Computergestützte Gruppenarbeit

Übungsblatt 6 - Lösung

Dr. Jürgen Vogel

European Media Laboratory (EML)
Heidelberg

FSS 2007

Eindeutige Identifizierer – Lösung (1)

Entwerfen Sie einen Mechanismus zur isolierten und zufälligen Vergabe von eindeutigen IDs für die Objekte des Anwendungszustands. Gehen Sie dabei insbesondere auf die Behandlung möglicher Kollisionen ein, falls zwei Instanzen die gleichen IDs doppelt vergeben sollten.

1) Zufällige Auswahl einer ID

- Zufallszahlengenerator: Linear Congruenial Generator (LCG), Mersenne Twister
- Systemzeit
- Hash-Funktion (z.B. MD5) mit einem oder mehr Eingabeparametern: Systemzeit, MAC-Adresse, Email-Adresse, …

➡ Vergleich mit den bisher vergebenen IDs zur Vermeidung von Kollisionen!

Eindeutige Identifizierer – Lösung (2)

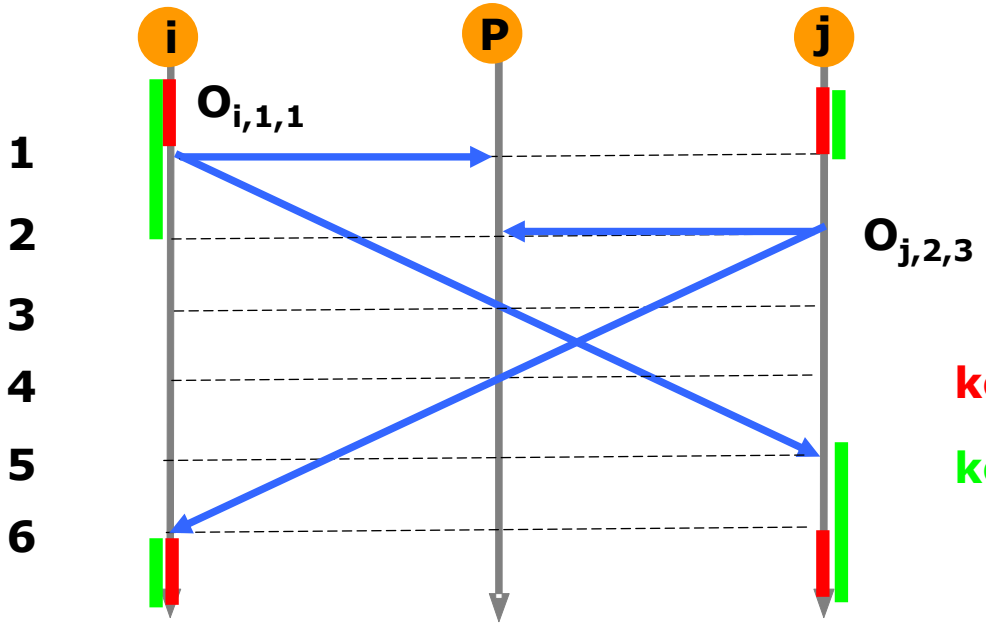
2) Kollisionsbehandlung

- Erkennung einer Kollision
  - Empfang eines Zustands, das ein Objekt mit einer ID erzeugt, für das bereits ein anderes Objekt existiert
  - Achtung: Abgrenzung zu nicht erzeugender Zustands-übertragung (z.B. zur Behandlung von Fehlern) erforderlich
- Behebung einer Kollision
  - Vergabe einer neuen ID für ein Objekt/beide Objekte durch erneute zufällige Auswahl (durch wen? Erzeuger oder Ent-decker?)
  - falls notwendig, Korrektur der ID in der Historie
  - Benachrichtigung aller Instanzen über die geänderte ID
  - ggf. Wiederholung, bis ID eindeutig ist

➡ für Objekt-IDs ist es meist sinnvoller, Kollisionen von vornherein auszuschließen

Konsistenzkriterien – Lösung (1)

1) Verdeutlichen Sie sich die Bedeutung von Konsistenz und Korrektheit für kontinuierliche Groupware anhand folgender Abbildung: Für welche Zeiträume lassen sich Aussagen über die Konsistenz und Korrektheit von i und j treffen? P bezeichne die perfekte Instanz.



konsistent: Vergleich S_i S_j

korrekt: Vergleich S_i S_P und S_j S_P

Konsistenzkriterien – Lösung (2)

2) Geben Sie ein Beispiel dafür an, dass ein Verfahren zwar Konsistenz, aber nicht Korrektheit herstellt.

