Verteilte Systeme

Vorlesung 6 vom 13.05.2004
Dr. Sebastian Iwanowski
FH Wedel

Inhaltsverzeichnis für die Vorlesung

Zur Motivation: 4 Beispiele aus der Praxis

Allgemeine Anforderungen an Verteilte Systeme

Konzepte verteilter Hardware

Die Client-Server-Beziehung und daraus entstehende Fragestellungen

Grundlagen der Kommunikation in verteilten Systemen

Nebenläufigkeitstechniken

➔ Entfernte Aufrufe / Objektmigration

Namensverwaltung / Namenssuche

Dienstevermittlung

Synchronisation von Daten

Konzepte zur Erzielung von Fehlertoleranz

Sicherheit

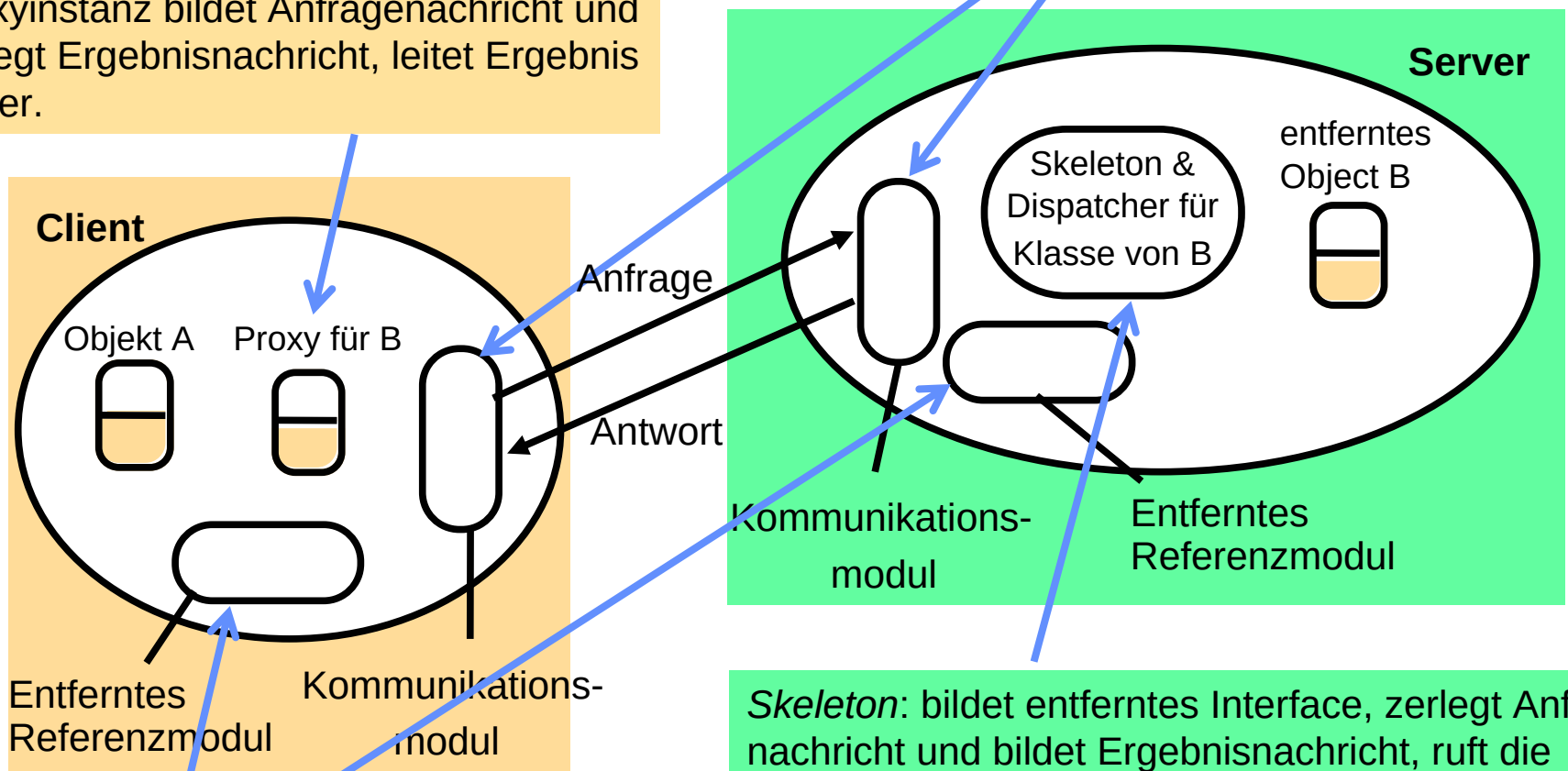
Ausblick auf konkrete Software: J2EE, SOAP,...

