5A Kooperation: Lösungen

5A.1 Wissens- und Verständnisfragen

1.a.) dass nicht auf jedem Computer alle Programme und Daten vorgehalten werden müssen / dass damit auch Geräte auf anderen Rechnerknoten angesprochen werden können

1.b.) alle drei Antworten richtig

1.c.) Polling, einen Interrupt

1.d.) RMI, RPC

1.e.) Eine Sammlung von Diensten in verteilten Systemen, die von darüberliegenden Anwendungen genutzt werden können

1.f.) Remote Procedure Call

1.g.) alle drei Antworten richtig

1.h.) Servlets

2.a.) Server, Client

2.b.) Polling

2.c.) Marshalling

2.d.) Remote Method Invocation

2.e.) rpcgen

2.f.) Registry

2.g.) Web Services

3.a.) Wahr, denn der Client fragt wiederholt aktiv beim Server nach, ob die Antwort vorliegt.

3.b.) Wahr, beispielsweise durch den Verlust von Daten im Kommunikationsnetz.

3.c.) Falsch, denn ONC RPC arbeitet prozedurorientiert, nicht objektorientiert.

3.d.) Falsch: Der Applet-Bytecode wird unmittelbar vom Server heruntergeladen und im Browser des Clients ausgeführt. Eine Registry spielt hier keine Rolle.

3.e.) Wahr, denn ihre Methoden reagieren auf HTTP-Befehle.

4.a.) Registry, denn sie ist kein Kooperationsmodell.

4.b.) TCP, denn es hat nichts mit der Definition von Schnittstellen zu tun.

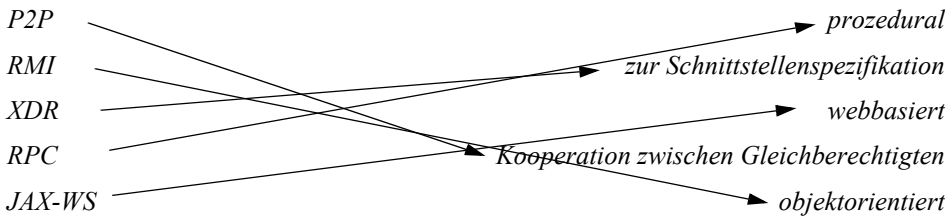
4.c.) RPC, denn es ist kein Begriff aus der objektorientierten Kooperation, sondern aus der prozedurorientierten.

4.d.) Java Server Pages, denn es steht in keinem Zusammenhang zu Java RMI.

4.e.) Applet, denn es wird nicht beim Server, sondern beim Client ausgeführt.

4.f.) IP, denn es ist kein Kooperationsmechanismus, sondern ein Netzwerkprotokoll.

5.)Abkürzungen und Eigenschaften:



6.a.) synchron = Client wartet passiv auf Antwort des Servers; asynchron = Client tut während der Arbeit des Servers etwas anderes.
asynchrone Kooperation mit Abfragen des Servers durch den Client ("Polling") oder mit Unterbrechung des Clients durch den Server ("Interrupt").

6.b.) Das Three Tier Model beschreibt ein System aus einem Client und zwei Servern, bei dem der eine Server (Unter-)Aufträge an den zweiten Server weiterreicht.

6.c.) Ein Stub ist eine Funktion auf Client-Seite, die dieselbe Schnittstelle wie die Server-Funktion hat und den Funktionsaufruf an die Funktion des Servers weiterleitet.

6.d.) Beide führen den Client-Auftrag nicht selbst aus, sondern leiten ihn an den Server weiter. Der RPC-Stub stellt eine prozedurorientierte Schnittstelle bereit, der RMI-Proxy eine objektorientierte.

6.e.) Sie ist nicht vorhanden: Der Programmierer und/oder Nutzer eines Programms mit diesen Funktionen muss die Adresse des Servers angeben.

6.f.) Es spezifiziert die Schnittstelle eines Objekts, auf das man über RMI zugreifen kann.

6.g.) Applets werden auf Seiten des Clients ausgeführt, Servlets auf Seiten des Servers.

5A.2 Sprachunabhängige Anwendungsaufgaben

1.)Chat = Peer-to-Peer, zentrale Auskunft = Client-Server

Eine synchrone Kooperation ist nur sinnvoll, wenn eine Anfrage schnell beantwortet wird, da sonst die (passiv verbrachte) Wartezeit zu lang würde.

Besser mit Antworten per Interrupt, da ein Polling sowohl auf Client- als auch auf Server-Seite zusätzlichen Bearbeitungsaufwand verursacht.

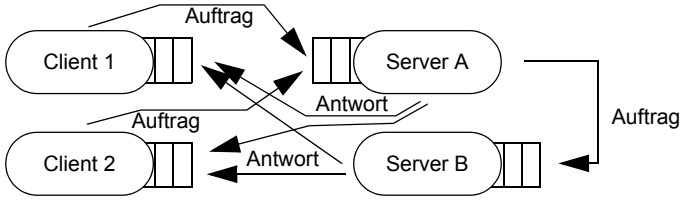
2.)Three Tier Model. Thin Client.

