Jini

Kurzfassung als Kapitel für die Vorlesung „Verteilte Systeme“

Friedemann Mattern

- **J**ava **I**ntelligent **N**etwork **I**nfrastructure
- **J**ini **I**s **N**ot **I**nitials

Jini

- Infrastruktur („Middleware“) für dynamische, kooperative, spontan vernetzte Systeme
  - Zur Programmierung / Realisierung verteilter Anwendungen

Jini

- Infrastruktur („Middleware“) für dynamische, kooperative, spontan ver[illegible]te Systeme
  - Zur Programmierung / Realisie[illegible]verteilter Anwendungen

• Framework von APIs zu nützlichen Funktionen / Services
• Hilfsdienste (discovery, lookup,...)
• Standardprotokolle und Konventionen

Jini

- Infrastruktur („Middleware“) für dynamische, kooperative, spontan vernetzte Systeme
  - Zur Programmierun[illegible]ealisierung verteilter Anwendungen

•Dienste, Geräte,... finden sich „automatisch“ („plug and play“)
• Neu hinzukommende Komponenten bzw. Interaktionsbeziehung
• Mobilität

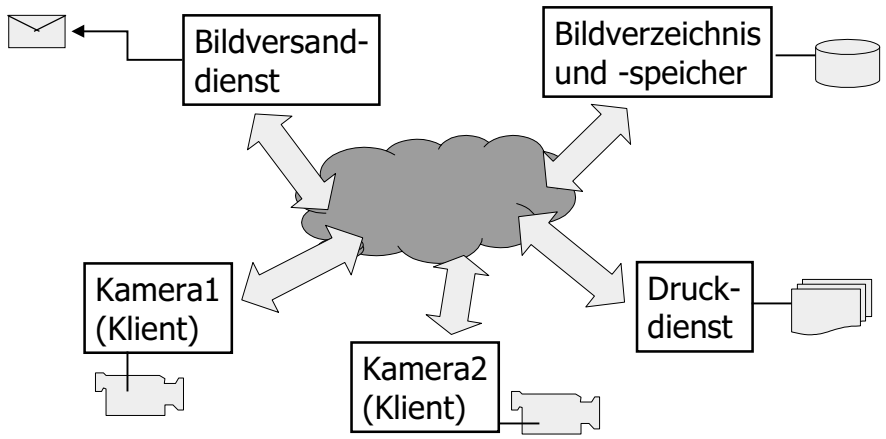
Jini

- Infrastruktur („Middleware“) für dynamische, kooperative, spontan vernetzte Systeme
  - Zur Programmierung / Realisierung verteilter Anwendungen
- Auf Java aufbauend und in Java implementiert
  - Typsichere (objektorientierte) Kommunikationsstruktur
  - Nutzt RMI (Remote Method Invocation)
  - „Überall“ JVM / Bytecode vorausgesetzt
  - Code shipping
- Konsequente Dienstorientierung
  - Alles ist ein Dienst (Hardware / Software / Benutzer)
  - Jini-System als Föderation von Services
  - Mobile Proxy-Objekte für Dienstzugriffe

Dienstparadigma

- Alles ist ein Dienst (Hardware / Software / Benutzer)
  - Vgl. Objektorientierung: „alles“ ist ein Objekt
  - Z.B. persistenter Speicher, Software-Filter, Auskunftsdienst…
- Laufzeitinfrastruktur von Jini bietet Mechanismen, um Dienste hinzuzufügen, zu entfernen, zu finden und zu verwenden
- Dienste werden über Interfaces definiert und stellen darüber ihre Funktionalität zur Verfügung
  - Dienst ist i.w. charakterisiert durch seinen (Objekt)typ und durch Attribute (z.B.: „600 dpi, Version 21.1“)
- Dienste (und Klienten als Dienstnutzer) finden sich „spontan“ zu Systemen zusammen („Föderation“)
  - Dienste können ihren Klienten Dienstproxies injizieren

