5 Übungsaufgabe #5: RPC-Semantiken

5.1 Allgemeines

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

Dennoch existieren Fehler (z. B. dauerhaft unerreichbare Hosts) 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, z. B. (mit dem Übungspendant *VSRemoteException*):

```
public interface VSRemoteObject extends VSRemote {
    public String getString() throws VSRemoteException;
    public String concat(String string) throws VSRemoteException;
}
```

Ziel dieser Übungsaufgabe ist es, das bestehende Fernaufrufsystem so zu erweitern, dass temporäre Kommunikationsfehler toleriert werden. Es es ist nicht erforderlich die Tolerierung von Rechnerausfällen zu unterstützen.

5.2 Last-of-Many (für alle)

Zunächst ist die *Last-of-Many*-Semantik zu implementieren, die im Wesentlichen zwei Charakteristika beinhaltet:

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

Wie in der Übung vorgestellt, umfasst *Last-of-Many* folgende Vorgehensweisen:

- 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 erneut lokal aus. Das auf diese Weise erhaltene Ergebnis sendet er anschließend an den Client zurück.
  Ausnahme: Erreicht den Server eine veraltete Anfrage, wird diese umgehend verworfen.
- Ein Client wartet solange, bis eine Antwort auf die letzte von ihm geschickte Anfrage zurück kommt. Diese reicht er an den Aufrufer weiter. Alle anderen Antwortnachrichten werden verworfen.

Aufgaben:

→ Implementierung der *Last-of-Many*-Semantik

Hinweise:

- Um diese Semantik realisieren zu können, müssen einzelne Fernaufrufe eines Clients auf Server-Seite eindeutig unterscheidbar sein (→ Fernaufruf-ID). Gleiches gilt für die zu einem einzelnen Fernaufruf gehörenden Anfragen (→ Einsatz von Sequenznummern).
- Für den Nutzer einer *VSRemoteEntity* soll es möglich sein bei ihrer Erzeugung (bzw. während ihrer Initialisierung) festzulegen, ob eine und wenn ja welche RPC-Semantik intern verwendet werden soll. Bei der Implementierung ist daher auf Modularität zu achten.

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

In dieser Teilaufgabe soll die *At-Most-Once*-Semantik umgesetzt werden. Die Hauptforderung besteht dabei darin, dass der Server jede zu einer einzelnen Anfrage gehörende Operation nur einmal ausführt. Dazu muss er folgende Fälle unterscheiden:

- *Neue Anfrage*: Eine neue Anfrage trägt eine unbekannte Fernaufruf-ID und führt zur Ausführung der entsprechenden Operation.
- *Alte, noch in Bearbeitung befindliche Anfrage*: Sobald die Bearbeitung fertig ist, wird die Antwort zurückgesendet.
- *Alte, bereits bearbeitete Anfrage*: Statt die Anfrage erneut zu bearbeiten wird das Ergebnis der vorherigen Ausführung zurückgegeben. Zu diesem Zweck ist es erforderlich, alle Ergebnisse auf Server-Seite zu speichern.

Aufgaben:

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

Hinweise:

- Da der Server über begrenzten Speicherplatz verfügt, ist eine (beliebig komplexe) Garbage Collection für nicht mehr benötigte Antworten zu implementieren, die den Speicher des Servers zuverlässig aufräumt. Dabei sollen (nach Möglichkeit) nur Antworten gelöscht werden, deren Fernaufrufe bereits beendet sind.

5.4 Testen der Implementierung (für alle)

Die Integration der Semantiken in das bestehende Fernaufrufsystem soll abschließend (mit *JUnit*) getestet werden. Dabei sind folgende in einem realen Umfeld auftretende Fehler zu berücksichtigen:

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

Um bestimmte Fehlersituationen bewusst herbeiführen zu können, muss das Kommunikationssystem an geeigneten Stellen sabotiert werden. Hierzu eignet sich insbesondere die Klasse *VSConnection*, da in *{send,receive}Message()* eine direkte Zugriffsmöglichkeit auf einzelne Nachrichten besteht. Es ist daher z. B. sinnvoll, die Fehlererzeugung in einer Unterklasse *VSBuggyConnection* zu realisieren.

Aufgaben:

→ Sabotage des Kommunikationssystems

→ Implementierung geeigneter Testfälle

Hinweise:

- Mit den Testfällen ist nicht nur jede Fehlerart einzeln, sondern auch Kombinationen aus Fehlern abzudecken.
- Es ist freigestellt auf welche Weise die geforderten Kommunikationsfehler tatsächlich simuliert werden. Die Verwendung einer *VSBuggyConnection* dient in diesem Zusammenhang nur als Vorschlag.
- Es ist darauf zu achten, dass jede Methode der in den Testfällen verwendeten Remote-Schnittstellen eine Remote-Exception im *throws*-clause aufweist.

5.5 Abgabe: am 7.7.2010/14.7.2010 in der Übung

Die für diese Teilaufgabe erstellten Testfälle sind in *vsue.tests* abzulegen und alle weiteren Dateien in einem Subpackage *vsue.rpc* zusammenzufassen.