Netzprogrammierung
13. Weitere Modelle der Netzprogrammierung

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto: tolk@inf.fu-berlin.de
http://www.robert-tolksdorf.de

Überblick

1. Schwächen des RPC-Konzepts
2. Koordinationssprachen
3. Peer-to-Peer
4. Agenten

Schwächen des RPC Konzepts

RPC

- RPC ist
  - sehr populär
  - weit übertragbar
  - implementierbar
- Aber:
  - RPC ist nicht abschließende Lösung
- [Tanenbaum/vanRenesse88] diskutieren einige der Probleme

Konzeptionelle Probleme

- Problem: Rollenidentifikation als Klient oder Server
  - sort < infile | uniq | wc -l > outfile
  - Wer liest, wer schreibt, wer betreibt die Berechnung?
    - fordert wc Zeilen vom uniq Prozess an?
    - fordert uniq den wc Prozess zur Weiterverarbeitung auf?
- Problem: Rollenwechsel in der Interaktion
  - Änderungsbenachrichtigungen an Klienten
    - Klienten sind dann auch Server
  - Signale des Klienten an Server
- Problem: Mehrparteieninteraktionen
  - Datenverteilung an mehrere Server

Weitere Probleme

- Transparenz bei Parametern
  - Zeiger
  - Globale Variablen
- Fehler
  - Server hat Fehler -> Klient blockiert
  - Klient hat Fehler -> Server steht alleine
  - Exactly-Once-Semantik schwer zu etablieren -> I/O
- Nebenläufigkeit
  - Blockierter Klient beim Aufruf (bei synchronem RPC)
  - Partielle Ergebnisse können nicht zur Weiterverarbeitung abgeliefert werden (z.B. bei Datenbankanfrage)

Alternativen zu RPC

- Neben dem RPC Konzept gibt es weitere Versuche, andere Interaktionsmodelle für Netzprogrammierung zu bilden
  - Koordinationssprachen: Tupelraum
  - Peer-to-Peer: Freie Rollen
  - Agenten: Autonome Komponenten

Koordinationssprachen

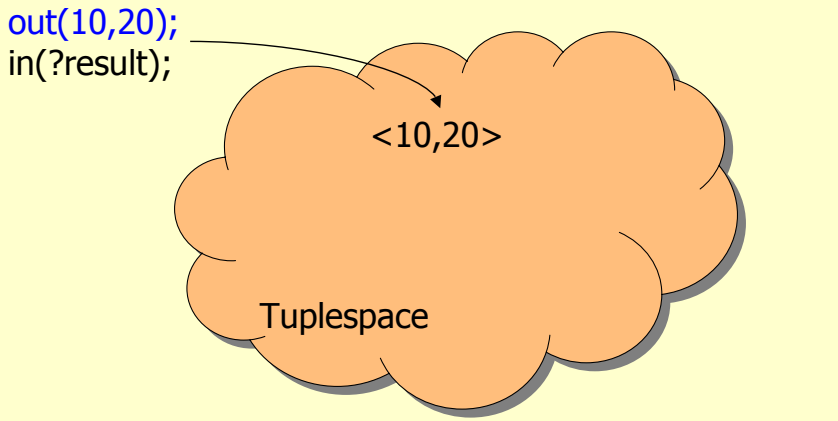
Koordinationssprachen

- Haben ihren Ursprung in der parallelen Programmierung mit der Sprache Linda
- Sind für netzbasierte Systeme ebenfalls anwendbar
- Haben sich dort zu einer Alternative zu RPC Modellen etabliert
- Sind teilweise kommerziell anerkannt
  - JavaSpaces, Sun
  - TSpaces, IBM

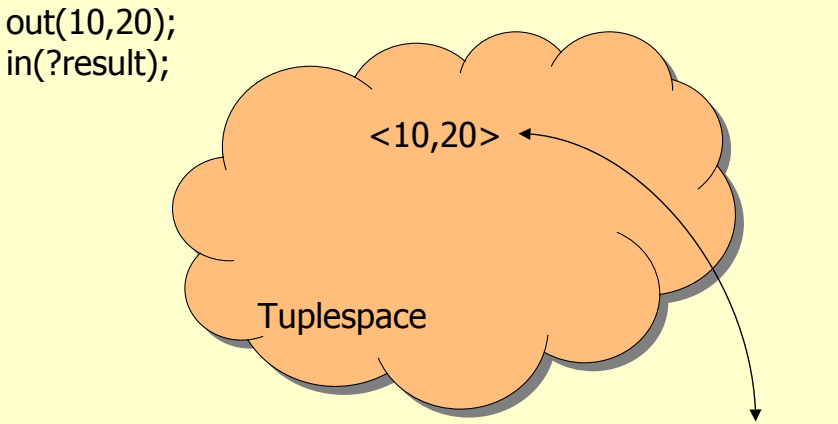
Koordinationssprachen

- Haupteigenschaften
  - Entitäten kommunizieren nur indirekt über einen gemeinsamen Datenraum (Tuplespace)
  - Kommunikationspartner sind anonym zueinander
  - Lebensdauer der Kommunikationspartner muss nicht überlappen
  - Mehrparteienkommunikation möglich
  - Inhärent nebenläufig
  - Abstrahiert völlig von Orten der Teilnehmer -> verteilt
  - Hauptproblem: Skalierbarkeit