Remote Method Invocation: Funktionsablauf

Proxyklasse implementiert das entfernte Interface.

Proxyinstanz bildet Anfragenachricht und zerlegt Ergebnisnachricht, leitet Ergebnis weiter.

führt Kommunikations-protokoll aus



Übersetzt zwischen lokalem und entfernten Objekt. Erzeugt entfernte Objektreferenzen.

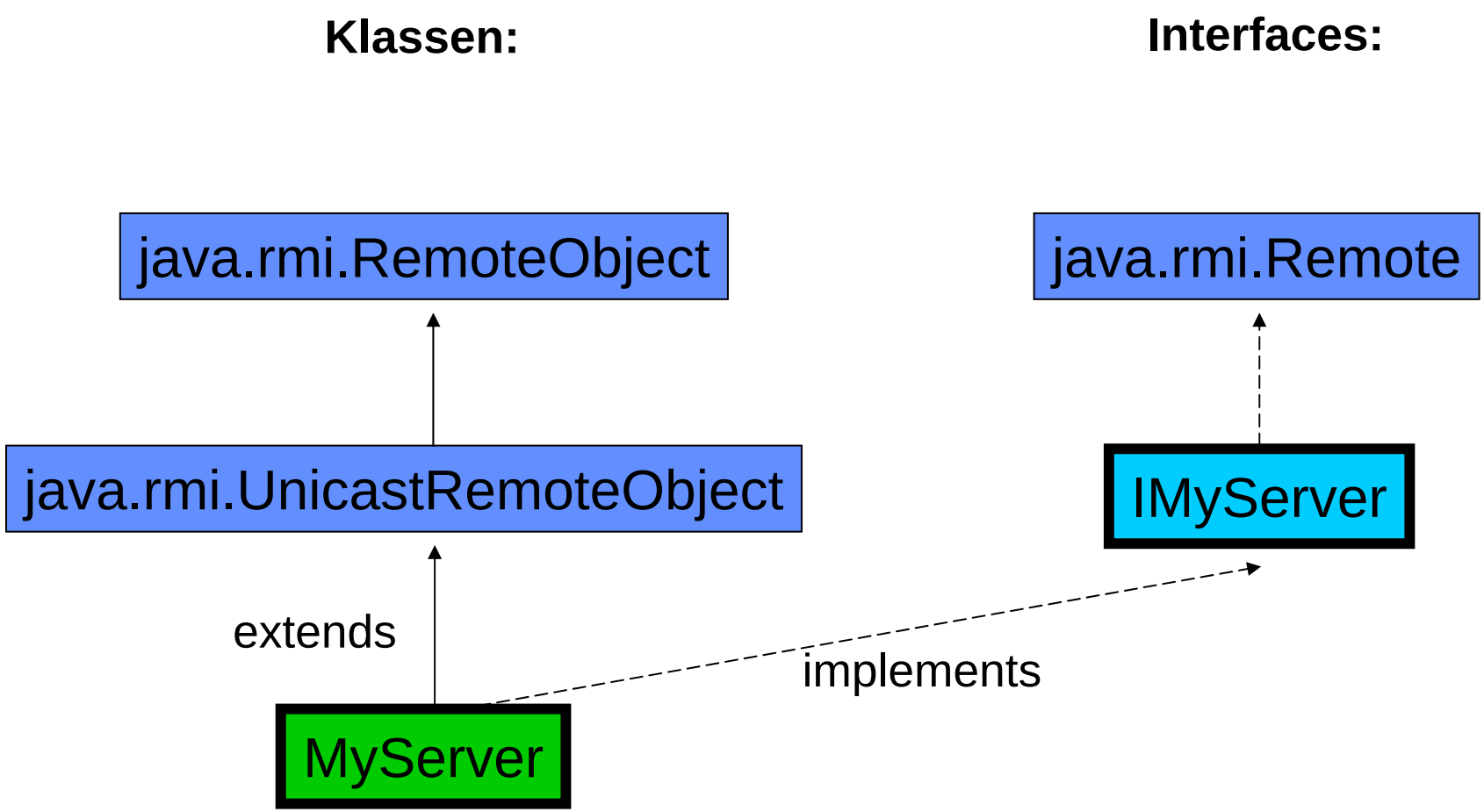
Skeleton: bildet entferntes Interface, zerlegt Anfrage-nachricht und bildet Ergebnisnachricht, ruft die implementierte Methode im entfernten Objekt auf

