Computergestützte Gruppenarbeit

Übungsblatt 5 - Lösung

Dr. Jürgen Vogel

European Media Laboratory (EML)
Heidelberg

FSS 2007

Floor Control für IDE – Lösung (1)

Für die Software-Entwicklung im Team soll eine Standard-Entwicklungsumgebung zur Groupware erweitert werden. Entwerfen Sie für den (synchronen und asynchronen) Zugriff auf den Programmcode eine passende Floor-Control und diskutieren Sie deren Eigenschaften, insbesondere potentielle Nachteile.

- Lesezugriff nicht reglementiert
  - problematisch bei sichtbaren Zwischenergebnissen
- explizite Kontrolle zur Anforderung der Schreibrechte
  - mutually-exclusive: verhindert paralleles Arbeiten (je nach Granularität)
  - selective: erfordert Verfahren zum Mergen von Code
- Rückgabe durch den Floor Holder
  - Blockade muss durch schnelle Bearbeitung bzw. überschneidungsfreie Aufgaben verhindert werden
- Warteschlange für Schreibzugriffe
  - Priorität: Teilnehmerrolle, Aufgabe (Bugfix > Feature)
  - maschinelle Feststellung der tatsächlichen Priorität ist schwierig; Fairness

Floor Control für IDE – Lösung (2)

- wählbare Granularität: Datei, Klasse, Funktion
  - Relationen zwischen Floors zur Kennzeichnung übergreifender Aufgaben
  - Blockade durch falsch gewählte Granularität
- Visualisierung der Schreib- und Lesezugriffe (Awareness)
  - Benachrichtigungen über Schreibzugriffe und freigewordene Schreibrechte
  - Überlast
- Kombination mit Versionskontrolle sinnvoll
  - Schreibrecht anfordern = aktuelle Version auschecken
  - Schreibrecht zurückgeben = neue Version einchecken
    → zwischenzeitliche Änderungen bleiben verborgen (relaxiertes WYSIWIS)

Floor Control: Warteschlange

Implementieren Sie eine Floor Control-Warteschlange für den Zugriff auf eine Ressource:

- es wird die implizite Floor Control-Politik verwendet
- die Ressource erlaubt maximal **`maxFloors`** gleichzeitige Floors
- gibt es mehr Zugriffswünsche als Floors, werden diese in einer Warteschlange gesammelt, so dass jeder Teilnehmer im Durchschnitt gleich häufig auf die Ressource zugreifen darf (Fairness)
- ein neuer Zugriffswunsch des Teilnehmers **`memberId`** wird durch Aufruf der Funktion **`requestFloor(integer memberId)`** mitgeteilt
- die Freigabe erfolgt über die Funktion **`releaseFloor(integer memberId)`**
- bei Zuteilung eines Floors soll **`grantFloor(integer memberId)`** aufgerufen werden

Implementieren Sie die Funktionen **`requestFloor`** und **`releaseFloor`** in Pseudo Code.

Floor Control: Warteschlange – Lösung

```
integer memberFloor[]     // floors granted for member so far
integer currentFloors     // floors currently granted
queue   requestedFloors   // queues the memberId's for open requests

requestFloor(integer memberId)
  if (currentFloors <= maxFloors)
    ++memberFloor[memberId]
    ++currentFloors
     grantFloor(memberId)
   else
    requestedFloors.add(memberId)

releaseFloor(integer memberId)
  if (requestedFloors.empty())
    --currentFloors
  else
    integer selected = requestedFloors.get(0)
    integer i = 1
    while (i < requestedFloors.size())
      if (memberFloor[requestedFloors.get(i)] < memberFloor[selected])
        selected = requestedFloors.get(i)
      ++i
    requestedFloors.delete(selected)
    ++memberFloor[selected]
    grantFloor(selected)
```

Datenmodell – Lösung (1)

Untersuchen Sie das Datenmodell der folgenden Groupware bezüglich des Anwendungszustands und seiner Partitionierung sowie der Zustandsänderungen und ihrer Kodierung:

Instant Messaging: diskret und synchron

- Anwendungszustand
  - partitioniert in Konversationen, die aus Nachrichten (inkl. ausgetauschten Dateien) bestehen
  - Teilnehmerliste mit Awareness-Informationen
- Zustandsänderungen
  - neue Nachricht (Event/Delta-State: ganze Zeile), Dateiübertagung (Event)
  - Ein- und Ausloggen als Event
  - Aktivität implizit über Nachrichten oder generelle Aktivität am Rechner (Maus- und Tastatur-Input) als Cue oder Event

Datenmodell – Lösung (2)

Email: diskret und asynchron

- Anwendungszustand
  - partitioniert in einzelne Nachrichten
- Zustandsänderungen
  - neue Nachrichten werden einmalig als State verschickt
  - keine nachträglichen Zustandsänderungen