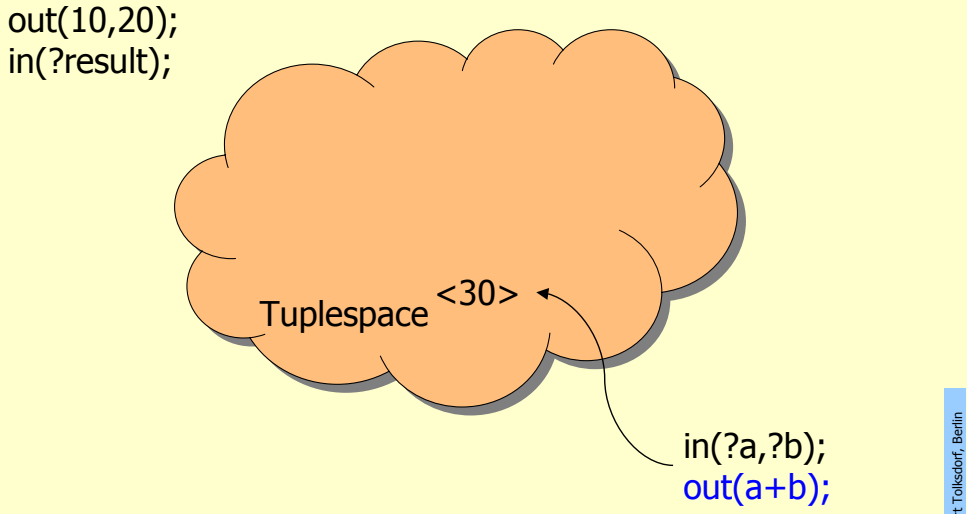
Indirekte Interaktion



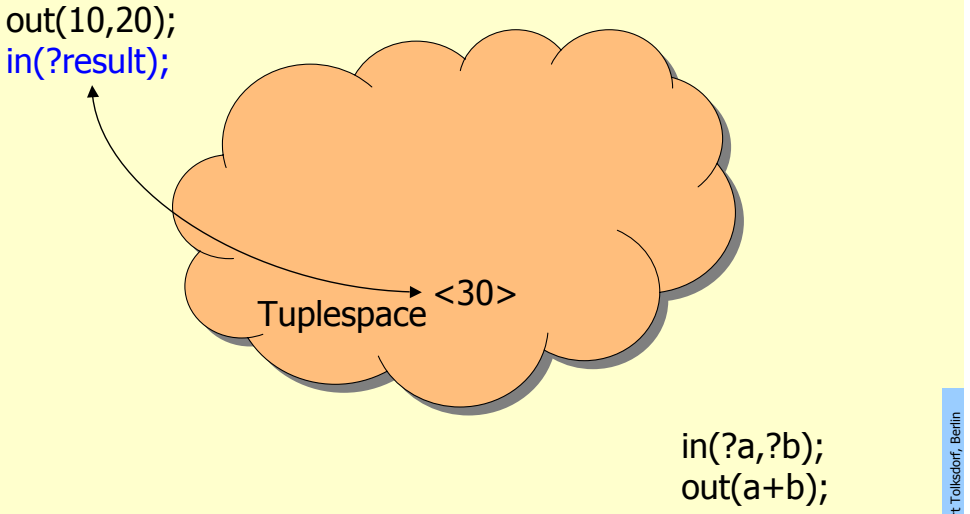
Indirekte Interaktion



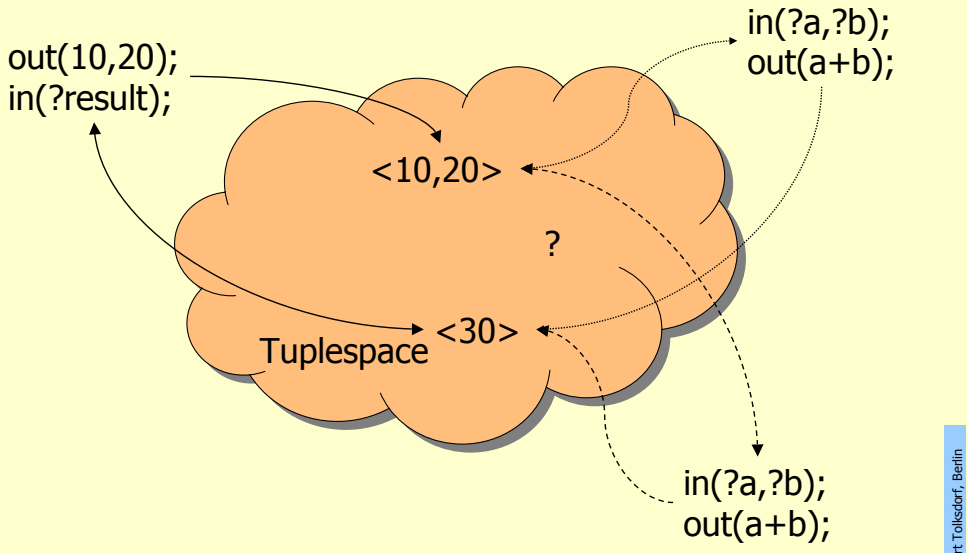
Indirekte Interaktion



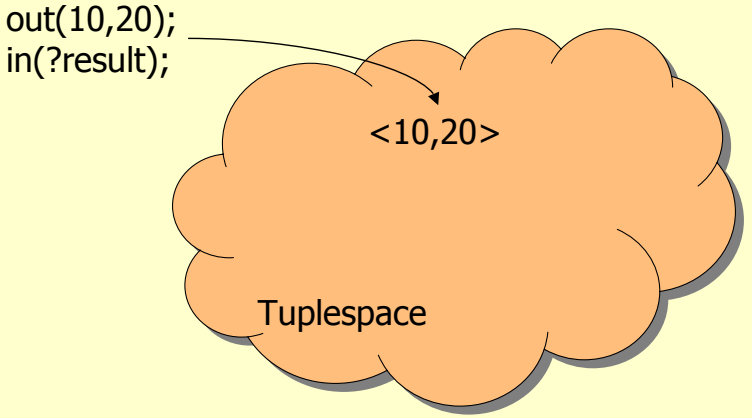
Indirekte Interaktion



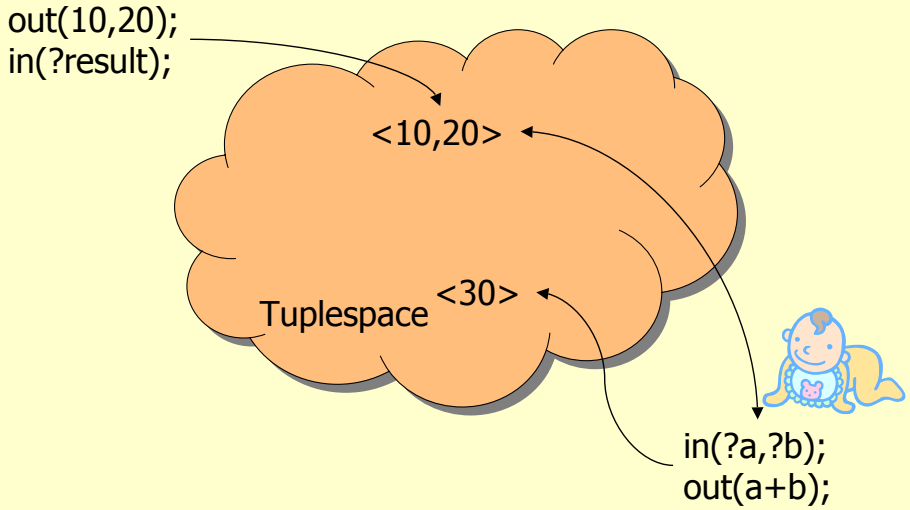
Anonyme Interaktion



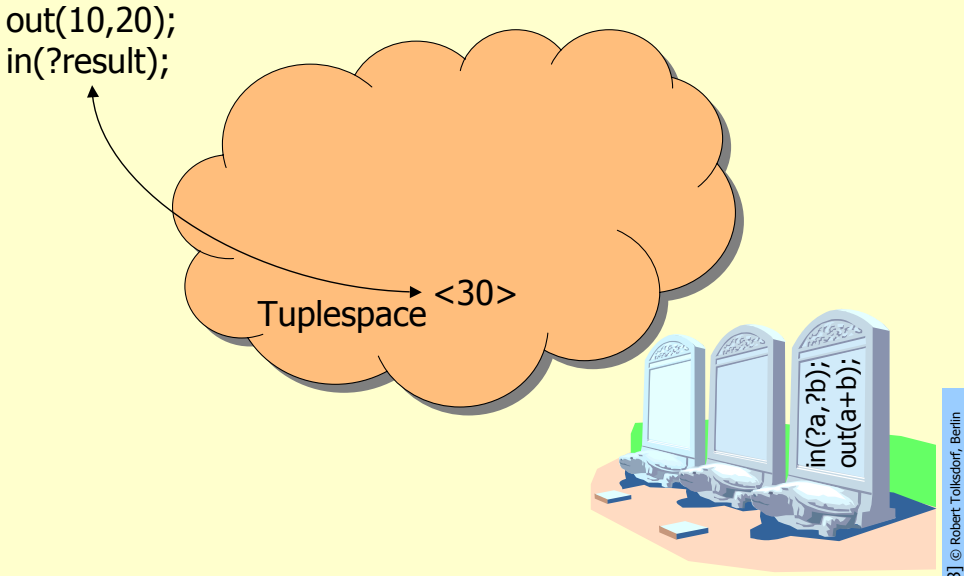
Unterschiedliche Lebensdauer



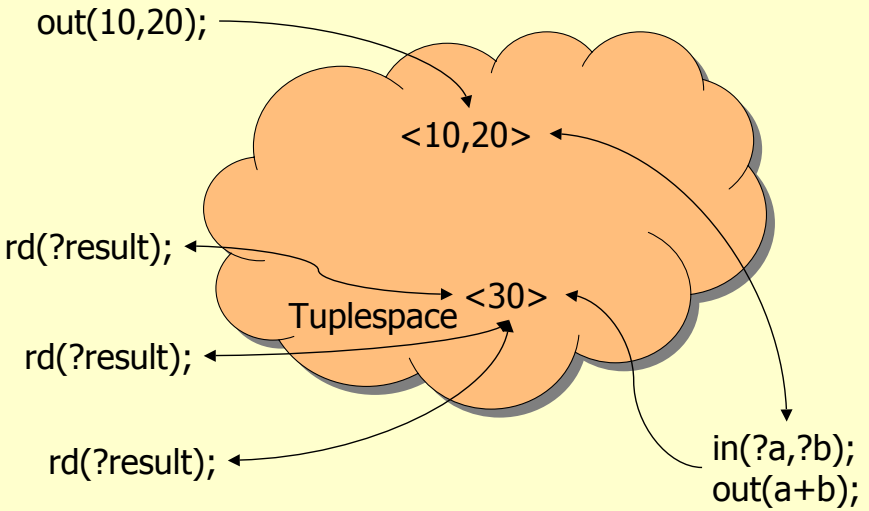
Unterschiedliche Lebensdauer



Unterschiedliche Lebensdauer



Mehrparteieninteraktion



Operationen

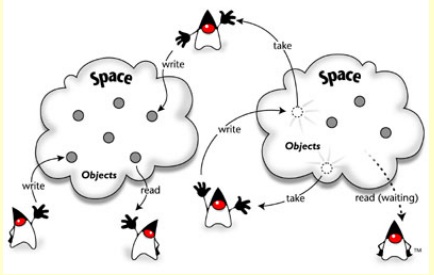
- out(tuple): Ablegen eines Tupels in den Tuplespace
- in(template): Herausnehmen eines passenden Tupels aus dem Tuplespace
  - Blockiert bis passendes Tupel vorliegt
    - Gleiche Anzahl Felder
    - Gleiche Typen der Felder
    - Gleicher Wert falls vorhanden
  - Zu <1,2,"start"> passen
    - <?int, ?int, ?string>, <1, ?int, ?string>, <1,2,"start">

