UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Aufgabenblatt 5
Musterlösung

Vorlesung Verteilte Systeme
Sommersemester 2020

Aufgabe 1: Programmierung: Factories

Die Lösung dieser Aufgabe finden Sie im Archiv `l05Files.zip`[1] auf der Vorlesungswebseite. Die Methode `getAllAccounts()` liefert eine serialisierte Liste mit allen Accounts zurück, der eigentliche Inhalt der Liste sind jedoch Remote-Referenzen.

Aufgabe 2: Threadsicherheit

Die Threadsicherheit einer RMI-Anwendung kann durch das Schlüsselwort synchronized bei den relevanten Methodendeklarationen in der Serverklasse erreicht werden:

Server-Klasse

```
public synchronized void add(int i) {
   number = number + i;
}

public synchronized int value() {
   return(number);
}
```

Clientseitig kann ein Remote-Objekt nicht gesperrt werden (ausser es werden speziell dafür eigene Remote-Mothoden implementiert). Wenn der Client

```
Number num = (Number)Naming.lookup(...);
synchronized(num) {
   num.add(10);
   System.out.println(num.value());
}
```

ausführen würde, würde lediglich das lokale Stub-Objekt gesperrt, was keinen wechselseitigen Ausschluss mit den anderen Clients bewirkt.

Aufgabe 3: Programmierung: Client-Callback - Fortschrittsbalken

Die Lösung dieser Aufgabe finden Sie im Archiv `l05Files.zip`[2] auf der Vorlesungswebseite. Das erwähnte Problem beim Erben von der Klasse `UnicastRemoteObject` ist, daß sich der Client nicht mehr beendet. Der Grund ist, daß zu einem Remote-Objekt ein Thread erzeugt wird, der eingehende Remote-Aufrufe bearbeitet. Ein Java-Programm terminiert aber erst, wenn alle Threads beendet sind.

[1] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l05Files.zip`
[2] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l05Files.zip`

Aufgabe 4: Namensdienste

a) IP-Adressen im Internet werden für die Adressierung von Hosts verwendet. Um auf einen Host zuzugreifen, muss eine IP-Adresse beispielsweise in eine Ethernet-Adresse aufgelöst werden.

b) Beispiele sind ISBN-Nummern für Bücher, ID-Nummern für Software und Hardware-Produkte, Personalnummern innerhalb eines Unternehmens und Ethernet-Adressen (obwohl einige Adressen genutzt werden, um eine ganze Maschine zu identifizieren und nicht nur die Ethernetkarte).

c) Beide Namen können positionsunabhängig sein, aber die erste gibt weniger Hinweise auf die Position der angegeben Einheit. Positionsunabhängigkeit bedeutet in diesem Fall, dass der Name der Einheit unabhängig von ihrer Adresse ist. Betrachtet man nur einen Namen, kann nichts über die Adresse der zugehörigen Einheit gesagt werden.

d) Zunächst, wenn Aliase nur privat und nicht öffentlich definiert sind, besteht das Risiko, dass ein User fälschlicherweise jedoch unbemerkt ein bestimmtes Objekt referenziert, welches den Alias verwendet. Der Alias könnte ein (falsches) Objekt im Namensraum eines anderen Users referenzieren. Des Weiteren können Aliase Zyklen innerhalb eines Graphens einbringen. Beispielsweise könnte ein Name wie `/users/fred` prinzipiell zu einem Alias von `/users` gemacht werden. Ein Resolver könnte hier potentiell in einer Art Endlosschleife enden, während er versucht, den Namen aufzulösen. Eine Lösung für dieses Problem kann das Setzen eines Limits für die Anzahl an Aliasen sein, die ein Resolver bei dem Versuch den Namen aufzulösen, testen kann.

Aufgabe 5: Namensdienste

Namensdienst: Zuordnung von Namen zu Objekten, um Dienste finden zu können. Physische Adressen sind ungeeignet, weil sie sich ändern können.

Binden: Zuordnung eines Namens zu einem Objekt, einer Ressource, oder einer physischen Adresse;

Beispiele: DNS (Rechnernamen, Mailexchanger), NIS (User + Namen + LoginID)

Hierarchische Namensdienste: Aufteilung des Namensraums in Zonen, pro Zone existiert ein Server, der die Unterserver für seine Unterzonen bzw. ganz unten die Einträge selbst kennt.

Iterative Namensauflösung: Client fragt die (Unter-)Server selbst nacheinander ab.

Rekursive Namensauflösung: Server fragt Unterserver, dieser seinen Unterserver, u.s.w.

Aufgabe 6: Programmierung: JNDI

a) JNDI (Java Naming and Directory Interface) stellt Methoden mit Standard-Verzeichnisoperationen bereit (lesen, schreiben, löschen, hinzufügen von Attributen zu Objekten und die Suche nach Objekten auf Basis dieser Attribute). Darüber hinaus definiert es eine einheitliche Schnittstelle für den Zugriff auf verschiedene Verzeichnisdienste.

Java Applikationen bieten mithilfe von JNDI zwei Möglichkeiten mit Namens und Verzeichnisdiensten zu Arbeiten. Einmal auf traditionellem Wege - traditionell bedeutet in diesem Zusammenhang auf tatsächlich existierende Verzeichnisdienste (Dateisystem, Mail, etc.) zuzugreifen. Die zweite Möglichkeit besteht darin, das Directory als einen Speicherort und eine Organisationsform für die Ablage von Objekten zu verwenden.

b) Die Lösung dieser Aufgabe finden Sie im Archiv `l05Files.zip`[3] auf der Vorlesungswebseite.

c) Die Lösung dieser Aufgabe finden Sie im Archiv `l05Files.zip`[4] auf der Vorlesungswebseite.

[3] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l05Files.zip`
[4] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l05Files.zip`