3 Übungsaufgabe #3: RPC-Semantiken

In den bisherigen Übungsaufgaben wurde davon ausgegangen, dass die für einen Fernaufruf notwendige Kommunikation zwischen zwei Rechnern stets zuverlässig und fehlerfrei abläuft. Diese Annahme lässt sich in realen Netzwerken jedoch nicht aufrechterhalten, so kann zum Beispiel eine Überlastsituation in einem Router zum Verlust von Nachrichten führen. Fernaufrufsysteme müssen mit solchen Fehlerszenarien zurechtkommen, also auch beim Auftreten von derartigen Fehlern korrekt funktionieren. Wie in der Vorlesung erläutert, lässt sich dies durch den Einsatz geeigneter RPC-Semantiken erreichen, die gegebenenfalls dafür sorgen, dass verloren gegangene Nachrichten erneut gesendet werden.

Dennoch existieren Fehler (z. B. dauerhaft unerreichbare Rechner), die auch mit Hilfe von RPC-Semantiken nicht kompensiert werden können. Diese (bei rein lokalen Methodenaufrufen nicht auftretenden) Fehler führen dazu, dass sich Fernaufrufe für den Nutzer nicht uneingeschränkt transparent realisieren lassen. Stattdessen muss der Nutzer in solchen Fällen vom System über die Ausnahmesituation unterrichtet werden. Dies erfolgt in Java RMI durch das Werfen einer `RemoteException`. Aufgrund der Tatsache, dass jede per Fernaufruf erreichbare Methode betroffen sein kann, ist es erforderlich, dass diese Exception Teil einer jeden Methodensignatur ist; siehe beispielsweise `VSBoard`-Schnittstelle:

```
public interface VSBoard extends Remote {
    public void post(VSBoardMessage message) throws RemoteException;
    public VSBoardMessage[] get(int n) throws IllegalArgumentException, RemoteException;
    public void listen(VSBoardListener listener) throws RemoteException;
}
```

Ziel dieser Übungsaufgabe ist es, das bestehende Fernaufrufsystem so zu erweitern, dass temporäre Kommunikationsfehler toleriert werden. Hierzu sollen die beiden Semantiken *Last-Of-Many* und *At-Most-Once* unterstützt werden. Die Tolerierung von Rechnerausfällen wird im Rahmen dieser Übungsaufgabe nicht betrachtet, da sie zusätzliche Mechanismen, zum Beispiel zur persistenten Sicherung des Anwendungszustands, erforderlich macht.

3.1 Last-Of-Many (für alle)

In dieser Teilaufgabe sind die Fernaufruf-Stubs und -Skeletons so zu erweitern, dass sie die Semantik *Last-Of-Many* unterstützen. Diese Semantik ist im Wesentlichen durch die beiden folgenden Merkmale charakterisiert:

- Die von einem Client per Fernaufruf angestoßene Operation wird auf Server-Seite nicht nur einmal, sondern unter Umständen (→ Fehlerfall) mehrmals ausgeführt.
- Dem Client werden nach jeder Ausführung stets die neuesten Ergebnisse zurückgeliefert.

Wie in der Tafelübung vorgestellt, werden diese Eigenschaften der Last-Of-Many-Semantik dabei durch folgende Vorgehensweisen erreicht:

- Erhält ein Client nach dem Absenden einer Anfrage vom Server innerhalb eines vordefinierten Zeitintervalls keine Antwort, sendet er die Anfrage erneut.
- Empfängt der Server eine bereits bearbeitete Anfrage, so führt er die betreffende Operation ein weiteres Mal lokal aus. Das auf diese Weise erhaltene Ergebnis sendet er anschließend an den Client zurück. Die Ausnahme stellen hierbei veraltete Anfragen dar, die vom Server umgehend verworfen werden.
- Ein Client wartet so lange, bis eine Antwort auf die letzte von ihm geschickte Anfrage zurück kommt. Diese reicht er an den Aufrufer weiter. Alle anderen Antwortnachrichten werden verworfen.

Die Umsetzung von Last-Of-Many macht es erforderlich, den Server in die Lage zu versetzen, einzelne Fernaufrufe eines Client eindeutig zu unterscheiden (→ Fernaufruf-ID). Darüber hinaus muss es dem Server möglich sein, die zum selben Fernaufruf gehörigen Anfragenachrichten zu unterscheiden (→ Einsatz von Sequenznummern).

