Middleware - Übung

Tobias Distler, Michael Gernoth, Rüdiger Kapitza

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

Wintersemester 2009/2010

Überblick

Gruppenkommunikation — Übersicht

- Bereitstellung von *Virtual Synchrony*
  - Zusammenschluss von Knoten (→ Gruppenkommunikationsteilnehmer) zu Gruppen
  - Senden von Nachrichten an die Gruppe (anstatt an jeden Knoten einzeln)
  - Alle Knoten erhalten jede
    - an die Gruppe versendete Nachricht
    - gruppeninterne Statusmeldung

    in der selben Reihenfolge

  → Alle Knoten einer Gruppe machen (vermeintlich) synchron Fortschritt
- Grundlegende Dienste
  - Membership-Service
  - Total-Ordering-Multicast
  - Zustandstransfer-Mechanismus
- Beispiele
  - Spread
  - **JGroups**

JGroups — Übersicht

- Flexibel konfigurierbare Gruppenkommunikation in Java
- Information und Sourcen
  - Webseite
    - `http://www.jgroups.org/index.html`
    - Manual und Tutorial
  - CIP: `/local/JGroups/`
- Bibliotheken
  - `jgroups-all.jar`: die eigentliche Bibliothek
  - `commons-logging.jar`: Unterstützung für Logging

Membership-Service

- Problemstellungen
  - Zusammensetzung einer Gruppe kann dynamisch variieren
    - Knoten kommen hinzu
    - Knoten verlassen die Gruppe
  - Fehlersituationen
    - Verbindungsabbruch zu einzelnen Knoten
    - Gruppenpartitionierung
- Aufgabe des Membership-Service

  Benachrichtigung aller Gruppenmitglieder über die gegenwärtige Zusammensetzung der Gruppe
- JGroups: Schnittstelle `org.jgroups.MembershipListener`
  - Benachrichtigung über Gruppenänderung

    ```
    void viewAccepted(View new_view)
    ```
  - Mitteilung eines Ausfallverdachts

    ```
    void suspect(Address suspected_mbr)
    ```

View

- Aktuelle Sicht auf die Gruppe

  Liste aller aktiven Gruppenmitglieder
- Problem
  - Keine gemeinsame Zeitbasis
  - Was bedeutet also „aktuell"?
- Lösung
  - Änderung der Gruppenzusammensetzung: Erzeugung einer neuen View
  - Aktuelle Teilnehmer einigen sich auf die neue View

  → Abfolge von Views fungiert als gemeinsame Zeitbasis
- JGroups: Klasse `org.jgroups.View`
  - Ausgabe der Gruppenmitglieder

    ```
    Vector<Address> getMembers()
    ```
  - Ausgabe der Gruppengröße

    ```
    int size()
    ```

Total-Ordering-Multicast

- Problemstellung
  - Clients sollen ihre Anfragen an einen beliebigen Server senden können
  - Alle Server müssen alle Anfragen in der selben Reihenfolge bearbeiten
    - Bewahrung konsistenter Server-Zustände
    - Bereitstellung konsistenter Antworten (z.B. für Fehlertoleranz)
- Total-Ordering-Multicast: alle aktiven Knoten einer Gruppe bekommen alle Nachrichten in der selben Reihenfolge zugestellt
  - Interne Algorithmen, die
    - jeder Nachricht eine eindeutige Sequenznummer zuweisen (→ totale Ordnung)
    - sicherstellen, dass jeder aktive erreichbare Knoten jede Nachricht erhält
    - jeder Knoten die Nachrichten in der richtigen Reihenfolge an die Anwendung weiter gibt
  - Hinweis

    Zustellung jeder Nachricht an **alle** Gruppenmitglieder; also auch an den Knoten, der die Nachricht ursprünglich gesendet hat

Nachricht

`org.jgroups.Message`

Message

- Kapselung der eigentlichen Nutzdaten
- Container für Protokoll-Header
- Konstruktoren

```
Message(Address dst)
Message(Address dst, Address src, Serializable obj)
[...]
```

`dst`: Zieladresse; falls `null` → alle
`src`: Ursprungsadresse; falls `null` → durch JGroups ausgefüllt
`obj`: Nutzdaten als Payload

- Wichtigste Methoden
  - Getter für Payload

    ```
    Object getObject()
    ```
  - Erzeugung einer Kopie

    ```
    Message copy()
    ```

Kommunikationskanal

`org.jgroups.JChannel`

JChannel

- Konstruktoren

```
JChannel()                          // Standardkonfiguration
JChannel(File properties)           // XML-Datei
JChannel(String properties)         // Konf. als Zeichenkette
```

- Wichtigste Methoden
  - Verbindungsaufbau

```
void connect(String cluster_name)
```

  - Diverse Getter-Methoden

```
Address getLocalAddress()           // eigene Adresse
String getClusterName()             // Gruppenname
View getView()                      // aktuelle View
```