    aber nicht
    - <?int, ?int>, <?int, ?string, ?int>, <10, ?int, ?string>, <1,2,3>
- rd(template): Auslesen eines passenden Tupels aus dem Tuplespace

Probleme

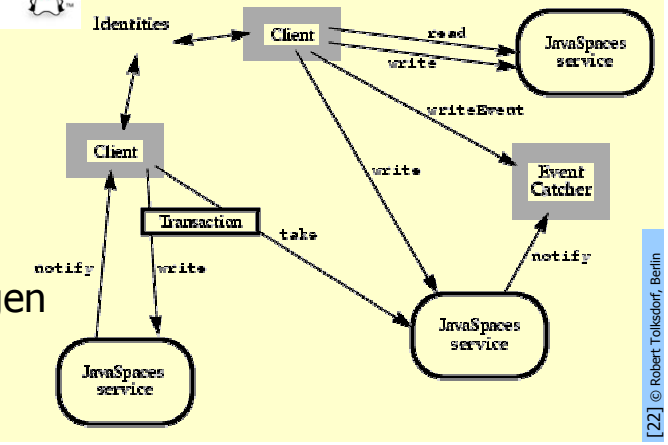
- Fehlertransparenz
  - Was passiert bei Fehler zwischen in(?a,?b); und out(a+b);?
- Termination von Interaktionen
  - Was passiert mit dem <30> Tupel, wenn alle interessierten Prozesse rd(?result); gemacht haben?
- Skalierbarkeit
  - Wie verteilt man den Tuplespace effizient?

JavaSpaces



Implementierung von Linda von Sun

- write
- take
- read
- Benachrichtigungen
- Transaktionen



Beispiel Entry Objekt

```
package jsbook.chapter1.helloWorldTwo;
import net.jini.core.entry.Entry;

public class Message implements Entry {
    public String content;
    public Integer counter;

    public Message() {  }

    public Message(String content, int initVal) {
        this.content = content;
        counter = new Integer(initVal);
    }

    public String toString() {
        return content + " read " + counter + " times.";
    }

    public void increment() {
        counter = new Integer(counter.intValue() + 1);
    }
}
```


Beispiel

```
package jsbook.chapter1.helloWorldTwo;

import jsbook.util.SpaceAccessor;
import net.jini.core.lease.Lease;
import net.jini.space.JavaSpace;

public class HelloWorldClient {
    public static void main(String[] args) {
        try {
            JavaSpace space = SpaceAccessor.getSpace();

            Message template = new Message();
            for (;;) {
                Message result = (Message)
                    space.take(template, null, Long.MAX_VALUE);
                result.increment();
                space.write(result, null, Lease.FOREVER);
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

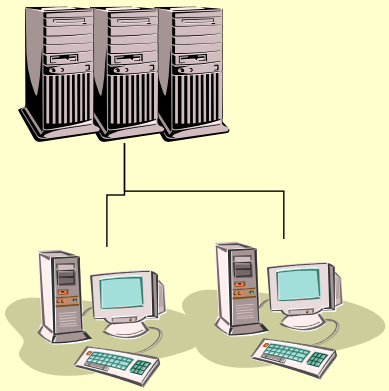
Beispiel