Jini-Föderation



Jini, 7

Netzzentrierung

- Das Netz steht bei Jini im Vordergrund
  - Treu dem Motto: „the network is the computer“
- Netz = Hardware und Softwareinfrastruktur
  - Inklusive Hilfsdienste
- Sichtweise nicht „Geräte, die vernetzt werden“, sondern „Netz, an das Geräte angeschlossen werden“
  - Netz existiert immer, Geräte und Dienste nur vorübergehend
- Das Netz ist etwas Dynamisches
  - Komponenten und Interaktionsbeziehungen kommen hinzu oder verschwinden
- Jini will dyn. Netze und adaptive Systeme unterstützen
  - Neu hinzukommende / sich entfernende Komponenten sollen möglichst wenig Änderungen bei anderen Komponenten im Netz verursachen

Jini, 8

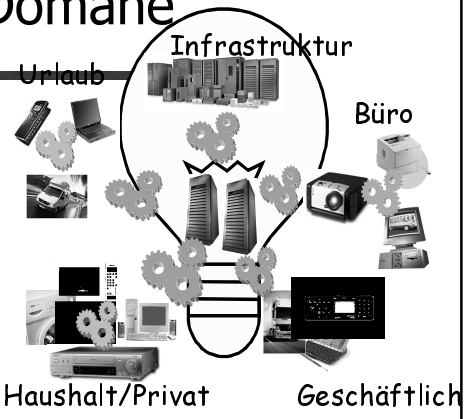
Spontane Vernetzung

- Objekte in einer offenen, verteilten, dynamischen Welt finden sich und bilden zeitlich befristete Gemeinschaft
  - Kooperation, Dienstnutzung…
- Typ. Szenario: Klient wacht auf (Gerät wird eingeschaltet, eingesteckt…) und fragt nach Diensten in der Umgebung
- Das Zueinanderfinden und Etablieren einer Beziehung soll schnell, unkompliziert und automatisch gehen

Wofür Jini?

- Infrastruktur für die Vision des Ubiquitous Computing
  - Wachsende Anzahl von Internet-Anschlüssen
  - Leistungsfähigere PDAs und Notebooks
  - Zunehmende Mobilität
  - Neue „Wireless Information Devices“
  - Unzählige Prozessoren in embedded Systems
    - z.B. Software-Update für die Waschmaschine
    - oder: Mikrowelle mit Internetanschluss?
  - Vernetzte Geräte
    - drahtlos, billig, hohe Bandbreite

Die Jini-Domäne



- Alles vernetzt...
  - Server
    - Access-Server
    - Webserver
    - Compute-Server
  - Desktop-Rechner
  - Mobile Geräte
    - Notebooks
    - Personal Digital Assistants (PDA)
    - SmartPhones
  - Haushalt: Waschmaschine, Spielzeug, ...
  - Eingebettete Systeme
  - Alltagsdinge („smart x“)

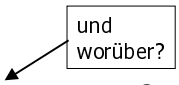
...und alle wollen kommunizieren!

Herausforderungen ubiquitärer Netze

und worüber?

- Wie redet der Toaster mit dem ABS-Steuercomputer?
  - Problem: Heterogenität von Hardware, Betriebssystem,...
  - Problem: Unterschiedliche Ressourcen, Umgebungen
  - Einheitliche „Sprache“? (z.B. Java-Bytecode, IDL, XML)

Herausforderungen ubiquitärer Netze



- Wie redet der Toaster mit dem ABS-Steuercomputer?
  - Problem: Heterogenität von Hardware, Betriebssystem,...
  - Problem: Unterschiedliche Ressourcen, Umgebungen
  - Einheitliche „Sprache“? (z.B. Java-Bytecode, IDL, XML)
- Umgang mit neuen Benutzungsmustern
  - Hohe Mobilität der Benutzer und Geräte
  - Neue Dienstleistungen / Geschäftsmodelle
    - $$$ durch Erbringung von Dienstleistungen
- Verbergen der Komplexität
  - Vor allem: Die Verwendung muss einfach sein
  - Keine manuelle Installation und Konfiguration
  - Adaption an das lokale Umfeld - nicht umgekehrt!