Dispatcher: bekommt die Nachricht vom Kommunikationsmodul, ruft die Methode des Skeletons auf (benutzt *methodID* der Nachricht).

Entfernte Aufrufe / Objektmigration

Teil 4: Java-RMI

Benötigte Klasseneinbindung des Servers



Beispiel: Schläger (Bat) trifft Ball

```
public class Bat {
    public void play (Ball ball) {   ball.hit();   }
    public static void main (String args[]) {
          Ball ball = new Ball();
          Bat bat = new Bat();
          bat.play(ball)  } }
```

soll auf Client laufen

```
public class Ball {
    public void hit() {
           System.out.println("Ball has been hit.") } }
```

soll auf Server laufen

Beispiel: Schläger (Bat) trifft Ball
Implementierung des Servers

Interface IBall:

```
import java.rmi.*;

public interface IBall extends Remote {
    public void hit() throws RemoteException;
}
```

Beispiel: Schläger (Bat) trifft Ball
Implementierung des Servers

Class Ball:

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

public class Ball extends UnicastRemoteObject implements IBall {
    public Ball() throws RemoteException {
            super(); }
    public void hit() {
            System.out.println("Ball has been hit."); }

    public static void main(Sting args[]) {
            try {
               Ball myBall = new Ball();
               Naming.rebind("Ball1", myBall);
            } catch (Exception e) { e.printStackTrace(); }}}
```

Beispiel: Schläger (Bat) trifft Ball
Implementierung des Clients

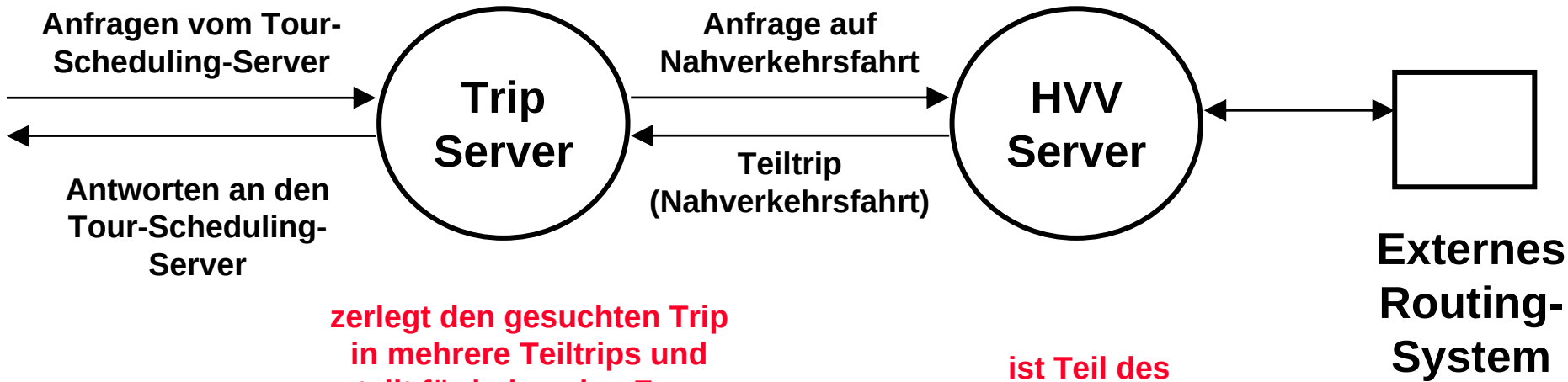
Class Bat:

```
import java.rmi.*;
public class Bat {
   public Bat () {
        super();
        System.setSecurityManager(new RMISecurityManager());
   }
   public void play(Ball ball) {
      try { ball.hit(); }
      catch (RemoteException e) { ... }
   }
   public static void main(String args[]) {
      Bat bat = new Bat();
      try {Ball ball = (Ball) Naming.lookup("hostname"+"Ball1");
           bat.play(ball); }
      catch (RemoteException e) { ... }
   }
}
```

Entfernte Aufrufe / Objektmigration

Praxisbeispiel:
Java-RMI im Touristeninformationssystem

Java-RMI im Touristeninformationssystem



zerlegt den gesuchten Trip in mehrere Teiltrips und stellt für jeden eine Frage an den entsprechenden Routing-Server, setzt die Antworten hinterher zusammen

ist Teil des Touristeninformationssystems, stellt Verbindung zur externen Informationsquelle dar.

Java-RMI im Touristeninformationssystem

Interface des TripServers (für Serverfunktion):

```
public interface ITripServer extends Remote
{
  public Trip getTrip(Integer queryId, TripQuery tripQuery)
        throws RemoteException;
        // für Anfrage des TourSchedulingServers
}
```

Java-RMI im Touristeninformationssystem

Konstruktor des TripServers:

```
public TripServer () throws RemoteException {
    ... // einige TripServer-spezifische Initialisierungen

    try {
      java.rmi.registry.LocateRegistry.createRegistry(1099);
       // macht Registraturtabelle auf (Serverfunktion)
    }
      catch(java.rmi.server.ExportException bindExc) { ...
       // für den Fall, dass Registraturtabelle schon existiert
    }

    System.setSecurityManager(new RMISecurityManager());
       // für Stubgenerierung des TripServers (Clientfunktion)

    try{
      Naming.rebind ("TripServer", this);
       // für Anfragen an den TripServer (Serverfunktion)
    }
    catch(Exception x){...}
}
```

Java-RMI im Touristeninformationssystem

Anfragemethode des TripServers an den HVVServer (für Clientfunktion):

```
private Trip ask // interne Methode des TripServers
  (StopPlace start, StopPlace dest, TimeStamp departure, TimeStamp arrival)
    ...
    TripQuery query = ...;
    try {
        IHVVServer hvvServer = (IHVVServer) Naming.lookup
                ("//" + System.getProperty ("HVVServerhost") +"/HVVServer");
        Integer publicQueryId = new Integer (maxQueryId++);
                // Zuweisung einer QueryId: Objekt wegen Serialisierbarkeit
        ...
        answer = hvvServer.processQuery (publicQueryId, query);
                // remote call (setzt automatisch neuen Thread)
        return answer;
    }
    catch (Exception e) {
        System.err.println
                ("TripServer: Exception while quering for the Trip: "
                        + e.getMessage());
        return null;
        }
```

Java-RMI im Touristeninformationssystem

Interface des HVVServers (für Serverfunktion):

```
public interface IHVVServer extends Remote{
  public SimTrip processQuery (Integer queryId, TripQuery query)
                                                    throws RemoteException;
        // für Anfrage des TripServers
}
```

Java-RMI im Touristeninformationssystem

Konstruktor des HVVServers:

```
public HVVServer () throws RemoteException {
    ... // einige HVVServer-spezifische Initialisierungen

    try {
      java.rmi.registry.LocateRegistry.createRegistry(1099);
       // macht Registraturtabelle auf (Serverfunktion)
    }
      catch(java.rmi.server.ExportException bindExc) { ...
       // für den Fall, dass Registraturtabelle schon existiert
    }

    //  System.setSecurityManager(new RMISecurityManager());
       // nicht nötig, wenn Router nicht in RMI-System

