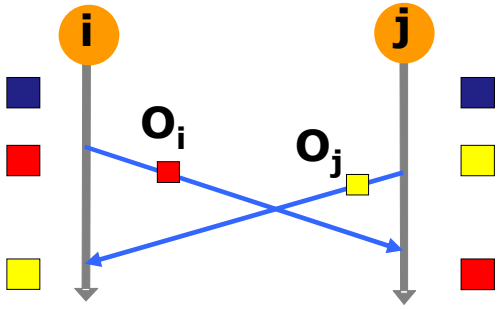
O_i = Stop

➡ Ausführungszeitpunkt einer Operation ist entscheidend

Konsistenzerhaltung (2)

2) Szenario: synchrone diskrete Anwendung

- Benutzer i und j ändern die Farbe eines Objekts fast zeitgleich mit O_i bzw. O_j
- die Netzverzögerung führt zu einer vertauschten Reihenfolge bei der Ausführung (i: O_i O_j, j: O_j O_i) und somit zu verschiedenen Objektzuständen → **Inkonsistenz**



➡ Reihenfolge bei der Ausführung von Operationen ist entscheidend

➡ Anwendung benötigt Mechanismen zur Konsistenzerhaltung

Inhalt

- Einführung in die Synchronisation replizierter Daten
- Konsistenzkriterien
  - diskrete Anwendungen
  - kontinuierliche Anwendungen
- Klassifikation von Konsistenzerhaltungs-Verfahren
- Ausgewählte Verfahren

Diskrete Anwendungen

- diskrete Anwendungen: Zustandsänderung durch Benutzeraktion
- Zustandsänderung O_k
  - als Event oder Delta-State: Update des aktuellen Zustands S
  - als State: ersetze den aktuellen Zustand S
- bei Ausführung einer Menge $\{O_k\}$ auf S_i einer Instanz i bestimmt die Ausführungsreihenfolge den neuen Zustand S'_i

➡ *Ordnung* von Operationen

- z.T. sind Operationen voneinander abhängig, z.B.:
  - O_1 erzeugt ein Rechteck
  - O_2 ändert die Farbe des Rechtecks
  - dann muss O_2 immer nach O_1 ausgeführt werden

- der Einfachheit halber betrachten wir im Folgenden o.B.d.A. einen unpartitionierten Anwendungszustand

Kausale Ordnung (1)

Definition

Seien $O_{i,a}$ und $O_{j,b}$ zwei Operationen der Instanzen i und j, dann gilt $O_{i,a} \rightarrow O_{j,b}$, wenn

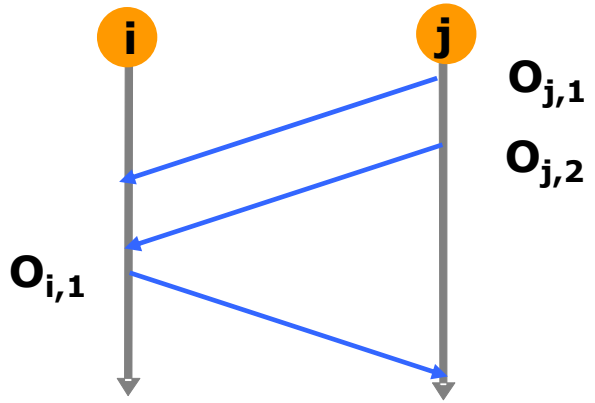
(1) i = j und $O_{i,a}$ wurde vor $O_{i,b}$ erzeugt oder

(2) $i \neq j$ und $O_{i,a}$ wurde ausgeführt, bevor $O_{j,b}$ erzeugt wurde, oder

(3) $\exists\ O_{k,c}$ mit $O_{i,a} \rightarrow O_{k,c}$ und $O_{k,c} \rightarrow O_{j,b}$

Wenn $O_{i,a} \rightarrow O_{j,b}$ bezeichnet man $O_{j,b}$ auch als **abhängig** von $O_{i,a}$