```
package jsbook.chapter1.helloWorldTwo;

import jsbook.util.SpaceAccessor;
import net.jini.core.lease.Lease;
import net.jini.space.JavaSpace;

public class HelloWorld {
   public static void main(String[] args) {
      try {
         Message msg = new Message("Hello World", 0);
         JavaSpace space = SpaceAccessor.getSpace();
         space.write(msg, null, Lease.FOREVER);
         Message template = new Message();
         for (;;) {
            Message result = (Message)
               space.read(template, null, Long.MAX_VALUE);
            System.out.println(result);
            Thread.sleep(1000);
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```


Peer-to-Peer

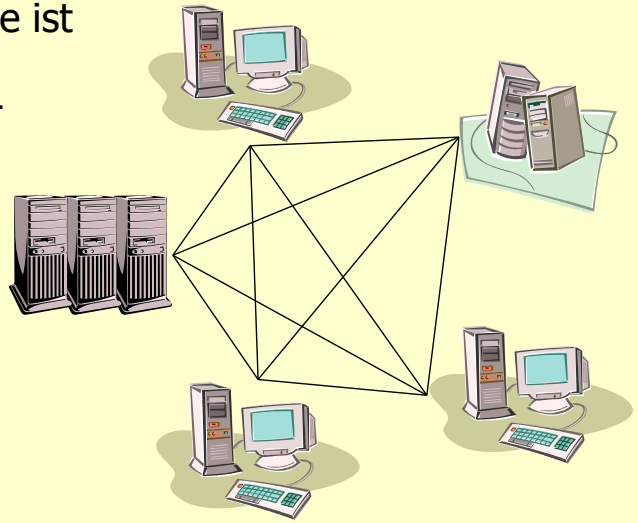
Peer-to-Peer

- Peer: der/die Ebenbürtige, der/die Gleichgestellte
- Client-Server:
  - Klient: Der Dienstnehmer
  - Server: Der Diensterbringer
  - *Sind nicht gleichgestellt*



Peer-to-Peer (P2P)

- Jeder arbeitet mit jedem zusammen
- Jeder ist gleichberechtigt
- Middleware ist erheblich komplexer



Peer-to-Peer (P2P)

- Peer-to-Peer, einfachste Definition: Es ist nicht Client-Server
- Server-Zentrierung
  - Anwendungsteile beim Klienten nutzen Dienste auf Server
  - führt zu Engpässen beim Server
  - macht Server zur kritischen Komponente (auch bei Replizierung)
  - Klienten treiben Anwendung vorwärts
  - Rigide Koordination (-> Transaktionen)
- P2P:
  - Anwendungsteile sind auf verschiedenen Rechnern, nutzen Dienste gegenseitig ohne Unterscheidung
  - Unterstützt
    - Wechselnde Rollen Klient/Server / Symmetrisches Client-Server
    - Asynchronität: Server benachrichtigt Klienten
    - Föderationen unter Gleichen

Peer-to-Peer (P2P)

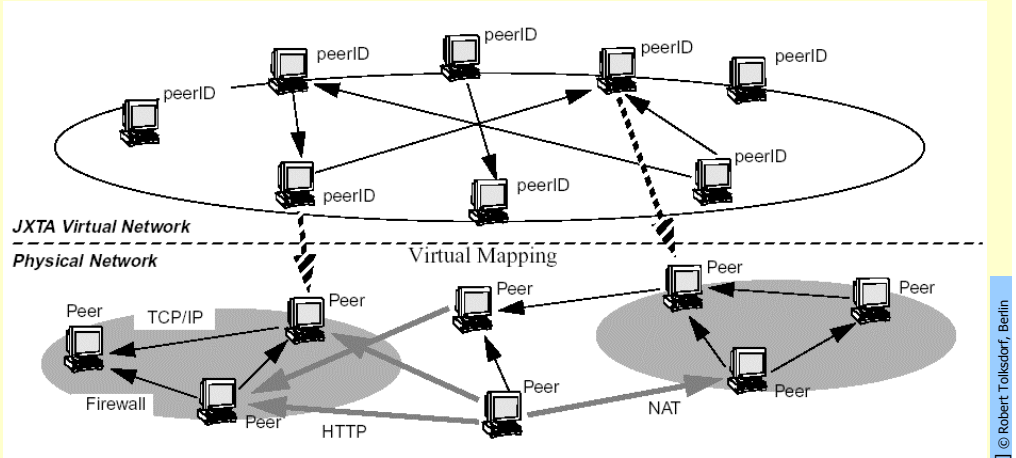
- Anwendungsszenarien
  - Verteilte Namens- und Verzeichnisdienste
  - Verteilte Dateisysteme mit Auslösung von Teilsystemen aus dem Netzwerk
  - Verteiltes Rechnen
  - Verteilte Mailverteilung
  - Hochskalierte ausfallsichere Systeme

JXTA

- P2P Plattform, Java-basiert
- www.jxta.org
- Spezialisierte P2P Systeme:
  - Gnutella: Filesharing
  - Napster: Musicfilesharing
  - AIM: Messaging
- JXTA:
  - Generische Infrastruktur für P2P
  - Anwendungs- und Plattformunabhängig