    try{
      Naming.rebind ("HVVServer", this);
       // für Anfragen an den HVVServer (Serverfunktion)
    }
    catch(Exception x){...}
}
```

Zusammenfassung: Java-RMI

- **maschinenunabhängige Namensverwaltung für Objektreferenzen**
- **Senden einer Nachricht (synchrone Kommunikation)**
- **RMI übernimmt Parameter- und Ergebnistransfer (Marshalling and Demarshalling):**
  - **Umrechnung in externe Datenrepräsentation**
  - **Wiederherstellung von Objektstrukturen**
  - **Nachrichtenobjekte müssen Interface "*serializable* " implementieren**
- **RMI übernimmt Threadsetzung und Koordination des Multithreading**
- **RMI läuft in heterogenen SW-Umgebungen, solange die Server die Virtual Machine von Java verwenden (betriebssystemunabhängig)**

Zusammenfassung: Java-RMI

einfache Sicherheitsfunktionen für RMI-Stubs

z.B. ist verboten:

- **Abhören von Ports**
- **Öffnen von Dateien**
- **Prozesse starten**
- **Manipulation fremder Threads**
- **Beenden einer Java Virtual Machine**

Probleme bei Java-RMI

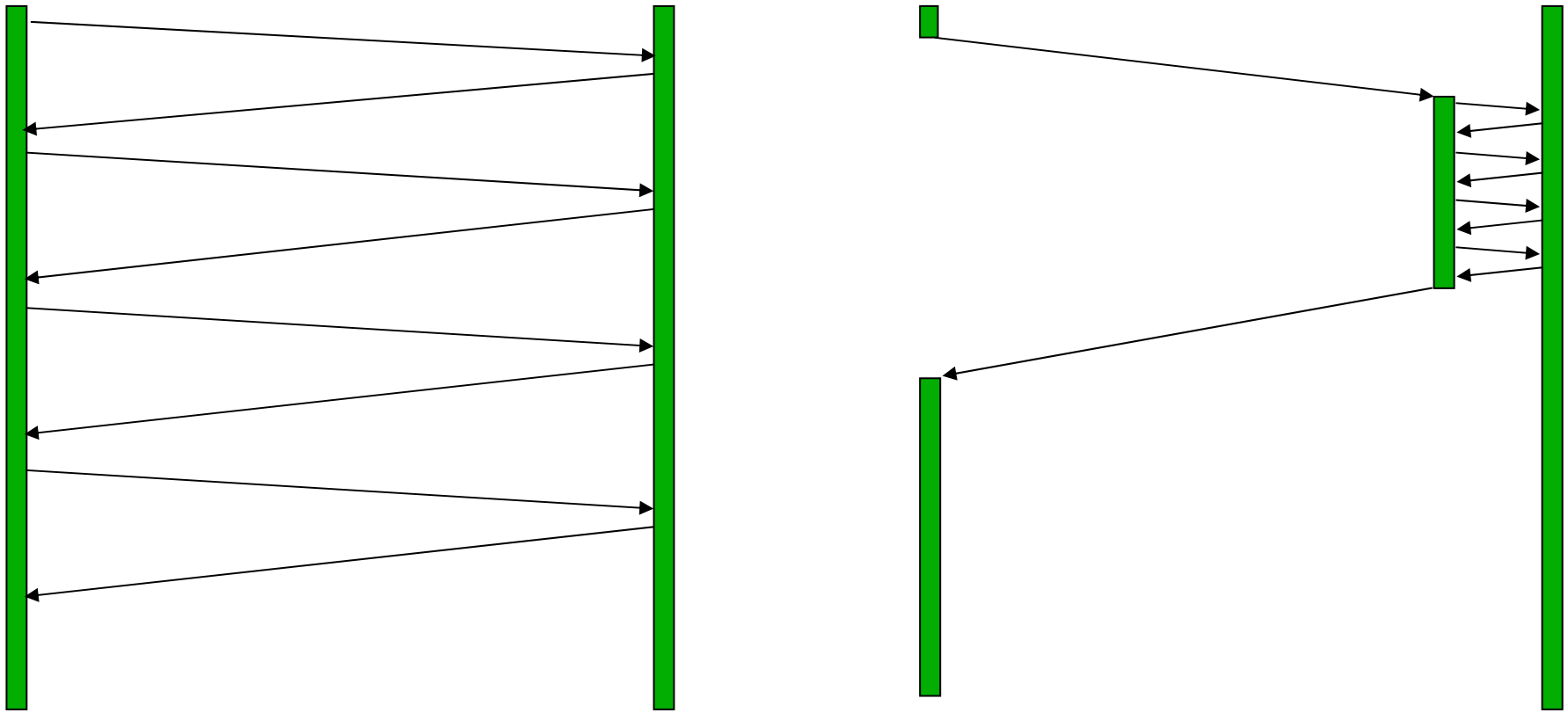
- **standardmäßig keine Verschlüsselung von Daten**
- **standardmäßig keine Authentifizierung**
- **kein Zugriffsrechtesystem: jeder kann die Registratur abfragen**
- **Stubs / Skeletons können simuliert werden: Trojaner möglich !**