Ein Ingenieur (Benutzer des Systems) übergibt dem „Kommunikations-Mitarbeiter“ (Benutzerschnittstelle in der Three-Tier-Architektur, realisiert auf der Client-Seite) eine Anfrage. Der Mitarbeiter gibt sie an den Anwalt (Verarbeitungslogik, realisiert durch einen ersten Server) weiter. Dieser analysiert die Anfrage mit ihren Dokumenten und fragt bei kritisch erscheinenden Einzelpunkten seinen Archivar (Datenbasis, realisiert durch einen zweiten Server) nach dazu passenden Patentschriften. Auf der Basis dieser Schriften fer-

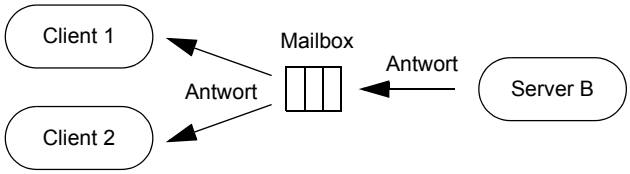
tigt der Anwalt ein Gutachten an und schickt es an den Kommunikations-Mitarbeiter in der Firma, der es dann dem Ingenieur präsentiert.

3.)Fahrkartenautomaten sind, wie Proxies, Objekte, die an einer Schnittstelle Zugriffsmethoden anbieten. Sie leiten, wie Proxies, Anfragen (z.B. zu Fahrplandaten) unmittelbar an den zentralen Server weiter. Allerdings geht ihre Funktionalität über die Funktionalität reiner Proxies, die nur die Weiterleitung von Aufträgen und Antworten realisieren, hinaus – zum Beispiel drucken die Automaten Fahrkarten aus und regeln die Ein- und Ausgabe von Bargeld.

4.)Client-Server-System mit Ports:



Client-Server-System mit Rückgabe-Mailbox:



Hier muss eine Empfängerangabe zurückgegeben werden, damit ein Client gezielt die Antworten abholen kann, die sich an ihn richten.

5.)Entscheidungskriterien: Auslastung der einzelnen Server / gleichmäßige Verteilung der Last.

Skizze ähnlich wie in Aufgabe 4.

Die beiden Eingangports der Server 1 und 2 können durch eine Mailbox ersetzt werden, aus der sich dann beide Server ihre Aufträge holen.

Eine Absenderangabe, damit die Antwort korrekt zugestellt werden kann.

5A.3 Programmierung unter UNIX/Linux

1.a.) Remote Procedure Call (RPC)

Der Code definiert die Schnittstelle eines Diensts, den ein Server anbietet.

maximalwert.x

rpcgen.

1.b.) #include "maximalwert.h"
#include <rpc/rpc.h>
float maximalwert_1_svc(drei_floats *triple, struct svc_req *r) {*

```
 static float result;
 result = triple->f1;
 if (triple->f2>result)
   result = triple->f2;
 if (triple->f3>result)
   result = triple->f3;
 return (&result); }
```

1.c.) clnt_create(), maximalwert_1(). Der erste Parameter von clnt_create ist der Name des Server-Rechners, also muss der Client wissen, wo genau der Server läuft.

```
2.)#include "mittel.h"
#include <rpc/rpc.h>
main(int argc; char *arg[]) {
 CLIENT *cl;
 char *server;
 drei_ints parameter;
 float *ergebnis;
 server=argv[1];
 cl=clnt_create(server,MITTEL_PROG,MITTEL_VERS,"udp");
 parameter.a1=2; parameter.a2=3; parameter.a3=4;
 ergebnis=mittel_1(&parameter,cl);
}
```

mittel.h, mittel_clnt.c, mittel_svc.c, mittel_xdr.c

```
3.)struct intpair {
 int a;
 int b; };
struct floatquadruple {
 float a;
 float b;
 float c;
 float d; };
program ARITH_PROG {
  version ARITH_VERS_1 {
    int GGT(intpair) = 1;
    int KGV(intpair) = 2;
    int MIN(floatquadruple) = 3;
   } = 1;
  version ARITH_VERS_2 {
    int GGT(intpair) = 1;
    int KGV(intpair) = 2;
    int MIN(floatquadruple) = 3;
   } = 1;
 } = 0x20000001;
```

5A.4 Programmierung in Java

1.)„class“ muss durch „interface“ ersetzt werden. Die Methodenköpfe müssen durch „throws RemoteException“ ergänzt werden.

