

Übungsblatt 2

zur Vorlesung

Verteilte Systeme/Ubiquitous Computing

Wintersemester 2007/2008

Achtung: Bitte beachten Sie auch die Web-Site der Vorlesung:
<http://www.mobile.ifi.lmu.de/Vorlesungen/ws0708/vs/>

Aufgabe 4: (H) RPC

Das C/S-Modell basiert auf dem Paradigma der Ein- und Ausgabe von Daten. Verteilte Berechnungen sind in diesem Modell nur schwer so zu verwirklichen, dass der Benutzer die Verteiltheit nicht bemerkt, sondern an eine lokale Ausführung seiner Befehle glaubt. Um die Verteilung von Aufgaben für den Benutzer transparent zu machen, ersannen Birell und Nelson den Remote Procedure Call (RPC).

- a. Bitte erläutern Sie den Ablauf eines RPCs und gehen Sie dabei auf das Konzept des kritischen Pfads (critical path) ein.
- b. Remote Method Invocation (RMI) kann als die objekt-orientierte Variante des RPCs aufgefasst werden. Java RMI ist eine Implementierung der RMI. Benutzen Sie das Java.RMI-Paket, um sämtliche Objekte zu implementieren, die es einem Klienten ermöglichen, eine entfernte Methode zur Ermittlung der Zeit aufzurufen. Eine Anleitung, wie RMI verwendet wird, finden Sie unter <http://java.sun.com/j2se/1.3/docs/guide/rmi/getstart.doc.html>.
 - (i) Beginnen Sie zunächst damit, das Interface zu schreiben, das der Stub des Klienten benötigt, um das entfernte Objekt zu referenzieren. Dieses muss die Signatur der aufzurufenden Methode beinhalten.
 - (ii) Implementieren Sie anschließend den Server. Dieser erbt vom Standard-Remote-Object "UnicastRemoteObject" und verwendet das zuvor geschriebene Interface!
 - (iii) Kompilieren Sie den Server und das Interface. Erzeugen Sie Stub und Skeleton durch die Verwendung des Befehls "rmic"!
 - (iv) Implementieren und kompilieren Sie den Client! Die Class-Datei des Interfaces und der Stub müssen auf dieselbe Maschine kopiert werden, auf der der Client residieren soll.
 - (v) Starten Sie auf der Serverseite den Remote Registry Server. Unter Windows geschieht dies durch den Befehl „start rmiregistry“. Melden Sie das entfernte Objekt dort an. Schenken Sie dabei der Sicherheitspolitik des RMI-Paketes besondere Aufmerksamkeit. Näheres hierzu erfahren Sie unter <http://java.sun.com/j2se/1.3/docs/guide/security>. Falls Sie nicht mehr weiter kommen, kopieren Sie einfach die erhaltene Fehlermeldung als Ganzes ins Suchfenster von www.google.com und suchen Sie nach einer der vielen Seiten, die das Problem behandeln.

- (vi) Starten Sie anschließend den Klienten und vergewissern Sie sich, dass Ihr Programm korrekt operiert!

Antwort:

- a. Hinter dem Vorschlag von Birell und Nelson von 1984 steht die Idee der Transparenz: Ein Prozess soll ein Unterprogramm auf einem anderen Rechner aufrufen können.

Ablauf:

- (i) Der Client ruft ein Unterprogramm im Client-Stub auf.
- (ii) Der Client-Stub erzeugt eine Nachricht und übergibt sie an den Kern.
- (iii) Der Kern sendet die Nachricht an den entfernten Kern.
- (iv) Der entfernte Kern übergibt die empfangene Nachricht dem Server-Stub.
- (v) Der Server-Stub packt die Parameter aus und ruft ein Unterprogramm im Server auf.
- (vi) Der Server führt das Unterprogramm aus und übergibt dem Server-Stub die Ergebnisse.
- (vii) Der Server-Stub verpackt die Ergebnisse in einer Nachricht und übergibt sie seinem Kern.
- (viii) Der entfernte Kern sendet die Nachricht an den Kern des Client.
- (ix) Der Client-Kern übergibt die Nachricht an den Client-Stub.
- (x) Der Client-Stub packt die Ergebnisse aus und übergibt sie dem Client.