- **keine Versionskontrolle zwischen Stub und Skeleton: Inkonsistenzen möglich**
- **keine Transaktionskonzepte für Transaktionen aus mehreren Methoden oder Zugriff auf gemeinsame Daten**

Entfernte Aufrufe / Objektmigration

letzter Teil:
Objektmigration

Objektmigration: Prinzip

Entfernter Zugriff **versus** **Migration**



Objektmigration: Anwendungsziele

- **Verringerung des Zeitaufwands für die Kommunikation**
- **Balancierung der Ressourcenverteilung**
- **Besitz- oder Verantwortungswechsel der SW im verteilten System**
- **Einsatz auf mobilen Geräten**
- **Ermöglichung des off-line-Vetriebs von Clients und Servern für lange Zeiträume**

Klassifikation nach Eigenschaften der migrierten Objekte

- **Objekte als Datenbehälter** — Klassische Datenübertragung
- **Objekte als Methodenträger** — Klassische Objektmigration
- **Objekte als autonome Softwareeinheiten mit eigenen Zielen** — Mobile Agenten

Anforderungen zur Ermöglichung von Objektmigration

gelten gleichermaßen für klassische Objektmigration und mobile Agenten:

- **Laufzeitumgebung an allen Hosts, die besucht werden sollen**
- **Registratur zum Auffinden von mobilen Objekten**
- **Stubs (Proxies), die zur Laufzeit verändert werden können**