Aufgabe:

→ Implementierung der Last-Of-Many-Semantik

Hinweis:

- Der Nutzer soll beim Start des Fernaufrufsystems einmalig festlegen können, welche Semantik verwendet wird. Bei der Umsetzung von Last-Of-Many ist daher auf Modularität zu achten. Als Standardeinstellung soll *Maybe* zum Einsatz kommen, was der bisherigen Implementierung vor dieser Übungsaufgabe entspricht.
- Auf Client-Seite soll eine maximale Anzahl von gesendeten Anfragen definiert werden können, nach der ein Fernaufruf als erfolglos abgebrochen wird, falls bis dahin kein valides Ergebnis vorliegt. Das Scheitern eines Fernaufrufs ist dem Aufrufer durch eine geeignete `RemoteException` zu signalisieren.

3.2 At-Most-Once (optional für 5,0 ECTS)

In dieser Teilaufgabe soll als weitere Semantik *At-Most-Once* zur Verfügung gestellt werden. Diese Semantik stellt sicher, dass der Server jede zu einer einzelnen Anfrage gehörende Operation nur höchstens einmal ausführt. Hierzu muss der Server in der Lage sein, anhand von Fernaufruf-IDs und Sequenznummern folgende Situationen zu unterscheiden:

- *Neue Anfrage*: Der Server erhält eine Anfrage mit einer ihm noch unbekannten Fernaufruf-ID. → Nach Ausführung der entsprechenden Operation sendet er das Ergebnis an den Client; zusätzlich speichert er das Ergebnis bei sich lokal ab.
- *Alte Anfrage*: Der Server erhält eine Anfrage mit einer ihm bereits bekannten Fernaufruf-ID. → Anstatt die Anfrage erneut zu bearbeiten und die Operation ein weiteres Mal auszuführen, sendet der Server das bereits vorliegende, lokal gespeicherte Ergebnis an den Client; die Antwortnachricht trägt dabei die Sequenznummer der letzten ihm bekannten Anfrage dieses Client.

Da der Server über begrenzten Speicherplatz verfügt, ist eine (beliebig komplexe) Garbage-Collection für nicht mehr benötigte Ergebnisse zu realisieren, die den Speicher des Server zuverlässig aufräumt. Die gewählte Strategie soll dabei nur Ergebnisse verwerfen, deren Fernaufrufe aus Sicht des Server bereits beendet sind.

Aufgabe:

→ Implementierung der *At-Most-Once*-Semantik

Hinweis:

- Wie in vielen Fernaufrufsystemen üblich, darf angenommen werden, dass ein einzelner Client-Stub keinen Bedarf mehr für das Ergebnis eines Fernaufrufs hat, sobald er einen weiteren Fernaufruf durchführt.

3.3 Testen der Implementierung (für alle)

Die Integration der in den vorherigen Teilaufgaben realisierten Semantiken in das bestehende Fernaufrufsystem soll abschließend unter Verwendung der Beispielanwendung („Schwarzes Brett“) aus den vorherigen Übungsaufgaben getestet werden. Dabei sind folgende, in einem realen Umfeld auftretende Fehler zu berücksichtigen:

- Verlust von Nachrichten
- Verzögerung einzelner Nachrichten
- Vervielfachung von Nachrichten

Da die aktuelle Implementierung des Kommunikationssystems aufgrund des verwendeten TCP-Protokolls diese Fehler bisher toleriert, sollen bestimmte Fehlersituationen bewusst herbeigeführt werden. Zu diesem Zweck ist das Kommunikationssystem an geeigneten Stellen zu sabotieren, so dass es Nachrichten verwirft, verzögert und/oder vervielfacht. Hierzu eignet sich insbesondere die Klasse `VSObjectConnection`, da in `{send,receive}Object()` eine direkte Zugriffsmöglichkeit auf einzelne Nachrichtenobjekte besteht. Es ist daher zum Beispiel sinnvoll, die Fehlererzeugung in einer Unterklasse `VSBuggyObjectConnection` zu realisieren.

Aufgaben:

→ Sabotage des Kommunikationssystems
→ Implementierung geeigneter Testfälle

Hinweise:

- Mit den Testfällen sind nicht nur die Fehlerarten bei der Nachrichtenübermittlung einzeln, sondern auch Kombinationen aus diesen Fehlern abzudecken.
- Auf welche Weise die geforderten Kommunikationsfehler tatsächlich simuliert werden, ist freigestellt. Die Verwendung einer `VSBuggyObjectConnection` dient in diesem Zusammenhang nur als Vorschlag.
- Es darf angenommen werden, dass gesendete Anfrage- und Antwortnachrichten entweder vollständig oder überhaupt nicht auf der Gegenseite ankommen. Der teilweise Verlust von Nachrichten bzw. von einzelnen Objekten einer Nachricht muss nicht betrachtet werden.

Abgabe: am Mi. 3.6.2015 in der Rechnerübung

Die für diese Übungsaufgabe erstellten Klassen sind in einem Subpackage `vsue.faults` zusammenzufassen.