Beispiel

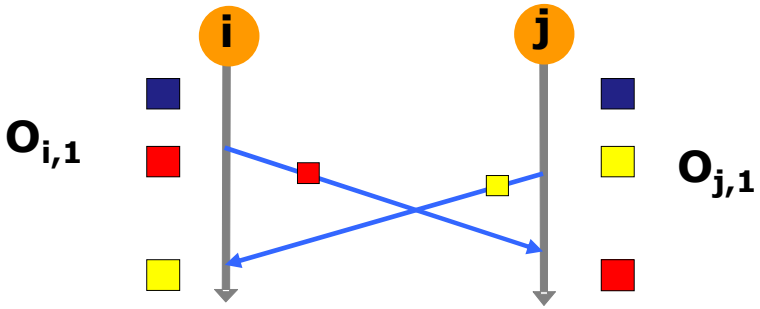


$O_{j,1} \rightarrow O_{j,2}$ und $O_{j,2} \rightarrow O_{i,1}$

Kausale Ordnung (2)

- wenn weder $O_{i,a} \rightarrow O_{j,b}$ noch $O_{j,b} \rightarrow O_{i,a}$, dann sind $O_{i,a}$ und $O_{i,b}$ **nebenläufig** ("concurrent"): $O_{i,a} \parallel O_{j,b}$
- wenn zwei nebenläufige Operationen $O_{i,a}$ und $O_{i,b}$ dieselben Attribute eines Objekts betreffen, dann heißen sie **konfliktär** ("conflicting"): $O_{i,a} \otimes O_{j,b}$

Beispiel



$O_{j,1} \parallel O_{i,1}$ und $O_{j,1} \otimes O_{i,1}$

Kausalität

Definition

Eine Anwendung garantiert **Kausalität** ("Causality"), wenn $\forall$ $O_{i,a}$, $O_{j,b}$ mit $O_{i,a} \rightarrow O_{j,b}$ bei allen Instanzen $O_{i,a}$ vor $O_{j,b}$ ausgeführt wird.

- Kausalität erfordert eine partielle Ordnung für abhängige Operationen
- für Konsistenz ist eine Ordnung auf allen Operationen notwendig

Konvergenz

Definition

Ausgehend von einem identischen Initialzustand S^0 garantiert eine Anwendung **Konvergenz** ("Convergence"), wenn gilt $S_i = S_j \; \forall \; i,j$, nachdem alle Instanzen dieselbe Menge Operationen $\{O_k\}$ ausgeführt haben.

- Konvergenz betrifft den Anwendungszustand nach der Ausführung einer bestimmten Menge an Operationen
- (1) Konvergenz erlaubt Abweichungen bei den einzelnen Instanzen, solange noch nicht alle Operationen empfangen oder ausgeführt wurden
- (2) Konvergenz erfordert nicht (!), dass alle Operationen bei allen Instanzen in der selben Reihenfolge ausgeführt werden müssen
- (3) Konvergenz erfordert nicht (!), dass alle Operationen zuverlässig übertragen werden müssen
- daher sind wir auch daran interessiert, ob der Zustand *korrekt* ist

Korrektheit

Sei P eine virtuelle "perfekte" Instanz, die alle Operationen $O_{i,a}$ einer Sitzung in der eindeutigen Reihenfolge ihrer Erzeugung ausführt.

- P berechnet den Zustand einer nicht verteilten Anwendung

Definition

Ausgehend von einem identischen Initialzustand S^0 garantiert eine Anwendung **Korrektheit** ("Correctness"), wenn gilt $S_i = S_P \; \forall \; i$, nachdem alle Instanzen i dieselben Operationen ausgeführt haben.

- Korrektheit gilt wie Konvergenz nur für Instanzen, die alle erforderlichen Operationen besitzen
- Korrektheit $\Rightarrow$ Konvergenz

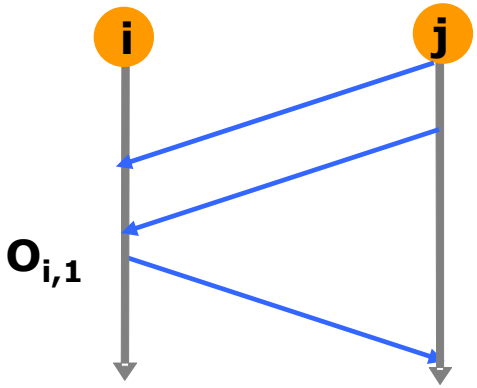
- Überprüfung von Kausalität, Konvergenz und Korrektheit?