Details zum Thema Mobile Agenten im Seminarvortrag:

Mobile Agenten

Informatik Seminar SS2004

Matthias Rohr (mi4295)

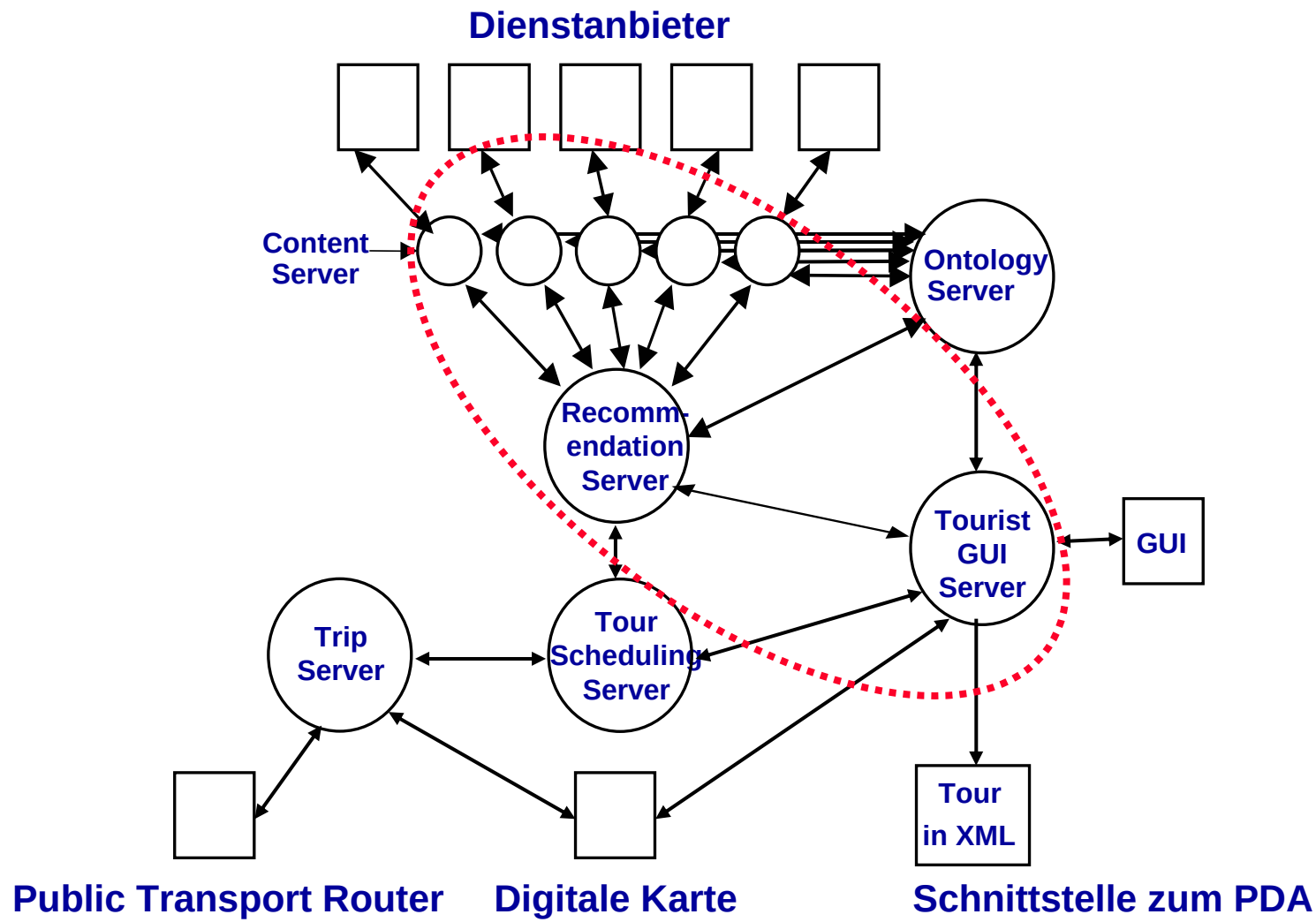
04.05.2004

Gliederung des Seminarvortrages

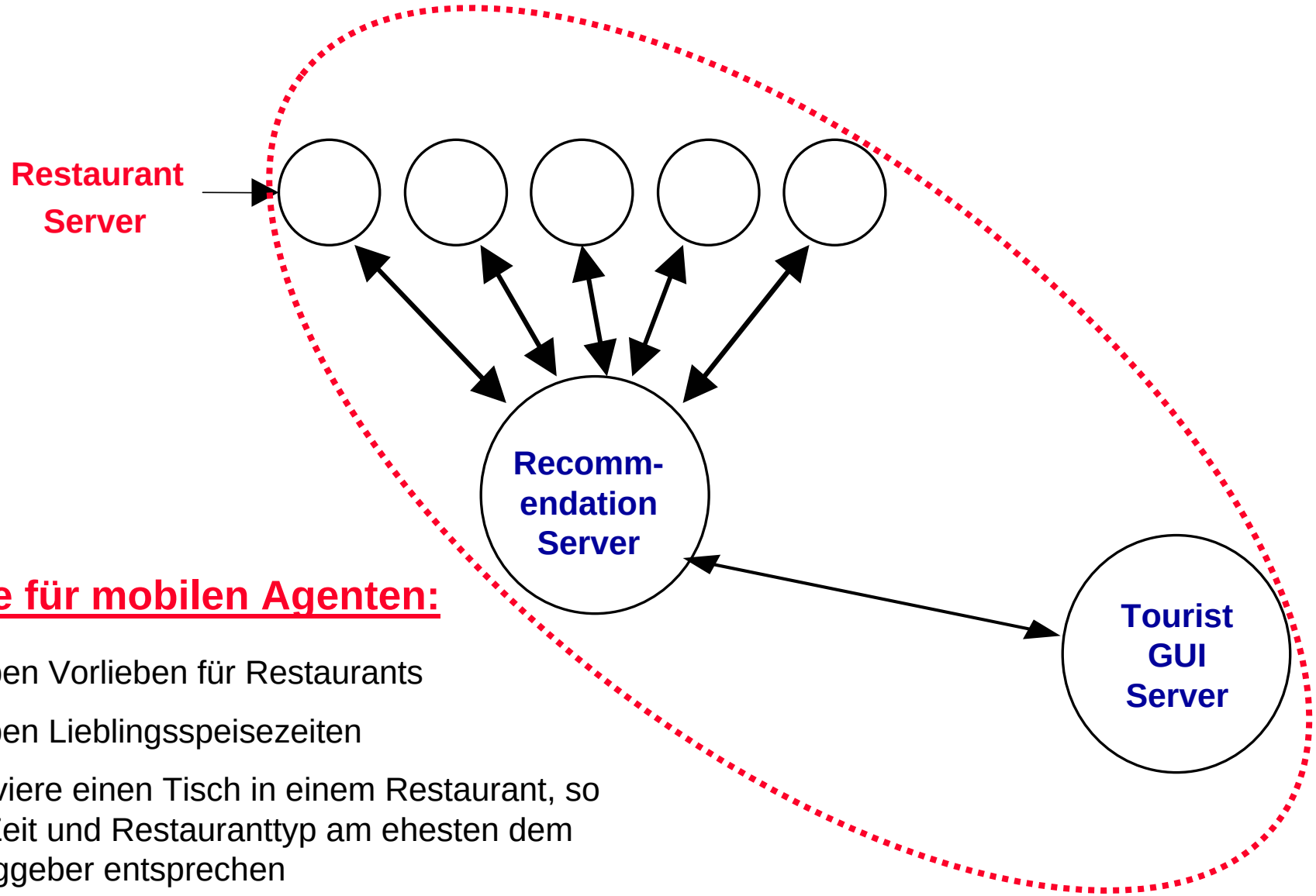
1. **Was sind mobile Agenten ?**
2. **Aspekte eines Agentensystems**
   - **Migration**
   - **Kommunikation**
   - **Sicherheit**
3. **Standardisierung**
4. **Fazit**

Die gesamte Präsentation befindet sich auf der Seminarseite im Internet (zu finden über ~iw/Lehrveranstaltungen).

Praxisbeispiel: Touristeninformationssystem



Praxisbeispiel: Touristeninformationssystem



Aufgabe für mobilen Agenten:

- Gegeben Vorlieben für Restaurants
- Gegeben Lieblingsspeisezeiten
- Reserviere einen Tisch in einem Restaurant, so dass Zeit und Restauranttyp am ehesten dem Auftraggeber entsprechen

Mobile Agenten: Mögliche Einsatzgebiete

- **Suchmaschinen**
- **Börsenbeobachter**
- **Elektronischer Preisvergleich**
- **Mehrwertleistungen**
- **Just-in-Time-Produktion**
- **Mobile Computing**

Wichtige Richtlinie:

Keine mobilen Agenten in sicherheitsrelevanten Anwendungen !

Mobile Agenten: Bewertung

Vorteile:

- **Reduktion der Netzwerkkommunikationslast**
- **Möglichkeiten zum Offline-Gehen des Auftraggebers, während der Agent aktiv ist**
