Übungsaufgabe #3: Fernaufrufsemantiken

In den bisherigen Aufgaben wurde davon ausgegangen, dass die für einen Fernaufruf notwendige Kommunikation zwischen zwei Rechnern stets fehlerfrei abläuft. Diese Annahme lässt sich in realen Netzwerken jedoch üblicherweise nicht aufrechterhalten, weshalb Fernaufrufsysteme so ausgelegt werden sollten, dass sie auch beim Auftreten von Netzwerkfehlern korrekt funktionieren. Wie in der Vorlesung erläutert, lässt sich dies durch die Implementierung geeigneter Fernaufrufsemantiken erreichen.

Ziel von Übungsaufgabe 3 ist es, das bestehende Fernaufrufsystem so zu erweitern, dass es temporäre Kommunikationsfehler tolerieren kann. Hierzu sollen die beiden Semantiken *Last-of-Many* und *At-Most-Once* unterstützt werden, wobei hinsichtlich deren Umsetzung in den Teilaufgaben 3.2 und 3.3 auf Modularität zu achten ist. Die Implementierung sollte also gestaltet sein, dass theoretisch weitere Semantiken integriert werden könnten, ohne die dann bestehenden Klassen signifikant modifizieren zu müssen. Die Tolerierung von Rechnerausfällen wird im Rahmen dieser Aufgabe nicht betrachtet, da sie zusätzliche Mechanismen erfordert (z. B. zur persistenten Sicherung des Anwendungszustands).

3.1 Methodenspezifische Auswahl der Fernaufrufsemantik (optional für 5,0 ECTS)

Die unterschiedlichen Charakteristika verschiedener Fernaufrufsemantiken (z. B. die Anzahl möglicher Ausführungen) führen dazu, dass im Allgemeinen nicht dieselbe Semantik für alle Operationen eines Diensts anwendbar ist. Das eigene Fernaufrufsystem soll daher die Möglichkeit bieten, mittels einer Annotation [Folien 3.1 : 4–5] namens `VSRPCSemantic` für jede Methode einer Remote-Schnittstelle einzeln festzulegen, welche Semantik jeweils zum Einsatz kommen soll:

```
public @interface VSRPCSemantic {
    VSRPCSemanticType value();
}
```

`VSRPCSemanticType` ist dabei eine Enum, die zur Unterscheidung der beiden im Kontext dieser Übungsaufgabe relevanten Fernaufrufsemantiken zwei Werte umfassen soll: `LAST_OF_MANY` und `AT_MOST_ONCE`.

Aufgaben:

→ Bereitstellung der Annotation `VSRPCSemantic`
→ Annotierung der Methoden der Schnittstelle `VSAuctionService` des Auktionsdiensts

3.2 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 zwangsweise nur einmal, sondern unter Umständen (d. h. im Fehlerfall) mehrmals ausgeführt.
- Dem Client-Stub werden nach jeder Ausführung stets die neuesten Ergebnisse zurückgeliefert.

Diese Eigenschaften der Last-of-Many-Semantik werden dabei durch folgende Vorgehensweise erreicht:

- Erhält ein Stub nach dem Absenden einer Anfrage vom Skeleton innerhalb eines vordefinierten Zeitintervalls [Folie 3.1 : 7] keine Antwort, sendet er die Anfrage erneut.
- Der Skeleton führt nach Empfang einer Anfrage die betreffende Operation in jedem Fall lokal aus, ohne dabei auf Duplikate zu prüfen. Das auf diese Weise erhaltene Ergebnis sendet er anschließend an den Stub zurück.
- Ein Stub wartet so lange, bis eine Antwort auf die neuste von ihm geschickte Anfrage zurück kommt. Diese reicht er an den Aufrufer weiter. Alle anderen Antwortnachrichten werden verworfen.

Die Umsetzung der Last-of-Many-Semantik erfordert, dass der Stub in der Lage ist, Antworten auf seine verschiedenen Fernaufrufe eindeutig zu unterscheiden (→ Fernaufruf-ID). Darüber hinaus muss es dem Stub möglich sein, die zum selben Fernaufruf gehörigen Antwortnachrichten auseinander zu halten und festzustellen, ob es sich um die Antwort auf die zuletzt gesendet Anfrage handelt (→ Einsatz von Sequenznummern).

Aufgabe:

→ Implementierung der Last-of-Many-Semantik

Hinweis:

- 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 der Client-Anwendung durch eine geeignete `RemoteException` zu signalisieren.

3.3 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 die Server-Seite jede zu einer einzelnen Anfrage gehörende Operation nur höchstens einmal ausführt. Hierzu muss der Skeleton in der Lage sein, anhand von Fernaufruf-IDs folgende Situationen zu unterscheiden:

- *Neue Anfrage*: Der Skeleton erhält eine Anfrage mit einer ihm noch unbekannten Fernaufruf-ID.
  → Nach Ausführung der entsprechenden Operation sendet er das Ergebnis an den Client-Stub; zusätzlich speichert er das Ergebnis bei sich lokal ab.
- *Alte Anfrage*: Der Skeleton 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 Skeleton das bereits vorliegende, lokal gespeicherte Ergebnis an den Stub.

Da der Skeleton über begrenzten Speicherplatz verfügt, ist ein Garbage-Collection-Mechanismus für nicht mehr benötigte Ergebnisse zu realisieren, der den Speicher auf Server-Seite zuverlässig aufräumt. Die gewählte Strategie soll dabei nur Ergebnisse verwerfen, deren zugehörige Fernaufrufe bereits garantiert beendet sind.

Aufgabe:

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

Hinweis:

- Es darf angenommen werden, dass ein Client keinen Bedarf mehr für das Ergebnis eines Fernaufrufs hat, sobald er einen weiteren Fernaufruf durchführt.
- Im Fall eines Verbindungsabbruchs soll der Client-Stub die noch verbleibenden Anfragewiederholungen über eine neue Verbindung senden.

3.4 Testen der Implementierung (für alle)

Die Integration der in den vorherigen Teilaufgaben realisierten Semantiken in das bestehende Fernaufrufsystem soll abschließend unter Verwendung des Auktionsdiensts aus den vorherigen Übungsaufgaben getestet werden. Dabei sind folgende Kommunikationsfehler zu berücksichtigen:

- Verlust von Nachrichten aufgrund von Verbindungsabbrüchen
- Verzögerung von Nachrichten

Die aktuelle Implementierung des Kommunikationssystems kann derartige Fehler trotz des verwendeten TCP-Protokolls nicht vollständig tolerieren. Zum Testen sollen diese Kommunikationsfehler nun bewusst herbeigeführt werden. Zu diesem Zweck ist das Kommunikationssystem an geeigneten Stellen so zu sabotieren, dass es Nachrichten verzögert oder infolge eines Verbindungsabbruchs verliert. 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 [Folie 3.1 : 6].

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.
- Ein Verbindungsabbruch lässt sich durch Aufruf von `close()` auf einem `Socket` nachbilden.

Abgabe: am Mo., 01.06.2026 bzw. Mi., 03.06.2026 in der Rechnerübung

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