Der *kritische Pfad* ist die Folge von Instruktionen, die bei jedem RPC ausgeführt wird. Er beginnt mit dem Aufruf des Client-Stubs durch den Client, geht dann über zum Kern, beschreibt die Nachrichtenübertragung, die Server-Seite, die Ausführungen des Unterprogramms und die Rückantwort.

Die Frage, die sich bei der Implementierung des RPC's stellt, ist nun, welcher Teil des kritischen Pfades der aufwendigste ist, also am meisten Rechenzeit in Anspruch nimmt. Die Schwachstellen müssen analysiert und dann die Ausführung optimiert werden.

- b. (i) Entfernte Objekte werden in java.rmi über Interfaces referenziert. Deswegen muss für unser entferntes Objekt "MeineZeitansage" zunächst ein Interface erzeugt werden:

```
package server;

import java.rmi.*;

public interface Zeitansage extends Remote
{
    String getTime() throws RemoteException;
}
```

- (ii) Danach kann das entfernte Objekt implementiert werden:

```
package server;

import java.rmi.*;
import java.rmi.server.*;
import java.text.DateFormat;

public class MeineZeitansage extends UnicastRemoteObject implements
    Zeitansage
{
    static String hostName="192.168.218.44";
    public MeineZeitansage() throws RemoteException
    {
        super();
        //Der Konstruktor des Default RemoteObjects (=UnicastRemoteObject) wird verwendet.
    }
}
```

```

16     public String getTime() throws RemoteException
17     {
18         System.out.println("Eine_Zeitanfrage_erfolgt.");
19         DateFormat myFormatter = DateFormat.getDateInstance();
20         String myTime = myFormatter.format(new java.util.Date());
21         return myTime;
22     }
23
24     public static void main(String args[])
25     {
26         //If security manager used, security policy must be set.
27         //Otherwise not.
28         System.setSecurityManager(new RMISecurityManager());
29         //Der Default SecurityManager des RMI-Pakets wird verwendet.
30         try
31         {
32             MeineZeitansage instance = new MeineZeitansage();
33             Naming.rebind("//"+hostname+"/Zeitansage", instance);
34             System.out.println("Das_Objekt_MeineZeitansage_ist_jetzt_registriert.");
35         }
36         catch (Exception ex)
37         {
38             System.out.println(ex);
39         }
40     }
41 }
42

```

- (iii) Beide Dateien sind mit *javac* zu kompilieren.

Der Stub und der Skeleton müssen nicht programmiert werden. Sie entstehen durch die Verwendung des "Remote Method Invocation Compilers". Der entsprechende Befehl für die Erzeugung des Stubs und des Skeletons ist *rmic MeineZeitansage*.

- (iv) Der Code für den Klienten lautet wie folgt:

```

package client;

import server.*;
import java.rmi.*;

public class MeinKlient
{
    static String hostname = "192.168.218.44";
    public static void main (String args[])
    {
        try
        {
            Zeitansage zeitansage = (Zeitansage)
            Naming.lookup("//"+hostname+"/Zeitansage");
            System.out.println(zeitansage.getTime());
        }
        catch (Exception ex)
        {
            System.out.println(ex);
        }
    }
}

```

Das Interface „Zeitansage.class“ und der durch den rmic-Befehl erzeugte Stub, MeineZeitansage_Stub.class, müssen auf die Maschine kopiert werden, auf der der Client residieren soll.

- (v) Der Remote Registry Server ist auf der Serverseite zu starten. Unter Windows geschieht dies durch den Befehl „start rmiregistry“.

Anschließend muss ein Objekt der Klasse „MeineZeitansage“ durch

`java server.MeineZeitansage`

erzeugt werden. Dadurch wird die Main-Methode aufgerufen, die eine Registrierung beim Remote Registry Server veranlasst.

Hierbei wird jedoch folgende Fehlermeldung ausgegeben:

`java.security.AccessControlException: access denied (java.net.SocketPermission 192.168.218.44:1099 connect,resolve)`

Ursache hierfür ist die Sicherheitspolitik des Remote Registry Servers. Die Instanz unserer Klasse „MeineZeitansage“ besitzt bisher nicht das Recht, einen Socket zu öffnen. Diese Sicherheitspolitik kann wie folgt geändert werden:

`java -Djava.security.policy=[vollständiger Pfad der Datei, die die neue Politik spezifiziert] server.MeineZeitansage`

Die Datei mit der neuen Sicherheitspolitik könnte bspw. folgenden Inhalt besitzen:

`grant { permission java.net.SocketPermission "192.168.218.44", "connect,resolve"; }`

Durch diese Zeile erhält das MeineZeitansage-Objekt das Recht, eine Verbindung zum Remote Registry Server auf- und abzubauen.