- **Flexiblere Reaktion auf Umgebung möglich als z.B. reines RMI**

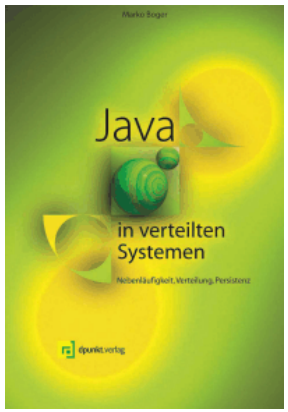
Nachteile:

- **Komplexität der Programmierung/Erstellung**
- **Gewisse Infrastruktur erforderlich**
- **Sicherheitsprobleme**
  (sowohl für den Agenten und als auch für die jeweilige Wirtsumgebung)

Fazit im Seminarvortrag zum Thema Mobile Agenten

- Viele Mögliche Anwendungsgebiete (v.a. durch steigende KI)
- Aber keine „Killerapplikation“ in Sicht
- Viele Unterschiedliche / Inkompatible Konzepte von Agentensystemen
- Standards lassen viele Punkte offen:
  - Agenten
  - Sicherheit
  - Anwendungen
- Kaum Umsetzung der vorhandenen Standards
- Konkrete Ansätze zur Portabilität in vorhandenen Systemen kaum erkennbar
- Mobile Agenten bisher noch hauptsächlich theoretische Disziplin

Weitere Java-Beispiele



Einfache Beispiele mit vollständigem Programmiercode in:

Marko Boger: *Java in verteilten Systemen*, Nebenläufigkeit, Verteilung, Persistenz, dpunkt.Verlag 1999, ISBN 3-932588-32-0

Wir haben bisher behandelt: Kap. 1 bis 4

Kap. 6 und 7: Objektmigration mit dem System Voyager

Programmiercode elektronisch verfügbar unter:

http://www.dpunkt.de/leseproben/3-932588-32-0/jivs_code.zip

Beim nächsten Mal:
Namensverwaltung / Namenssuche