Autorennspiel: kontinuierlich und synchron

- Anwendungszustand
  - statisch: Rennstrecke und unveränderliche Objekte (z.B. Bäume)
  - dynamisch: Rennwagen und veränderliche Objekte (z.B. bewegliche Hindernisse)
  - ggf. partitioniert
- Zustandsänderungen
  - statischer Zustand einmalig als State
  - Änderungen der dynamischen Objekte nach Interaktion (z.B. Lenkbewegung) als Event
  - implizite Berechnung der kontinuierliche Änderungen

Partitionierung – Lösung (1)

1) Was sind die Merkmale von Large-Scale Distributed Simulations

- Simulation geographischer Räume mit sehr hoher Anzahl von bewegten Objekten (>100.000) und Instanzen (>1000)
- jede Instanz verwaltet einen bestimmten Teil der Objekte
- militärische (Schlachtfeld-)Simulation, Online-Rollenspiele
- auch Augmented/Mixed Reality

2) Was versteht man unter "Interest Management" und was ist die Motivation für dessen Verwendung?

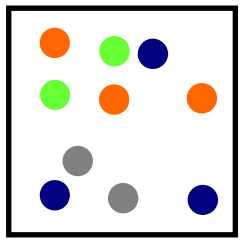
- jeweils nur Auslieferung der relevanten Operationen für die Objekte jeder Instanz
- dadurch Reduktion von Netz- und Rechenbelastung und bessere Skalierbarkeit
- Definition von instanzspezifischen Filtern ("Interest Expressions"), z.B. geographische "Domain of Interest (DOI)"
- die DOI kann sich ändern, z.B. wenn sich das Objekt bewegt
- Anwenden des Filters durch den "Interest Manager (IM)"

Partitionierung – Lösung (2)

3) Welche beiden grundsätzlichen Möglichkeiten gibt es, eine große Menge an simulierten Objekten so zu partitionieren, dass eine bestimmte Instanz nur die Zustände einer Teilmenge fortschreiben muss? Welche Probleme können jeweils auftreten?

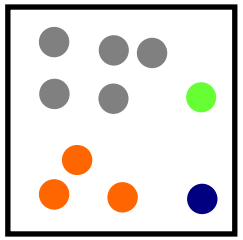
- *objekt-basiert*: jede Instanz simuliert eine bestimmte Anzahl von Objekten mit beliebigem Aufenthaltsort.

  Problem: Interaktion mit benachbarten Objekten, die von anderen Instanzen betreut werden.



- *grid-basiert*: jede Instanz simuliert eine bestimmte virtuelle Region.

  Problem: bei bewegten Objekten Konzentration in bestimmter Region möglich, d.h., Überlastung einzelner Regionen (z.B. bei Online-Rollenspielen)



Partitionierung – Lösung (3)

4) Was versteht man unter extrinisic / intrinsic Filtering? Wägen Sie die beiden Möglichkeiten miteinander ab.

- extrinsic Filtering: Entscheidung auf Paketebene
- intrisic Filtering: Entscheidung durch Operationssemantik
- extrinsic ist schneller und einfacher zu realisieren (keine detaillierte Anwendungslogik erforderlich), aber auch ungenauer als intrinsic

5) Wägen Sie ab, ob man den IM eher beim Sender oder bei den Empfängern einer Operation einsetzen sollte.

- Sender: reduziert den Netzlast für Operationen, aber Netzlast für Filterdefinition
- Empfänger: Netzlast durch irrelevante Operationen
- Verschiebung der Rechenlast
- Auswahl ist anwendungsabhängig

Partitionierung – Lösung (4)

6) Erklären Sie die Funktionsweise von JPSD und STOW-E

JPSD

- senderseitiges Intrinsic Filtering mit Definition in Prädikatenlogik
- jede Instanz hat einen IM, der für jede ausgehende Operation die Menge der Empfänger berechnet und nur an diese ausliefert

STOW-E

- grid-basierte Aufteilung in Zellen
- alle Instanzen innerhalb eines LANs teilen sich einen IM ("Application Gateway")
- die IM tauschen untereinander aus, welche Zellen für die eigenen Instanzen jeweils von Interesse sind
- daraus kann jeder IM alle Zellen bestimmen, die für mindestens eine entfernte Instanz von unmittelbaren Interesse sind
- Operationen für diese Zellen versendet der IM an alle anderen IM mit hoher Frequenz
- Operationen, die sich auf Zellen außerhalb des unmittelbaren Interesses befinden, werden mit geringer Frequenz gesendet
- aus allen eingehenden Operationen selektiert ein IM diejenigen, die für die eigenen Instanzen von Interesse sind