Für nähere Informationen sei auf die Seite:

<http://java.sun.com/j2se/1.3/docs/guide/security/permissions.html#SocketPermission> verwiesen.

- (vi) Jetzt kann der Client nach dem Kompilieren ordnungsgemäß gestartet werden (`java client.MeinKlient`).

Aufgabe 5: (H) Das World Wide Web als großes Verteiltes System

- Was ist HTML?
- Wie ist ein URL aufgebaut?
- Welche Arten aktiver Inhalte kennen Sie?

Antwort:

- a. **HTML - HyperTextMarkupLanguage:**

- Instanz von SGML, der Standard Generalized Markup Language
- spezifiziert den Inhalt und das Layout von Webpages
- definiert Links und die damit verbundenen Ressourcen
- in einer Datei gespeichert, auf die ein Webserver zugreifen kann
- nur der Browser, nicht der Webserver, interpretiert den HTMLText

- b. **URL - Uniform Resource Locator:**

Struktur: `<schema>://<hostname>[:port]/[path/filename[#section]]/?arguments`

Inhalte zwischen zwei eckigen Klammern sind optional.

Gültige Protokolle sind z.B.: file, http, https, ftp, mailto, news, telnet, WAIS

Eine URL kann also bspw. folgendes Aussehen besitzen:

`ftp://ftp.downloads.com/software/das_gewünschte_Programm.exe`

oder

`http://Servername_laut_DNS_oder_IP-Adresse:80/Zielpfad/`

Somit ergeben sich drei Möglichkeiten, über http auf Ressourcen im Internet zuzugreifen:

- Zunächst einmal kann nur der Servername eingegeben werden, bspw. <http://www.mobile.ifi.lmu.de/>. Hierbei wird die Standard-Internetseite des angesprochenen Servers zurückgegeben.
- Ist die gewünschte Seite genauer bekannt, so kann zusätzlich der komplette Pfad angegeben werden, bspw. <http://www.mobile.ifi.lmu.de/Vorlesungen/ss06/vs.shtml>.
- Gerade bei Suchmaschinen wird häufig die Suchanfrage als Argument angehängt. Beispiel: <http://www.google.de/search?q=Linnhoff-Popien>

c. **Arten aktiver Inhalte:**

- CGI-Programme
- Javascript
- Applet
- Servlet
- ActiveX-Inhalte (z.B. Flash-Programme)

Aufgabe 6: (H) Gruppenkommunikation

- a. Wodurch unterscheidet sich die Gruppen- von der Zwei-Parteien-Kommunikation?
- b. Was ist der Unterschied zwischen Unicast, Multicast und Broadcast?
- c. IP-Multicast ist eine bekannte Implementierung der Gruppenkommunikation. Benutzen Sie das java.net-Paket und implementieren Sie das Szenario der Aufgabe 4, das wie folgt erweitert wird:

Mehrere Zeit-Server, die auf verschiedenen Maschinen installiert sind, bilden eine Gruppe. Ein Klient wendet sich an jene und erfragt die Zeit. Aufgrund der Antworten der Gruppenmitglieder können etwaige Unterschiede zwischen den lokalen Zeiten identifiziert werden.

- Beginnen Sie damit eine Klasse für die Zeit-Server zu implementieren. Schreiben Sie den Code so, dass Sie mehrere Instanzen dieser Klasse unterscheidbar erzeugen können. Der Server öffnet zunächst eine neue Socket und meldet sich bei der Gruppe an. Dies geschieht über eine IP-Multicast-Adresse, die zwischen 224.0.0.1 und 239.255.255.255 gewählt werden kann. Anschließend wartet der Server auf eine Anfrage und antwortet bei Verlangen mit der lokalen Zeit.
- Schreiben Sie im zweiten Schritt den Klienten. Dieser muss ebenfalls eine Socket öffnen und der Gruppe beitreten. Eine Anfrage wird an die Gruppe versendet. Die eingehenden Antworten werden einfach auf den Bildschirm ausgegeben.
- Erzeugen Sie mehrere Instanzen des Zeit-Servers auf unterschiedlichen Computern und starten Sie Ihren Klienten. Wie Sie sehen, ist die Synchronisation der Uhren in einem verteilten System ein reales Problem.

