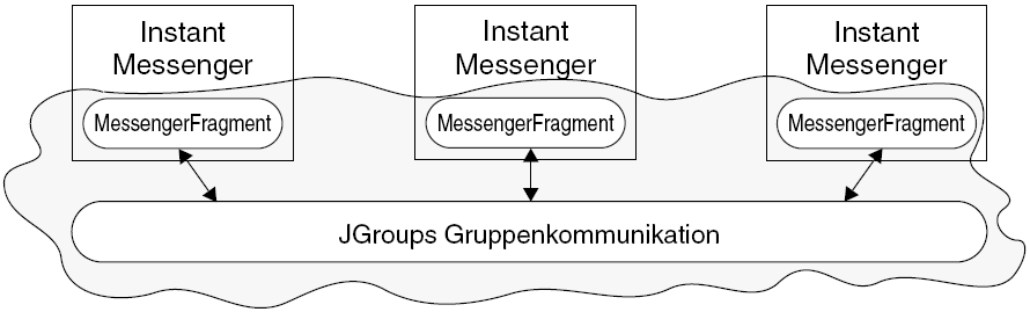
5 Übungsaufgabe #5: FORMI – Instant-Messenger

5.1 Allgemeines

In dieser Aufgabe soll mit Hilfe von *FORMI* (Fragmentierte Objekte basierend auf RMI) und *JGroups* (Gruppenkommunikation) eine verteilte und fehlertolerante Instant-Messenger-Anwendung erstellt werden. Die erforderlichen Quellen und Bibliotheken befinden sich unter `/proj/i4mw/pub/aufgabe5/` bzw. `/local/formi/` und `/local/JGroups/`.



Die Kommunikation zwischen einzelnen Messenger-Instanzen findet mittels Messenger-Fragmenten statt, die *MessengerFragmentInterface* implementieren und mindestens folgende Methoden bereitstellen:

```
public interface MessengerFragmentInterface extends Remote {
    public void sendMessage(String s) throws RemoteException;
    public void setReceiver(ReceiverInterface receiver) throws RemoteException;
}

public interface ReceiverInterface {
    public void receiveMessage(String msg);
}
```

Mit *sendMessage()* lässt sich eine Nachricht an alle Fragmente senden. *setReceiver()* registriert einen Empfänger, der vom Fragment per *receiveMessage()* informiert wird, sobald eine neue Nachricht vorliegt.

5.2 Frontend

Um den Instant-Messenger-Dienst zu initialisieren bzw. um diesem beitreten zu können, ist die Klasse *InstantMessenger* zu erstellen. Sie soll das (lokale) Fragment aufnehmen und das Frontend für die Ein- und Ausgabe bereitstellen. Für den Nachrichtenempfang implementiert sie die Schnittstelle *ReceiverInterface*.
FORMI arbeitet wie RMI mit Remote-Referenzen. Üblicherweise werden diese ebenfalls verteilt verwaltet. Um die Aufgabe etwas zu vereinfachen, soll eine einzelne (globale) RMI-Registry auf dem Rechner des ersten Teilnehmers verwendet werden. Beim Start des Messenger ist zu überprüfen, ob der Dienst (d.h. also mindestens ein Fragment) bereits vorhanden ist. Ist dies der Fall, soll die vorhandene Remote-Referenz genutzt werden, um eine lokale Kopie des Fragments zu erhalten. Andernfalls ist mittels

```
createObject(Class<? extends FormiFragment> objType,
             Class<? extends FragImplFactory> fragImplFac, Object[] commCred,
             Object[] addFactoryArgs, Object[] addFragArgs)
```

der Klasse *FragmentedObjectFactory* ein neues Fragment zu erzeugen und an der RMI-Registry zu binden.

Aufgaben:

→ Implementierung der Klasse *InstantMessenger*

Hinweise:

- Beim Aufruf von *createObject()* müssen die Kommunikationsparameter (*commCred*) des neu zu erzeugenden Fragments übergeben werden. Hierfür kann die Klasse *org.jgroups.stack.IpAddress* zum Einsatz kommen, die allerdings nicht serialisierbar ist und daher entsprechend abgeleitet werden muss (siehe 5.3).
- Die Standard-Factory-Implementierung (*fragImplFac*-Parameter) der *FragmentedObjectFactory* ist *DefaultFragImplFactory*.