Jini, 13

Probleme mit aktueller Middleware

- Systeme verbergen das Netz und dessen „realen Probleme“ vor dem Programmierer
  - Programmierer „sehen“ das Netz und dessen Probleme nicht (Unzuverlässigkeit, Latenzen, Bandbreiten, ...)
  - Ausnahmebehandlung auf Anwendungsebene praktisch nicht möglich

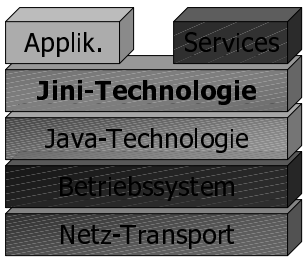
Jini, 14

Trugschlüsse bei verteilten Systemen

- Die ideale Sicht…
  - Das Netz ist verlässlich
  - Die Latenz ist null
  - Die Bandbreite ist unendlich
  - Das Netz ist sicher
  - Die Topologie ändert sich nie
  - Es gibt einen Systemverwalter
- …stimmt nicht mit der Realität überein
  - Jini nimmt sich einiger dieser Punkte an
  - Sie werden zumindest nicht verborgen oder ignoriert

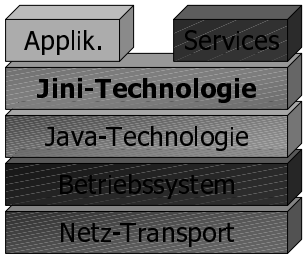
Jini aus der Vogelperspektive

- Jini ist eine Menge von APIs in Java2
- Ist eine Erweiterung der Java2-Plattform hinsichtlich verteilter Programmierung
- Ist Abstraktionsschicht zwischen Applikation und zugrundeliegender Infrastruktur (Netz, Betriebssystem)
  - Jini ist „Middleware“



Jini aus der Vogelperspektive

- Jini ist eine Menge von APIs in Java2
- Ist eine Erweiterung der Java2-Plattform hinsichtlich verteilter Programmierung
- Ist Abstraktionsschicht zwischen Applikation und zugrundeliegender Infrastruktur (Netz, Betriebssystem)
  - Jini ist „Middleware“



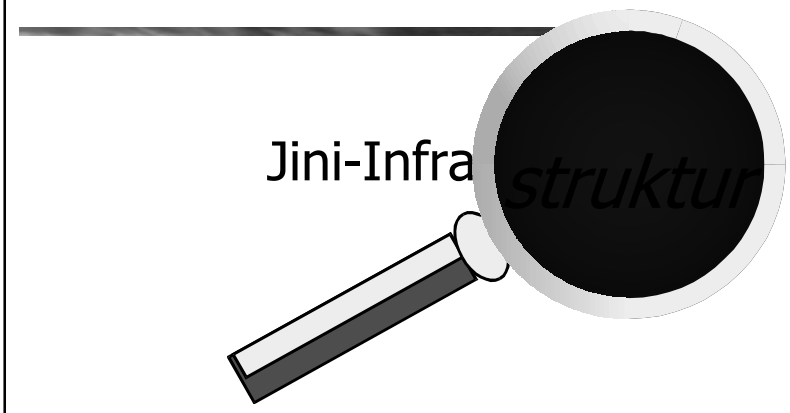
- Erweiterung von Java in drei Dimensionen:
  - Infrastruktur (z.B. Discovery, Lookup)
  - Programmiermodell (z.B. verteilte Ereignisse)
  - Dienste

Java-Bezug von Jini

- Setzt insbesondere JVM (als Bytecode-Interpreter) und RMI voraus
  - Homogenität in einer ansonsten heterogenen Welt
  - Ist das realistisch und letztendlich erfolgreich?

können Basisprotokoll für Discovery und Join; besitzen eine JVM; ...