Zustandsvektoren

Zustandsvektor ("State Vector")

- für jede Instanz i: Tupel (i,SN_i)
- sei n die Anzahl der Instanzen und $SN \in \mathbb{N}_0$ die Sequenznummer von i, dann gilt: $SV := \langle(1,SN_1),(2,SN_2),\ldots,(n,SN_n)\rangle$
- Definition: $SV[i] := SN_i$
- für jede Operation O_i von i inkrementiere SV[i] (beginnend bei 0)
- jeder Operation O_i ist der inkrementierte Zustandsvektor SV_{O_i} zugewiesen
- jedem Zustand S_i einer Instanz i ist ein Zustandsvektor SV_i zugewiesen, der die auf ihm ausgeführten Operationen repräsentiert

Beispiel



$SV_i = SV_j = \langle(i,0),(j,0)\rangle$

$SV_{O_{j,1}} = \langle(i,0),(j,1)\rangle$

$SV_{O_{j,2}} = \langle(i,0),(j,2)\rangle$

$SV_{O_{i,1}} = \langle(i,1),(j,2)\rangle$

$= SV_i = SV_j$

Überprüfen von Kausalität mit SV (1)

Sei SV_{O_i} der Zustandsvektor von O_i und SV_j der Zustandsvektor von Instanz j, wenn diese O_i empfängt.

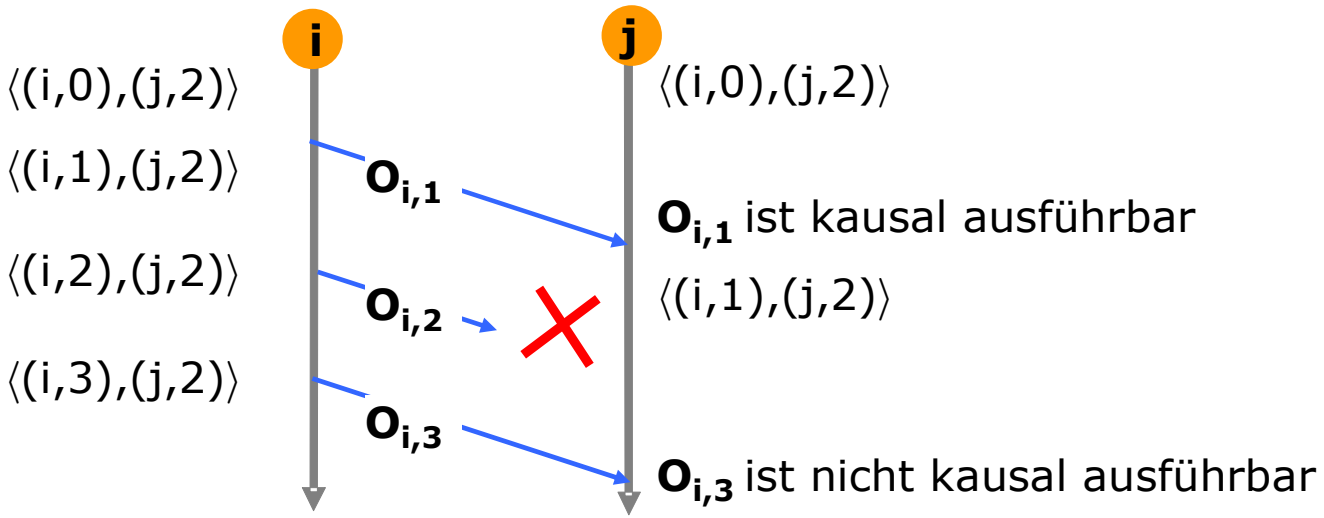
Dann kann O_i von j ausgeführt werden, wenn