5.3 Fragment

Für den verteilten Dienst ist die Basisklasse *MessengerFragment* zu erstellen, welche die Schnittstelle *MessengerFragmentInterface* implementiert.

Messenger-Historie Der Instant-Messenger bietet seinen Benutzern als zusätzlichen Dienst eine *verteilte Nachrichtenhistorie* an, mit der sich alle Nachrichten seit der Entstehung des ersten Fragments einsehen lassen. Hierfür soll das Frontend so erweitert werden, dass die Historie ausgegeben werden kann. Der Aufbau der Historie wird durch das Messenger-Fragment geleistet: Es protokolliert sämtliche ein- und ausgehenden Nachrichten mit und hinterlegt sie als Fragmentzustand, der durch die Klasse *MessengerState* realisiert ist:

```
public class MessengerState {
    public String[] getHistory();
}
```

Kommunikation mit JGroups Um den Dienst fehlertolerant zu machen, soll zur Kommunikation JGroups (`http://www.jgroups.org`) verwendet werden. Innerhalb des Fragments ist hierfür ein *JChannel* zu erzeugen. Die notwendigen Kanal-Parameter finden sich in der Vorgabe (→ *JGroups_Parameter_Dynamic.java*). In den Parametern können zusätzlich alle anderen Kommunikationsteilnehmer angegeben werden, um die Verfügbarkeit zu gewährleisten. Zu Testzwecken kann zunächst mit statischen Parametern gearbeitet werden: Es wird nur die Kontaktadresse des ersten/initialen Fragments in die Konfiguration übernommen.
Nachdem das Fragment als Empfänger gesetzt wurde, kann es dem Kanal mit einer eindeutigen Nummer (z.B. der GUID: *fragImplFactory.guid.toString()*) beitreten. Ereignisse auf dem Kommunikationskanal werden von JGroups über folgende Methoden (→ *ExtendedReceiverAdapter*) signalisiert:

```
public void viewAccepted(View newView)
public byte[] getState()
public void setState(byte[] state)
public void receive(Message msg)
```

viewAccepted() teilt eine Änderung an der Menge der Gruppenkommunikationsteilnehmer mit, so dass sich die Kanalparameter dynamisch aktualisieren lassen. *getState()* wird aufgerufen, wenn ein anderer Teilnehmer den Objektzustand (des Fragments) anfordert. Mit *setState()* setzt JGroups den Objektzustand des lokalen Fragments. *receive()* wird aufgerufen, wenn eine Nachricht empfangen wurde.

Bei der Übertragung der Remote-Referenz wird (ausschließlich) die Fragment-Factory (*DefaultFragImplFactory*) serialisiert. Damit die Adressen neu hinzu gekommener Fragmente bei der Übertragung einer Referenz auch verfügbar sind, müssen sie in dem Factory-Objekt eines jeden aktiven Fragments abgelegt werden. Die bei der Fragmenterzeugung per *createObject()* initial gesetzten Kommunikationsparameter lassen sich später durch

```
public Object[] getCommunicationProperties();
public void setCommunicationProperties(Object[] communicationProperties);
```

verändern. Die Kommunikationsparameter sollten also bei jeder Änderung aktualisiert werden. Hierfür lassen sich die Statusmeldungen von JGroups verwenden (siehe oben).

Aufgaben:

→ Erstellung eines Makefile, das den FORMI-Compiler (`/local/formi/lib/compiler.jar`) aufruft, damit das benötigte Fragment-Interface (*MessengerFragment_Ifc.java*) generiert wird.

→ Implementierung der Klasse *MessengerFragment*

→ Erweiterung des Frontend um die Ausgabe der Nachrichtenhistorie

Hinweise:

- Bei der Fragmenterzeugung muss der Objektzustand explizit mit *JChannel.getState()* angefordert werden.
- Die *FragmentedObjectFactory* verwendet nicht den Standard-Konstruktor, sondern *FormiFragment(FragImplFactory fragImplFactory, Object[] cc)*, der entsprechend in *MessengerFragment* überladen werden sollte.
- JGroups gewährleistet eine zuverlässige Kommunikation, die Historie muss daher nicht noch zusätzlich abgeglichen werden. Da sich die Historie nicht in dem Factory-Objekt befindet, muss sie von jedem neuen Fragment zunächst (implizit) angefordert werden. Dafür sind die JGroups-Methoden *getState()* und *setState()* entsprechend umzusetzen.

5.4 Abgabe: am 14.01.2010 in der Rechnerübung