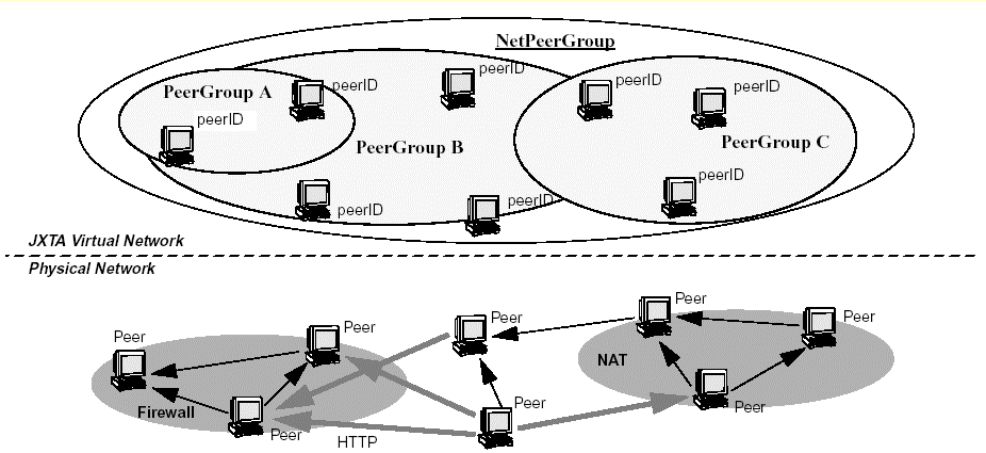
JXTA Konzepte

- Bezeichner
  - UUID ist 128 Bit lange eindeutige Bezeichnung
- Peer
  - Etwas, das die für einen Peer notwendigen Protokolle spricht



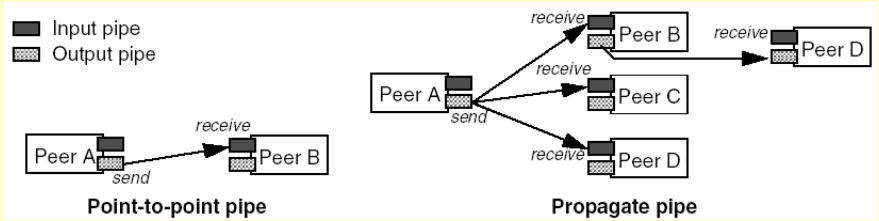
JXTA Konzepte

- PeerGroup
  - Etwas virtuelles, das die für eine PeerGroup notwendige Protokolle spricht



JXTA Konzepte

- Mitteilungen
  - Datagramm mit Sender, Empfänger sowie Inhaltsteilen mit Headern
- Pipe
  - Unidirektionale Kommunikationskanäle
  - Virtuell, können an mehrere Endpunkte gebunden werden

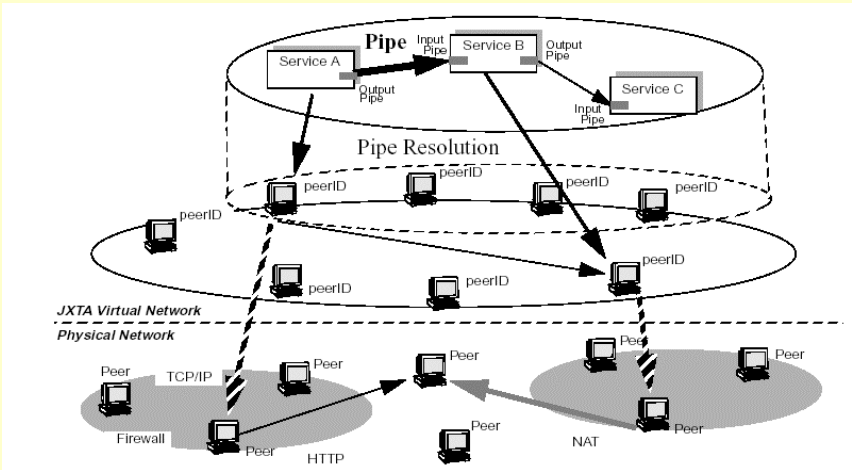


JXTA Konzepte

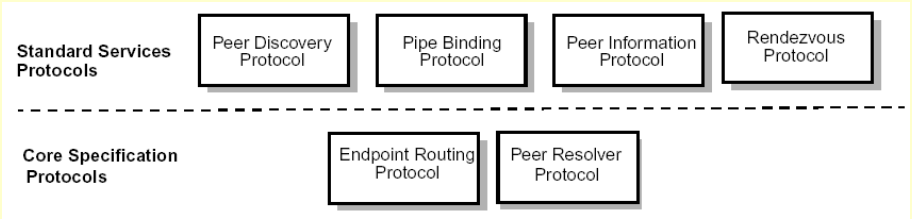
- Dienstangebot/Advertisement
  - XML-Dokument mit Angaben über Dienstangebot

```
<?xml version="1.0"?>
<!DOCTYPE jxta:LprAdv>
<jxta:LprAdv xmlns:jxta="http://jxta.org">
 <Id> urn:jxta:uuid-
596162616461626 14A757874614D504725184FBC4E5D498BB0919F772
FE006C704 </Id>
 <Name> Xerox workSet </Name>
 <Desc> Postscript Duplex Printer </Desc>
 <model> Xerox workSet </model>
 <res> 1200 DPI </res>
 <ipAdrr> 192.9.200.128 </ipAdrr>
 <port> 9100 </port>
 <latitude> 0.0 </latitude>
 <longitude> -0.0 </longitude>
 <pipe> urn:jxta:uuid-
596162616461626 14A757874614D504725184FBC4E5D498BB0919F772
FE006C704 </pipe>
 </jxta:LprAdv>
```