(1) $SV_{O_i}[i] = SV_j[i] + 1$ und

(2) $SV_{O_i}[k] \leq SV_j[k]\ \forall\ k \neq i$

In diesem Fall nennt man O_i **kausal ausführbar** ("causally ready")

Beispiel



Überprüfen von Kausalität mit SV (2)

- bevor O_i von j ausgeführt werden kann, wurden alle Operationen, von denen O_i abhängt, von j empfangen und ausgeführt
- ist O_i nicht kausal ausführbar, muss sie gepuffert werden, d.h. die Notification Time erhöht sich
- alle lokalen Operationen sind per Definitionem bei ihrem Urheber kausal ausführbar, d.h. können mit theoretischer Response Time von 0 ausgeführt werden

Überprüfen von Konvergenz mit SV

Für die Überprüfung von Konvergenz wird eine Ordnung auf allen Operationen benötigt.

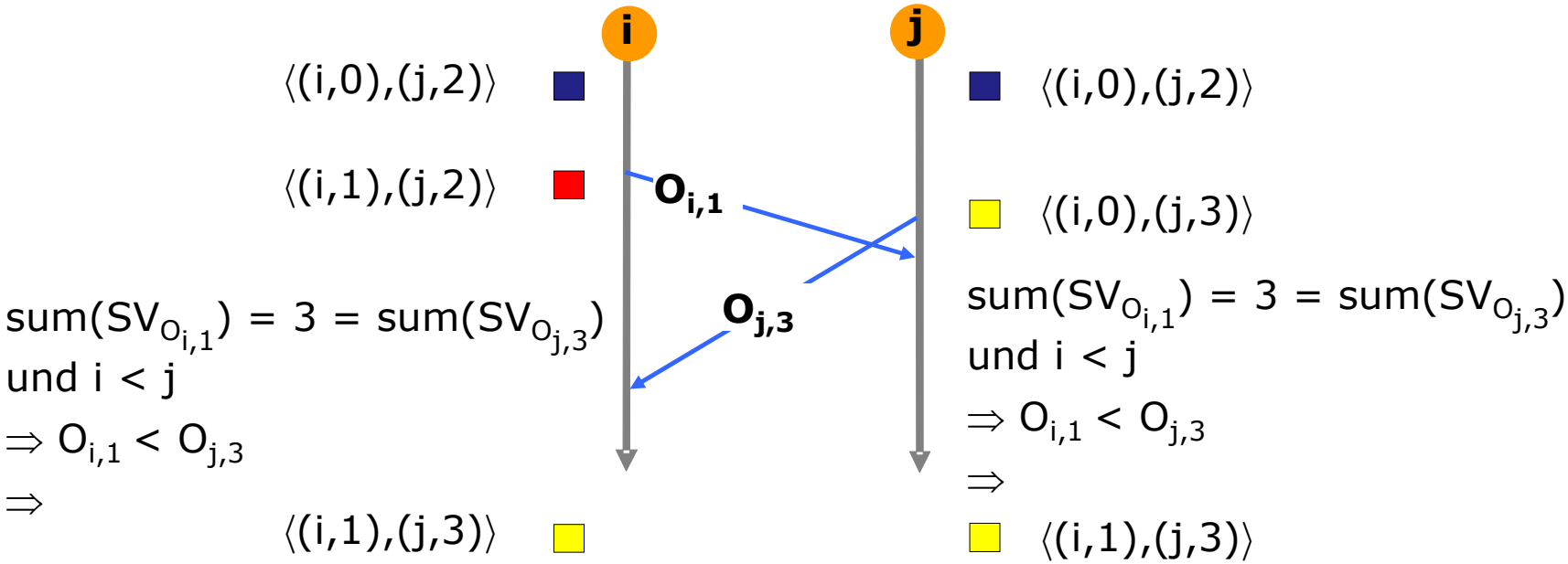
Definition

Seien O_i und O_j zwei Operationen der Instanzen i und j, SV_{O_i} und SV_{O_j} die dazugehörigen Zustandsvektoren und $sum(SV) := \sum_k SV[k]$.
Dann ist $O_i < O_j$ wenn
(1) $sum(SV_{O_i}) < sum(SV_{O_j})$ oder
(2) $sum(SV_{O_i}) = sum(SV_{O_j})$ und $i < j$.