Antwort:

- a. In der Gruppenkommunikation kommuniziert einer mit vielen (one-to-many). Die Zusammensetzung der Empfänger kann sich während einer Sitzung ändern, da Gruppen inhärent dynamisch sind. Gesendet wird an die Gruppe, nicht an die einzelnen Gruppenmitglieder.

- b. **Unicast:** Es werden genau so viele Ende-zu-Ende-Übertragungen ausgeführt, wie es Empfänger gibt.
Multicast: Es wird ein Multicast-Tree aufgebaut. Der Sender ist die Wurzel des Baumes. Er sendet nur an Knoten der ersten hierarchischen Ebene. Diese geben die Nachricht an ihre Kinder weiter und duplizieren hierfür die Nachricht, falls nötig. Die Empfängerknoten wiederum propagieren die Nachrichten an ihre Kinder und so weiter, bis alle Knoten im Baum erreicht wurden.
Broadcast: Der Sender sendet eine Nachricht an alle direkten Nachbarn. Die wiederum senden an ihre direkten Nachbarn. Jeder Knoten, der eine Nachricht erhält, die er schon einmal versendet hat, verwirft diese einfach.

- c. (i) Der Quelltext des Servers:

```
//TimeServer
2 //Command: java TimeServer multicast_addr srv_name

4 import java.net.*;
  import java.io.*;
6 import java.text.DateFormat;
  import java.util.Date;
8
  public class TimeServer{
10 public static void main(String args[]){
    try{
12      InetAddress group=InetAddress.getByName(args[0]);
        MulticastSocket s = new MulticastSocket(4321);
14      s.joinGroup(group);
        String id = args[1];
16
        while(true){
18          //Receive something
            byte[] buf = new byte[1000];
20          DatagramPacket msg = new DatagramPacket(buf, buf.length);
            s.receive(msg);
22          String inmsg = new String(msg.getData(), 0, msg.getLength());
            System.out.println("TimeServer_" + id + "_received:_" + inmsg);
24
            //If a request, then reply
26          if(inmsg.equals("What_time")==true){
              DateFormat df = DateFormat.getDateTimeInstance
28                (DateFormat.FULL, DateFormat.FULL);
              String rpl = id + "_sends:_" + df.format(new Date());
30              System.out.println("TimeServer_" + rpl);
              byte[] om = rpl.getBytes();
32              DatagramPacket outmsg = new DatagramPacket(om, om.length, group, 4321);
              s.send(outmsg);
34          }
        }
36      //s.leaveGroup(group);
    }catch(SocketException e){
38      System.out.println("Socket:" + e.getMessage());
    }catch(IOException e){System.out.println("IO:" + e.getMessage());}
40 }
}
```

- (ii) Der Quelltext des Klienten:

```
//Client
2 //Command: java Client multicast_addr

4 import java.net.*;
```

```
import java.io.*;
6 import java.text.DateFormat;
import java.util.*;

8
public class Client{
10 public static void main(String args[]){
    try{
12         InetAddress group=InetAddress.getByName(args[0]);
        MulticastSocket s = new MulticastSocket(4321);
14         s.joinGroup(group);
        s.setSoTimeout(3000); //set timeout to 3sec

16         //Send request "What time"
        String req = "What_time";
18         byte[] om = req.getBytes();
        DatagramPacket outmsg = new DatagramPacket(om, om.length, group, 4321);
        s.send(outmsg);

22         while(true){
24             //Receive replies
            byte[] buf = new byte[1000];
26             DatagramPacket msg = new DatagramPacket(buf, buf.length);
            try{
28                 s.receive(msg);
            }catch(InterruptedException e){
30                 s.leaveGroup(group);
                return;
32             }
            System.out.println("Client_received:_ " + new String(msg.getData()));
34         }
    }catch(SocketException e){
36         System.out.println("Socket:_ " + e.getMessage());
    }catch(IOException e){System.out.println("IO:_ " + e.getMessage());}
38 }
}
```