JXTA Konzepte



Protokolle



- Peer Resolver Protocol: Protokoll für generische Suchanfragen
- Endpoint Routing Protocol: Anfragen zu möglichen Routen zu anderem Peer (Peer ist dann Router)
- Peer Discovery Protocol: Auffinden von Dienstangeboten, Peers, Peergroups
- Pipe Binding Protocol: Binden eines Dienstangebots an eine PipePeer Information Protocol: Protokoll zur Abfrage von Peer-Eigenschaften
- Peer Membership Protocol Management von Gruppenmitgliedschaften

Agenten

Autonomie

- Die RPC-basierte Marssonde
  - Flugdatenübermittlung per RPC von der Erde
  - Ändert Kurs auf RPC Aufruf
  - Entscheidungen über Kursänderungen werden auf der Erde getroffen
- Unpraktikabel
  - Sehr lange Latenz
  - Entscheidungsfristen kleiner als Netzlatenz
  - Risiko der Fehlersemantik… Realer Absturz…
- Problem
  - Sonde ist nicht autonom in ihren Entscheidungen

Agenten

- Agent ist
  - ein Rechnersystem, das *eigenständig* für einen Benutzer agiert
- Mehragentensystem ist
  - ein Rechnersystem, das aus mehrere Agenten besteht, die miteinander *interagieren*
- Agenten müssen
  - kooperieren
  - sich koordinieren
  - verhandeln

Agenten

- Vorteile
  - Autonomie
  - Verschiebung der Sichtweise auf Systeme von der Berechnung zur Interaktion
  - Besseres Abbild realer Gesellschaften
- Nachteile
  - Unklare Realisierung
  - Unterscheidung zu
    - Verteilten Systemen (aber: Autonomie)
    - KI (aber: Verteiltheit und Realisierungsfrage)
    - Spieltheorie/Ökonomie (aber: Automatisierung)
    - Gesellschaftswissenschaften (aber: Informatiker)

Agenten

- Schwache Definition
  - Autonomie: Agent operiert ohne direkte Steuerung von außen und kontrolliert seine Aktionen selber
  - Responsiveness/Empfindlichkeit: Agent beobachtet Umgebung und reagiert auf Änderungen darin
  - Proaktiveness/Initiativ: Agent reagiert nicht nur, sondern wird selbständig zur Erreichung von Zielen aktiv
  - Sozial: Agent interagiert mit anderen zur Erreichung eigener und deren Ziele
- In der Regel mit Fokus auf Software-Agenten

Agenten

- Starke Definition
  - Mobilität: Agenten bewegen sich in ienem elektronischen Netzwerk (siehe auch: Roboter)
  - Veracity/Aufrichtigkeit: Agenten geben nicht wissentlich falsche Informationen weiter
  - Rationalität: Agenten beschädigen nicht durch Aktionen ihre eigenen Ziele
  - Kooperativität: Agenten arbeiten mit (menschlichen) Auftraggebern zusammen und übernehmen deren Ziele
- Zieht Aspekte menschlichen Verhaltens ein
- … Intelligent Agents, Smart Agents …

Architekturen

- *Deliberative* (Abwägend) *Architectures*
- Annahme: Intelligente Aktivität durch
  - symbolische Muster zur Repräsentation von Probleme
  - Operationen darauf zur Lösungsgenerierung
  - Suche auf Mustern zur Auswahl von Lösungen
- Agent sollte symbolisches Modell seiner Welt haben
- Beliefs, Desire, Intentions (BDI) Agenten:
  - Jeweils Modelle explizit repräsentiert
  - Tief erforscht
  - Schlecht skalierbar

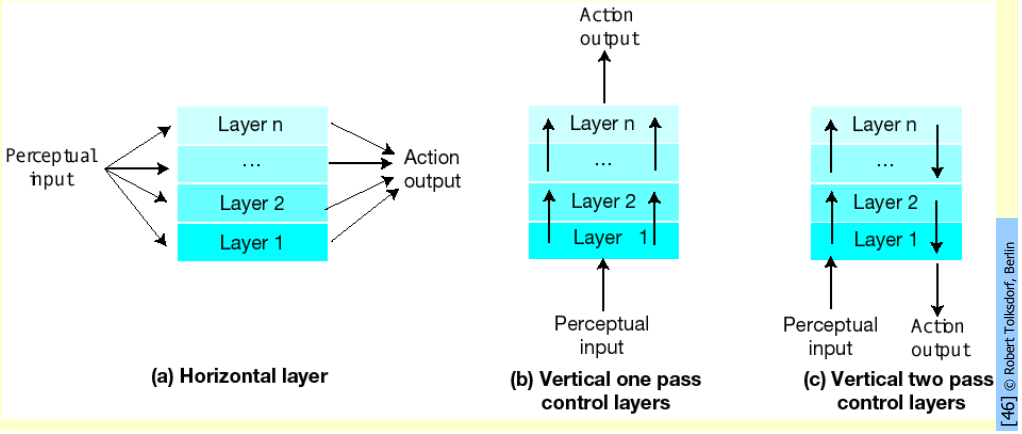
Architekturen

- *Reactive Architectures*
- Annahme: Intelligentes Verhalten durch Beobachung der Umgebung und Reaktion darauf
- Ohne interne Repräsentation
- Entscheidungen getroffen
  - mit wenig Informationen
  - einfachen Regeln (situativ)
  - in Echtzeit
- -> Autonome Roboter