  - Nachrichtenversand

```
void send(Message msg)
void send(Address dst, Address src, Serializable obj)
```

Hinweis: Senden einer Nachricht an alle → `dst = null` setzen

Nachrichtenempfang

- Synchron: am `JChannel`
  - Blockierende Methode (`deprecated`)

```
Object receive(long timeout)
```

  - Beispiel

```
Object object = channel.receive(0); // wait forever
if(object instanceof Message) {
    Message msg = (Message) object;
    [...] // extract data with 'msg.getObject()'
} else if(object instanceof View) {
    [...] // handle view
} else {
    [...] // handle events
}
```

- Asynchron: per `org.jgroups.MessageListener`

```
public interface MessageListener {
    void receive(Message msg);
    [...] // siehe spaeter
}
```

Kombinierte Listener und Adapter

- Kombinierte Schnittstelle: `org.jgroups.Receiver`

```
public interface Receiver extends MembershipListener,
                                  MessageListener {}
```

- Erweiterte Adapterklasse: `org.jgroups.ExtendedReceiverAdapter`
  - Implementiert (unter anderem) `Receiver`
  - Eigene Receiver-Klasse als Unterklasse von `ExtendedReceiverAdapter`
- Beispiel

```
public class TestReceiver extends ExtendedReceiverAdapter {
    public void receive(Message msg) {
        System.out.println("received message " + msg);
    }

    public void viewAccepted(View new_view) {
        System.out.println("received view " + view);
    }
}
```

- Registrierung am `JChannel`

```
void setReceiver(Receiver r)
```

Zustandstransfer

- Problem
  - Knoten bearbeiten alle Anfragen, um ihre Zustände konsistent zu halten
  - Was ist mit Knoten, die
    - später hinzu kommen, also nicht alle Anfragen kennen
    - mit der Bearbeitung der Anfragen nicht hinterher kommen oder
    - aufgrund eines Fehlers über kaputte Zustandsteile verfügen?
- Lösung: Zustandstransfer
  - Die Gruppenkommunikation sorgt dafür, dass ein Knoten (z.B. beim Gruppenbeitritt) eine Kopie des aktuellen Zustands erhält
  - Der aktuelle Zustand stammt von einem Knoten aus der Gruppe
- JGroups: zusätzliche Methoden des `org.jgroups.MessageListener`
  - Bereitstellung des eigenen Zustands

```
byte[] getState()
```

  - Setzen des lokalen Zustands auf `state`

```
void setState(byte[] state)
```

→ Teil von `ExtendedReceiverAdapter`

Gruppenkommunikation

Gruppenkommunikation (JGroups)

Aufgabe 5

Übersicht

- Aufgabe: Implementierung eines Instant-Messenger
- Funktionalität
  - Senden/Empfangen von Nachrichten
  - Anzeigen einer Nachrichten-Historie, die alle jemals gesendeten Nachrichten enthält
- Vorgaben
  - Realisierung als fragmentiertes Objekt (→ FORMI)
  - Jedes Fragment verfügt über den vollen Messenger-Zustand
  - Fragment-Kommunikation mittels JGroups
- Pub-Verzeichnis
  - FORMI-Bibliothek
  - JGroups-Bibliothek

Fragment-Implementierung/-Initialisierung in FORMI

- Implementierung eines eigenen Fragments
  - Bereitstellung einer Remote-Schnittstelle

```
public interface MessengerFragmentInterface
                 extends Remote {...}
```

  - Bereitstellung der Fragment-Implementierung

```
public class MessengerFragment extends FormiFragment
             implements MessengerFragmentInterface {...}
```

- Initialisierung des ersten Fragments

```
IpAddress[] commParams = new IpAddress[1];
commParams[0] = [...] // lokale IP-Adresse;

MessengerFragmentInterface frag = (MessengerFragmentInterface)
    FragmentedObjectFactory.createObject(
        MessengerFragment.class, DefaultFragImplFactory.class,
        commParams, null, null);
```

- Weitergabe von Remote-Referenzen mit Hilfe der RMI-Registry
  - Veröffentlichung einer Remote-Referenz: `{,re}bind()`
  - Beschaffung einer Remote-Referenz: `lookup()`

Hinweise

- FORMI: Fragment-Interface erzeugen
  - `MessengerFragment` implementieren
  - FORMI-Compiler verwenden

```
java -jar /local/formi/lib/compiler.jar \
     -classpath /local/formi/classes:<Fragment-Pfad> \
     -keep messenger.MessengerFragment
```

- JGroups
  - Statische Hilfsmethoden: `org.jgroups.util.Util`
    - Serialisierung eines Objekts (→ Fragment-Zustand)

```
static byte[] objectToByteBuffer(Object obj)
```

    - Deserialisierung eines Objekts

```
static Object objectFromByteBuffer(byte[] buffer)
```

    - ...
  - Hilfsmethode für `JChannel`-Parameter (→ Pub-Verzeichnis)

```
static String getJGroupsProperties(List<IpAddress> members)
```