Konvergenz liegt dann vor, wenn alle Instanzen ein Menge von Operationen so ausführen, dass derselbe Zustand erreicht wird, der durch Ausführung dieser Operationen nach obiger Ordnung bestimmt wird.

Anmerkung: Im zweiten Fall sind auch andere Tie-Breaker denkbar.

Beispiel


i
j
⟨(i,0),(j,2)⟩
⟨(i,1),(j,2)⟩
$O_{i,1}$
$O_{j,3}$
⟨(i,0),(j,2)⟩
⟨(i,0),(j,3)⟩
$sum(SV_{O_{i,1}}) = 3 = sum(SV_{O_{j,3}})$
und i < j
$\Rightarrow O_{i,1} < O_{j,3}$
$\Rightarrow$
$sum(SV_{O_{i,1}}) = 3 = sum(SV_{O_{j,3}})$
und i < j
$\Rightarrow O_{i,1} < O_{j,3}$
$\Rightarrow$
⟨(i,1),(j,3)⟩
⟨(i,1),(j,3)⟩

Bemerkungen zur globalen Ordnung

Eine Folge von Operationen, die nach < geordnet ist, genügt auch der Kausalität:

Es gilt $O_i \rightarrow O_j \Rightarrow O_i < O_j$.

Inhalt

- Einführung in die Synchronisation replizierter Daten
- Konsistenzkriterien
  - diskrete Anwendungen
  - kontinuierliche Anwendungen
- Klassifikation von Konsistenzerhaltungs-Verfahren
- Ausgewählte Verfahren

Kontinuierliche Anwendungen

Zustandsänderung durch

- (1) Benutzeraktion: Übertragung als Operation
- (2) Fortschreiten der Zeit: automatische lokale Berechnung

➡ bei Ausführung einer Menge $\{O_k\}$ auf S_i einer Instanz i bestimmen die Ausführungsreihenfolge und die Ausführungszeitpunkte den neuen Zustand S'_i

- der Einfachheit halber betrachten wir im Folgenden o.B.d.A. einen unpartitionierten Anwendungszustand

Zeit in kontinuierlichen Anwendungen

Voraussetzung: jede Instanz besitzt eine Uhr mit hinreichender Genauigkeit und Synchronität

Synchronisation von Uhren

- Network Time Protocol (NTP)
- GPS-Uhren
- ➡ vollständige Synchronität ist praktisch nicht zu erreichen (Ungenauigkeiten in Hardware oder Betriebssystem

Verwendung des Begriffs "Zeit"

- Gegeben sei eine physische Uhr. Dann bezeichnen wir mit Zeit einen bestimmten Wert dieser Uhr.
- Aufgrund von Ungenauigkeiten kann es sein, dass nicht alle Uhren der einzelnen Anwendungsinstanzen diesen Wert gleichzeitig erreichen.

Zeitliche Ordnung

Notationen

- $S_{i,t}$: Zustand der Instanz i zum Zeitpunkt t
- O_{i,t^*}: Operation der Instanz i, die zum Zeitpunkt t^* auszuführen ist

  zur Vereinfachung nehmen wir an, dass die Auflösung der Uhr ausreicht, gleichzeitige Ausführungszeitpunkte zweier Operationen zu verhindern (sonst: Tie-Breaker)
- H: Operations-Historie = Menge aller Operationen in einer Sitzung

Definition

Seien O_{i,t^*_i} und O_{j,t^*_j} zwei Operationen, dann gilt

$O_{i,t^*_i} < O_{j,t^*_j}$ wenn $t^*_i < t^*_j$

Konsistenz

Definition

Eine kontinuierliche Anwendung garantiert **Konsistenz** ("Consistency"), wenn zu jedem Zeitpunkt t bei allen Instanzen i und j, die alle Operationen $O_{k,t^*} \in H$ mit Ausführungszeitpunkt $t^* \leq t$ empfangen haben, die Zustände $S_{i,t}$ und $S_{j,t}$ identisch sind.