- Soft State-Verfahren mit unzuverlässiger Datenübertragung (z.B. bei Large Scale Distributed Simulations) → Dead Reckoning
- Objektduplikation / MOVIC
  - Konsistenz durch die einzig mögliche MCGS
  - keine Korrektheit, da es für P keine Operationen $O_i \parallel O_j$ gibt und folglich auch keine Duplikate

3) Sei $\rightarrow$ die kausale und $<$ die globale Ordnung. Gilt auch $O_i < O_j \Rightarrow O_i \rightarrow O_j$?

- nein für $O_i \parallel O_j$

Soft State-Verfahren (1)

Implementieren Sie das in der Vorlesung besprochene Soft State-Verfahren anwendungsunabhängig in Pseudo-Code:

- verwenden Sie für die Verwaltung des Anwendungszustands die folgende Datenstruktur:

```
Type Object {
    Integer id              // eindeutiger Identifizierer
    Binary data             // Anwendungsdaten
    Boolean active          // lokal aktiv?
    Time announced          // letzte Ankündigung
}
Hashtable Object objects    // Hashtabelle mit allen Objekten
                            // Zugriff auf Objekte per Iterator
                            // objects[0],…,objects[n] oder auf
                            // einzelne Objekte per objects(id)
```

- schreiben Sie die folgenden Funktionen
  - **`setInterest(Integer id, Boolean active)`**
    Wird von der Anwendung aufgerufen, um das Objekt mit dem Identifizierer id aktiv/passiv zu setzen. Jede Instanz kündigt nur die Objekte an, die lokal aktiv sind.

Soft State-Verfahren (2)

- **`changeData(Integer id, Binary data)`**
  Wird von der Anwendung zur Veränderung des Objekts id aufgerufen.
- **`update()`**
  Wird periodisch alle T Zeiteinheiten aufgerufen und versendet die lokalen Ankündigungen. Verwenden Sie zu diesem Zweck die Funktion **`send(Integer id)`**.
- **`receive(Integer id, Binary data)`**
  Wird von der Netzwerkschnittstelle aufgerufen, wenn die Ankündigung einer entfernten Instanz zum Objekt id empfangen wurde.

- Objekte, die seit 5T nicht angekündigt wurden, sollen mit der Funktion **`delete(Integer id`**) gelöscht werden.
- Zum Abfragen der aktuellen Zeit können sie die Funktion **`getTime()`** verwenden.

Soft State-Verfahren – Lösung (1)

```
setInterest(Integer id, Boolean active) {
    Object object = objects(id)
    if (object == NULL) return
    object.active = active
}

changeData(Integer id, Binary data) {
    Object object = objects(id)
    if (object == NULL) {
        object = new Object(id)
        objects.add(object)
    }
    object.active = TRUE
    object.data = data
}
```

Soft State-Verfahren – Lösung (2)

```
update() {
    currentTime = getTime()
    for (Integer i = 0; i < n; ++i) {
        Object object = objects[i]
        if (object.active == TRUE) {
            if (currentTime - object.announced > T) {
                send(object.id)
                object.announced = currentTime
            }
        } else {
            if (currentTime - object.announced > 5T) {
                delete(object.id)
            }
        }
    }
}
```

Soft State-Verfahren – Lösung (3)

```
receive(Integer id, Binary data) {
    Object object = objects(id)
    if (object == NULL) {
        object = new Object(id)
        objects.add(object)
    }
    object.data = data
    object.announced = getTime()
    // Anwendung benachrichtigen
}
```