Übungsaufgabe #3: Object Request Broker

18.11.2003

Ziel dieser Aufgabe ist es, einen Object Request Broker (ORB) zu implementieren. Die Architektur und Funktionsweise eines ORB's wurde dafür bereits in der Übung besprochen.

Um den Implementierungsaufwand etwas zu reduzieren, wird ein Stub/Skeleton-Generator zur Verfügung gestellt. Dieser ist unter `/proj/i4mw/pub/aufgabe3/stubgen` zu finden. Zu einer gegeben Klasse können die benötigten Hilfsklassen durch den folgenden Aufruf des Stub/Skeleton-Generator erzeugt werden:

```
java stubgen.StubGen <class_name>
```

Im Rahmen der Übung haben Sie das Werkzeug *ant* kennengelernt. Schreiben sie ein Buildfile welches alle nötigen Teilschritte für die Übersetzung der Aufgabe durchführt und geben sie dieses zusammen mit der Lösung ab.

a) Bevor mit der Implementierung begonnen wird, soll zunächst (wie in der letzten Aufgabe) ein Design der Anwendung erstellt werden, um die anschließende Implementierung zu erleichtern. Es sollten die geplanten Klassen, Interfaces und Aufgaben festgelegt werden, benötigte Threads und die Synchronisation gemeinsamer Datenstrukturen.
Lesen Sie sich zuerst die folgenden Teilaufgaben durch und erstellen sie dann Ihre Lösung. Eine einfache ASCII-Textdatei reicht, bei umfangreicheren Grafiken kann auch ein PDF-Dokument erstellt werden.

b) Implementieren sie die Klasse `ObjectStore`. Dieser soll mindestens die in der Datei `/proj/i4mw/pub/aufgabe3/orb/ObjectStore.java` definierten Methoden testen. Die Datei kann als Ausgangspunkt für die Entwicklung der Klasse `ObjectStore` genutzt werden.

c) Ziel dieser Teilaufgabe ist die Verarbeitung von Anfragen durch den ORB. Die Zustellung von Antworten wird in Teilaufgabe d) behandelt. Erstellen sie die Klassen `Broker` und `Request`.
Die Klasse `Broker` soll eine Methode
`public int exportObject(Object obj) throws AlreadyRegisteredException`
zur Verfügung stellen, die das Registrieren von Objekten als entfernte Objekte ermöglicht (Nutzen sie hierfür die Klasse `ObjectStore`). Als Rückgabewert liefert die Methode die Objekt-ID. Im Konstruktor des Brokers soll ein neuer Thread erzeugt werden, der ankommende Anfragen entgegen nimmt. Diese werden als Objekte vom Typ `DatagramPacket` zugestellt. Benutzen sie ihre Benutzerkennung (UID) als Portnummer für den `DatagramSocket`.
Anfragen werden durch Objekte vom Typ `Request` repräsentiert und sollten über die folgenden Methoden verfügen:
`void process() throws Exception`
wird vom Server in einem eigenen Thread aufgerufen und verarbeitet die Anfrage.
`Object send() throws Exception`
wird vom Client aufgerufen um Anfragen an den Server zuzustellen und die zugehörigen Antworten entgegenzunehmen, dabei blockiert der Client bis eine Antwort eintrifft.

Ein Objekt vom Typ `Request` verfügt mindestens über die Felder Objekt-ID, Methoden-ID sowie die Parameter der Methode in Form eines Arrays. Diese werden im Konstruktor der Klasse initialisiert.

```
public Request(ClientContext cctx, int oid, int mid,
               Object[] parameters)
```

Eine Instanz der Klasse `ClientContext` enthält die Informationen, welche für den Aufruf der Client-Methoden, wie zum Beispiel `send()`, benötigt werden.

Sollten auf Seite des Servers zusätzliche Informationen für die Methode `process()` benötigt werden, kann dies ähnlich realisiert werden. Es ist jedoch zu beachten, dass ein Objekt vom Typ `ServerContext` nicht im Konstruktor der Klasse `Request` übergeben werden kann, da diese Objekte durch den Client erzeugt werden.
Im Rahmen dieser und der nächten Teilaufgabe kann man davon ausgehen, dass die Pakete in FIFO-Reihenfolge zugestellt werden. Um diese Teilaufgaben zu testen werden die Klassen `test.Server` und `test.Client` angeboten (`/proj/i4mw/pub/aufgabe3/test`).

d) Nun soll die Klasse `Receiver` implementiert werden. Im Konstruktor dieser Klasse soll ein neuer Thread erzeugt werden, der auf Objekte vom Typ `Reply` und `ExceptionReply` wartet. Als Portnummer soll hierbei die UID + 10000 verwendet werden.

Ein Objekt vom Typ `Reply` wird zugestellt wenn eine Methode erfolgreich aufgerufen werden konnte. Sollte dies nicht der Fall sein wird ein Objekt vom Typ `ExceptionReply` zurückgeliefert. Dieses Objekt kapselt die auf der Server-Seite aufgetretene Exception, so dass sie an den eigentlichen Aufrufer zugestellt werden kann.

e) Ziel der letzten Teilaufgabe ist es eine *Exactly-Once*-Semantik zu implementieren. Das Programm sollte mit allen auftretenden Fehlern wie Verlust von Paketen, duplizierten Paketen und Zustellung außerhalb der Reihenfolge zurechtkommen. Verwenden Sie zum Testen statt `java.io.DatagramSocket` die Klasse `comm.CrazyDatagramSocket` (`/proj/i4mw/pub/aufgabe3/comm/CrazyDatagramSocket`). Das Verhalten der Kommunkationsschicht kann über die Methoden `setSendLossRate()`, `setSendDuplicationRate()` und `setSendOutOfOrderRate()` der Klasse `CrazyDatagramSocket` verändert werden.

<u>Bearbeitung:</u> bis zum 11.12.2003/18:00 Uhr
Alle Dateien sollen im Verzeichnis `/proj/i4mw/`*`loginname`*`/aufgabe3/` abgelegt und mit dem `abgabe`-Programm abgegeben werden.
Die Bearbeitung ist in 2er Gruppen möglich.