Konsistenz

- betrifft den Zustand nach der Ausführung eines bestimmten Teils von H
- ➡ erlaubt Abweichungen bei den einzelnen Instanzen, solange noch nicht alle Operationen empfangen oder ausgeführt wurden
- ➡ erfordert nicht (!), dass alle Operationen bei allen Instanzen in der selben Reihenfolge und zum selben Zeitpunkt t^* ausgeführt werden müssen
- ist unabhängig von der Synchronität der Uhren: gleiche Zustände bei einem bestimmten Wert der gemeinsamen Uhr, unabhängig davon, wann dieser erreicht wird

Korrektheit

Sei P eine virtuelle "perfekte" Instanz, die alle Operationen $O_{i,t^*} \in H$ zum Zeitpunkt t^* ausführt und somit die zeitliche Ordnung einhält.

- P berechnet den Zustand einer nicht verteilten Anwendung

Definition

Eine Anwendung garantiert **Korrektheit** ("Correctness"), wenn zu jedem Zeitpunkt t für alle Instanzen i, die alle Operationen $O_{k,t^*} \in H$ mit Ausführungszeitpunkt $t^* \leq t$ empfangen haben, gilt $S_{i,t} = S_{P,t}$.

- Korrektheit ist wie Konsistenz unabhängig von der Uhren-Synchronität und gilt nur für Instanzen mit den notwendigen Operationen
- legt keine bestimmte Reihenfolge oder Ausführungszeitpunkte fest (der Zustand muss nur so sein als ob!)

Kausalität bei kontinuierlicher Groupware

Kausalität: Ausführungsreihenfolge abhängiger Operationen

- wichtig auch für kontinuierliche Anwendungen, z.B. um zu verhindern, dass eine Operation ausgeführt wird, bevor das Zielobjekt erzeugt wurde
- aber: wenn $O_i \rightarrow O_j$ und k empfängt O_j zuerst, müsste O_j bis zum Empfang von O_i verzögert werden

➡ potentiell Folgefehler (temporäre Inkonsistenzen)

➡ u.U. sollte eine kontinuierliche Anwendung Kausalität daher (für bestimmte Operationen) ignorieren

Zusammenfassung

- diskrete Anwendungen: Kausalität, Konvergenz und Korrektheit
- kontinuierliche Anwendungen: Konsistenz und Korrektheit
- (syntaktische) Konsistenzkriterien sagen nichts über die Semantik von Operationen aus, d.h. ein Zustand kann aus der Sicht des Benutzers unlogisch sein, obwohl ihn das System für korrekt befindet

Temporäre Inkonsistenzen

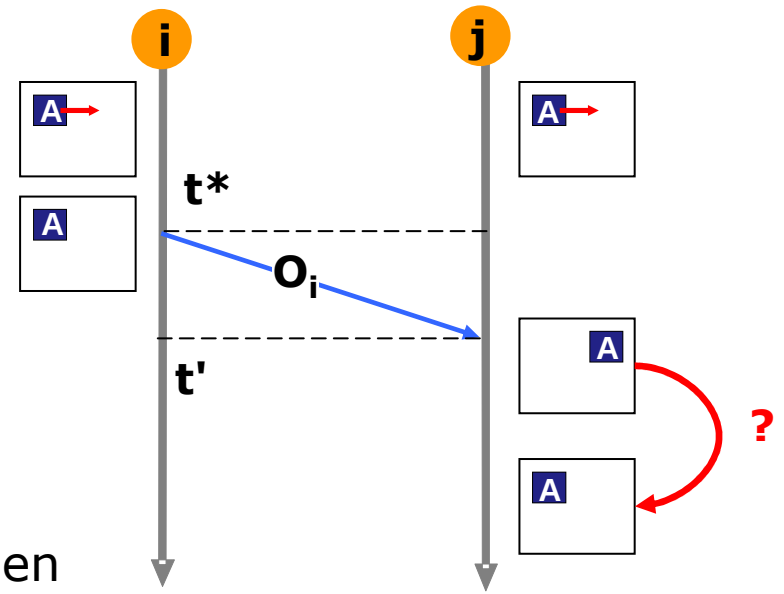
- Konvergenz, Konsistenz und Korrektheit erheben keine Forderungen an Situationen, in denen eine Instanz noch nicht über alle notwendigen Operationen verfügt
- ➡ **temporäre Inkonsistenzen** ("Artefakte") sind möglich