2.)Datei MinMaxImpl.java:

```
import java.net.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class MinMaxImpl extends UnicastRemoteObject implements MinMaxInterface {
 public MinMaxImpl() throws RemoteException {}
 public int minimum(int x1, int x2, int x3) throws RemoteException {
   int min = x1;
   if (min>x2) min = x2;
   if (min>x3) min = x3;
   return min;
 }
 public int maximum(int x1, int x2, int x3) throws RemoteException {
   int max = x1;
   if (max<x2) max = x2;
   if (max<x3) max = x3;
   return max;
 }
}
```

Datei MinMaxServer.java:

```
import java.rmi.*;
import java.rmi.server.*;
public class MinMaxServer {
  public static void main(String args[]) {
    try {
      Naming.rebind("MinMax", new MinMaxImpl());
    } catch (Exception e) {
      System.out.println("MinMax-Server Error:"+ e.getMessage()); }
  }
}
```

3.)Datei MittelInterface.java:

```
import java.rmi.*;
public interface MittelInterface extends Remote {
  public double mittel(double x1, double x2, double x3) throws RemoteException;
}
```

Datei MittelImpl.java:

```
import java.net.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
```

```
public class MittelImpl extends UnicastRemoteObject implements MittelInterface {
 public MittelImpl() throws RemoteException {}
 public double mittel(double x1, double x2, double x3) throws RemoteException {
   return (x1+x2+x3)/3.0;
 }
}
```

Datei MittelServer.java:

```
import java.rmi.*;
import java.rmi.server.*;
public class MittelServer {
  public static void main(String args[]) {
    try {
      Naming.rebind("Mittel", new MittelImpl());
    } catch (Exception e) {
      System.out.println("Mittel-Server Error:"+ e.getMessage()); }
  }
}
```

```
import java.rmi.*;
public class MittelClient  {
 public static void main(String args[]) {
  try {
   MittelInterface mittelserver;
   String path;
   path = new String("rmi://" + args[0] + "/Mittel");
   mittelserver = (MittelInterface) Naming.lookup(path);
   double a = ...;
   double b = ...;
   double c = ...;
   double ergebnis = mittelserver.mittel(a,b,c);
   System.out.println("Ergebnis: "+ergebnis);
  } catch (Exception e) {
      System.out.println("Error: " + e.getMessage());
    }
 }
}
```

Dem Client muss also die Interface-Definition public interface MittelInterface ... bekannt sein.

4.)Applet:

```
import java.awt.*;
import javax.swing.*;
import java.util.*;
class SteuerThread extends Thread {
 private JApplet applet;
 SteuerThread(JApplet applet) {
  this.applet = applet; }
```

```
 public void run() {
  try {
   for (;;)
    { sleep(1000);
      applet.repaint(); }
  } catch (InterruptedException e) {}
 }
}
public class ZeitApplet extends JApplet {
 public void start() {
  SteuerThread st = new SteuerThread(this);
  st.start(); }
 public void paint(Graphics g) {
  GregorianCalendar cal = new GregorianCalendar();
  String uhrzeit = cal.get(Calendar.HOUR)+":"
                        +cal.get(Calendar.MINUTE)+":"+cal.get(Calendar.SECOND);
  g.setColor(Color.WHITE);
  g.fillRect(0,0,300,250); // lösche die alte Ausgabe
  g.setColor(Color.BLACK);
  g.setFont(new Font("Roman",Font.BOLD,24));
  g.drawString(uhrzeit,50,50);
 }
}
```

Webseite:

```
<HTML>
 <HEADER>
  <TITLE>Ein Applet zur Anzeige der Zeit</TITLE>
 </HEADER>
 <BODY>
  <H2>Die aktuelle Uhrzeit:</H2>
  <APPLET CODE="ZeitApplet.class" WIDTH=300 HEIGHT=250>
  </APPLET>
 </BODY>
</HTML>
```

5.)JAX-WS

Datei serverdir/service/MinMax.java auf dem Server-Knoten:

```
package service;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
@WebService
@SOAPBinding(style=Style.RPC)
public class MinMax {
 public int minimum(int x1, int x2, int x3) {
```

```
 int min = x1;
 if (x2<min) min = x2;
 if (x3<min) min = x3;
 return min;
}
public int maximum(int x1, int x2, int x3) {
 int max = x1;
 if (x2>max) max = x2;
 if (x3>max) max = x3;
 return max;
}
}
```

Datei serverdir/server/MinMaxServer.java auf dem Server-Knoten:

```
package server;
import javax.xml.ws.Endpoint;
import service.MinMax;
public class MinMaxServer {
 public static void main (String args[]) {
  MinMax minmaxServer = new MinMax();
  Endpoint endpoint = Endpoint.publish(
          "http://serveradresse:55555/minmax", minmaxServer);
 }
}
```

Datei clientdir/client/MinMaxClient.java auf dem Client-Knoten:

```
package client;
import service.MinMax;
import service.MinMaxService;
public class MinMaxClient {
 public static void main(String args[]) {
  MinMaxService service = new MinMaxService();
  MinMax minmax = service.getMinMaxPort();
  int a, b, c;
  a = ...; b = ...; c = ...;
  System.out.println("Minimum: "+minmax.minimum(a,b,c));
  System.out.println("Maximum: "+minmax.maximum(a,b,c));
 }
}
```

```
wsimport -keep http://Server-IP-Adresse:Portnummer/minmax?wsdl
```