- *Hybrid Architectures*
  - Kombination aus Deliberative/Reactive Architectures

Architekturen

- *Layered (geschichtete) Architectures*
- Agent besteht aus Subsysteme, die Problemteile verarbeiten
- Unterschiedliche Konfigurationen:



Implementierung

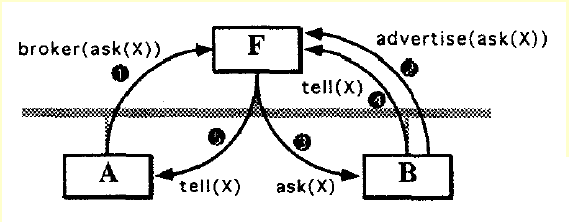
- Kooperation
  - Kooperative Interaktion: Agenten koordinieren ihre Aktionen. Koordination ist durch Programmierer vorgegeben
  - Vertragsbasierte Kooperation: Absichten der Agenten stehen leicht in Konflikt. Koordination durch Marktmechanismen wie Auktionen
  - Verhandlungsbasierte Kooperation: Verhandlungen zwischen Agenten, deren Resourcenbedarf in Konflikt steht
- Rationalität
  - Alleine: Agent handelt so, dass er seine Ziele erreicht
  - In Gemeinschaft: Agent handelt so, dass der gemeinsame Nutzen größer ist als der gemeinsame Verlust bei einer anderen Aktion

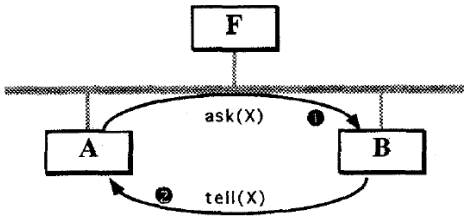
Implementierung

- Kommunikation
  - Agenten müssen Wissen austauschen
  - Agent Communication Languages (ACL)
    - KQML: Sprache zur Äußerung von Kommunikationsakten
      - Auf Sprechakten basiert
      - Mitteilungen haben Typ
      - Definierte Semantik
    - KIF: Sprache zur Repräsentation von Wissen
      - Wie wird der semantische Inhalt der Mitteilung repräsentiert
    - FIPA ACL
      - FIPA Konsortium
  - Ontologien
    - Worüber wird geredet
    - Was sind die Konzepte der Anwendungsdomäne

KQML

- (ask-all
  :content „price(IBM, [?price, ?time])“
  :receiver stock-server
  :language standard-prolog
  :ontology NYSE-TICKS)





Beispielsystem

- Zum Ausprobieren:
  JADE (Java Agent DEvelopment Framework)
- http://sharon.cselt.it/projects/jade/
- Java-basiert, FIPA kompatibel, Grafische Oberfläche

Zusammenfassung

Zusammenfassung

1. Schwächen des RPC-Konzepts
   1. Rollen, Transparenz, Fehler, Nebenläufigkeit
2. Koordinationssprachen
   1. Tuplespace
   2. Indirekt, anonym, zeitlich entkoppelt, Mehrparteieninteraktion, nebenläufig, verteilt
3. Peer-to-Peer
   1. Ebenbürtige Partner
4. Agenten
   1. Autonomie
   2. ACL

Literatur

- Andrew S. Tanenbaum and Robbert van Renesse. A critique of the remote procedure call paradigm. In Research into Networks and Distributed Applications (EUTECO 88), ed., R. Speth. Elsevier Science Publishers, 1988, pp. 775-783.
- David Gelernter, Nicholas Carriero. Coordination languages and their significance. Communications of the ACM, Volume 35, Issue 2 (February 1992). Pages: 97 - 107
- Eric Freeman, Susanne Hupfer, Ken Arnold. JavaSpaces Principles, Patterns, and Practice. Addison-Wesley, 1999.
- Li Gong. JXTA: A Network Programming Environment. IEEE Internet Computing. 5(3):88-95. 2001
- Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, Bill Yeager. Project JXTA 2.0 Super-Peer Virtual Network. Sun Microsystems, Inc. http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf
- Michael Wooldridge. An Introduction to Multiagent Systems. John Wiley & Sons, 2002.
- Eleni Mangina. Review of Software Products for Multi-Agent Systems. AgentLink (Hrsg.). http://www.agentlink.org/resources/other-pubs.html
- Tim Finin, Richard Fritzson, Don McKay, Robin McEntire. KQML as an agent communication language. Proceedings of the third international conference on Information and knowledge management. 456 – 463. 1994.
- Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf (Editors). Coordination of Internet Agents: Models, Technologies, and Applications. Springer Verlag, 2001. ISBN 3540416137.