Beispiel

- sei O_{i,t^*} eine Stopp-Operation für ein bewegtes Objekt A
- falls j O_{i,t^*} nach t^* empfängt, ist A auf einer falschen Position
- ➡ bis t' ist S_j korrekt
- ➡ bei t' muss j die Position von A berichtigen, um Korrektheit herzustellen

  (1) abrupte Änderung

  (2) interpolierte graduelle Änderung



Sekundäre Inkonsistenzen (1)

- während einer temporären Inkonsistenz sieht der Benutzer einen falschen Zustand
- das System bemerkt die temporäre Inkonsistenz erst beim Empfang der betroffenen Operation
- der Benutzer kann aber weiterhin Aktionen ausführen

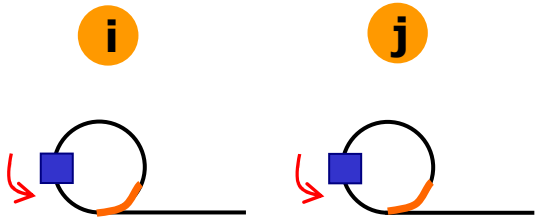
➡ Operationen auf inkorrektem Zustand

➡ **sekundäre Inkonsistenzen** (= semantisch)

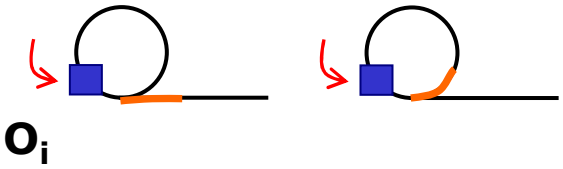
Sekundäre Inkonsistenzen (2)

Beispiel

- Zug-Simulation mit zwei Instanzen i und j

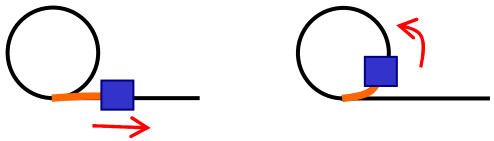


- Zug fährt auf eine Weiche zu, i stellt die Weiche mit O_i



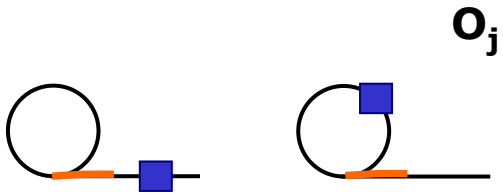
- j empfängt O_i erst nachdem der Zug die Weiche passiert hat

➡ j's Zug ist auf falscher Position



- wenn sich j darüber wundert, dass der Zug nicht auf die Nebenstrecke fährt, zieht er die Bremse mit O_j

➡ wäre O_i rechtzeitig angekommen, hätte j dies nicht getan



Inhalt

- Einführung in die Synchronisation replizierter Daten
- Konsistenzkriterien
- Klassifikation von Konsistenzerhaltungs-Verfahren
  - Soft State vs. Hard State
  - pessimistische vs. optimistische Konsistenzerhaltung
- Ausgewählte Verfahren

Soft State (1)

Soft State-Verfahren

- eine Instanz versendet periodisch den aktuellen Anwendungs-zustand als State an alle anderen Instanzen (=Ankündigung), sofern dies noch nicht von einer anderen Instanz gemacht wurde
- geänderte oder neue Objekte werden implizit durch diese Ankündigungen übermittelt
- (Ankündigungs-)Intervall = Zeitspanne zwischen zwei auf-einanderfolgenden States
- jede Instanz merkt sich den Zeitpunkt der letzten Ankündigung
- empfängt eine Instanz für ein bestimmtes Objekt für mehrere Intervalle keine Ankündigung, wird dieses gelöscht ("Timeout")
- alle Ankündigungen werden unzuverlässig übertragen; Paket-verluste werden durch nachfolgende Ankündigungen repariert

➡ lose gekoppelte Synchronisation

Das Soft State-Verfahren wird oft in Protokollen verteilter Systeme eingesetzt, z.B. in RTP, RSVP, SAP und RIP.

Soft State (2)

Vorteile

+ alle möglichen Fehler und Inkonsistenzen werden durch die Ankündigungen implizit behoben
  - Konsistenzerhaltung
  - Initialisierung von Instanzen
  - Behandlung von Paketverlusten
  - ➡ sehr robust
+ geringe Komplexität und einfach zu implementieren

Nachteile

- hohe Notification Time (abhängig von Intervall und Verlustrate)
- nicht vorhersehbare Notification Time
- häufig temporäre Inkonsistenzen
- wiederholter Paketverlust kann zu falschen Timeouts führen
- hohe Datenrate durch periodische Übertragung des Zustands

Hard State (1)

Hard State-Verfahren

- einmalige, explizite und sofortige Benachrichtigung über neue, geänderte und gelöschte Objekte
- Übertragung von Lösch- und Änderungsoperationen als Event (Cue), neue Objekte als States
- zuverlässige Übertragung aller Operationen
- ➡ enge (oder lose) gekoppelte Synchronisation

- wird häufig auf der Anwendungsebene von verteilten Systemen verwendet, z.B. in Groupware

Hard State (2)

Vorteile

+ geringe und besser vorhersehbare Notification Time
+ geringere Wahrscheinlichkeit für temporäre Inkonsistenzen
+ effiziente Datenübertragung und geringe Datenrate

Nachteile

- hohe Komplexität, die Anwendung benötigt
  - Konsistenzerhaltungs-Mechanismen zur Behandlung verspäteter oder falsch sortierter Operationen
  - Late-Join-Verfahren
  - zuverlässiges Transportprotokoll
- ➡ aufwendige Implementierung und Fehlersuche

Wenn nicht anders angegeben, verwenden die nachfolgend vorgestellten Konsistenzerhaltungs-Mechanismen Hard State.

Inhalt

- Einführung in die Synchronisation replizierter Daten
- Konsistenzkriterien
- Klassifikation von Konsistenzerhaltungs-Verfahren
  - Soft State vs. Hard State
  - pessimistische vs. optimistische Konsistenzerhaltung
- Ausgewählte Verfahren

Pessimistische Konsistenzerhaltung

- explizite Kontrolle des Objektzugriffs
- gleichzeitige Modifikation eines Objekts durch verschiedene Benutzer wird verhindert (z.B. durch Floor Control mit mutually-exclusive Politik)
- implizite Einhaltung der definierten Konsistenzkriterien

Bewertung

+ keine temporären Inkonsistenzen
- kann zu eingeschränkter Responsiveness führen
- verhindert manche Formen der Kooperation (z.B. gemeinsames Aufheben eines Gegenstands in einer virtuellen Welt)

Optimistische Konsistenzerhaltung

- jeder Benutzer darf zu jeder Zeit alle Objekte lesen und ändern
- gleichzeitige konfliktäre Änderungen sind möglich
- mögliche (temporäre) Inkonsistenzen
- explizite Maßnahmen zur Einhaltung der Konsistenzkriterien

Bewertung

+ unterstützt alle Kooperationsformen
+ hohe Responsiveness
+ gut geeignet für Szenarien mit kurzer Notification Time und geringer Wahrscheinlichkeit für echten parallelen Zugriff
- temporäre (primäre) Inkonsistenzen
- sekundäre Inkonsistenzen