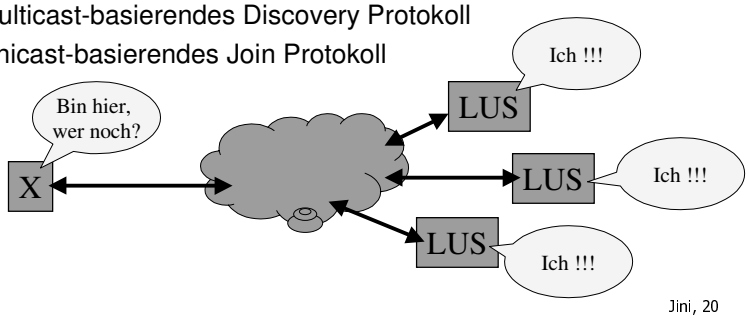
- Allerdings: Geräte, die nicht „Jini enabled“ sind bzw. keine JVM haben, können diesbezüglich von einem Software-Stellvertreter (ggf. irgendwo im Netz) verwaltet werden
  - „Device Bay“ bzw. „Proxy“
- Sicherheit („safety“) von Java überträgt sich auf Jini
  - Typsicherheit, referentielle Integrität, Prüfung von Array-Grenzen,...



Jini, 19

Jini Infrastruktur

- Ziel: Spontane Vernetzung und Bildung verteilter Systeme ohne apriori Wissen über das lokale Netz
  - Wie bekommen neue Dienstleister und Dienstnutzer Kenntnis von den lokalen Gegebenheiten?
  - Lookup-Service als Anlaufstelle für Dienstanbieter und Klienten
  - Multicast-basierendes Discovery Protokoll
  - Unicast-basierendes Join Protokoll



Jini, 20

Jini-Infrastruktur

- Wesentliche Strukturkomponenten sind:
  - Lookup-Service als Repository / Namensdienst / Trader
  - Protokolle aufbauend auf TCP / IP
    - Discovery&Join, Lookup von Diensten
  - Proxy-Objekte (Programmcode wird zum Klienten transportiert)
- Ziel: Spontane Vernetzung und Bildung verteilter Systeme ohne Apriori-Wissen über das lokale Netz
- Wie bekommen neue Dienstleister und Dienstnutzer Kenntnis von den lokalen Gegebenheiten?

Lookup-Service (1)

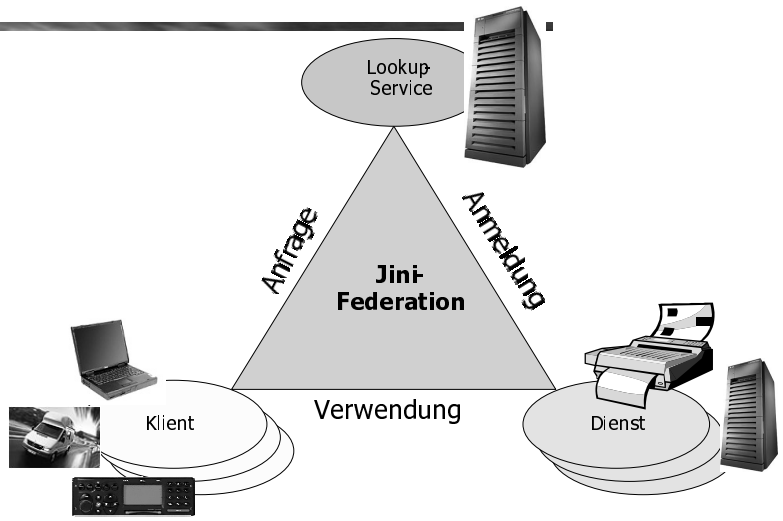
- Ähnlich RMI Registry und CORBA Naming Service
- Kernelement jeder Jini-Föderation
- Repository für Dienstanbieter
- Aufgabe:
  - Anlaufstelle für Dienste und Klienten
    - Registrierung von Diensten (Dienste machen sich publik)
    - Vermittlung von Diensten (Klienten finden Dienste)
  - Bietet Mechanismen, um Dienste und Klienten zusammenzuführen

Lookup-Service (2)

- Jini-Systeme gruppieren sich um einen oder mehrere Lookup-Service(s)
- Dienste registrieren Dienstleistung und Fähigkeiten beim Lookup-Service (Registrierung)
- Klienten finden Dienste über Lookup-Service („Lookup“)
- Weitere Möglichkeiten:
  - Erhöhung der Robustheit durch redundante Auslegung der Lookup-Services
  - Zuständigkeiten können auf mehrere (logisch getrennte) Lookup-Services verteilt werden

Jini, 23

LUS (3)



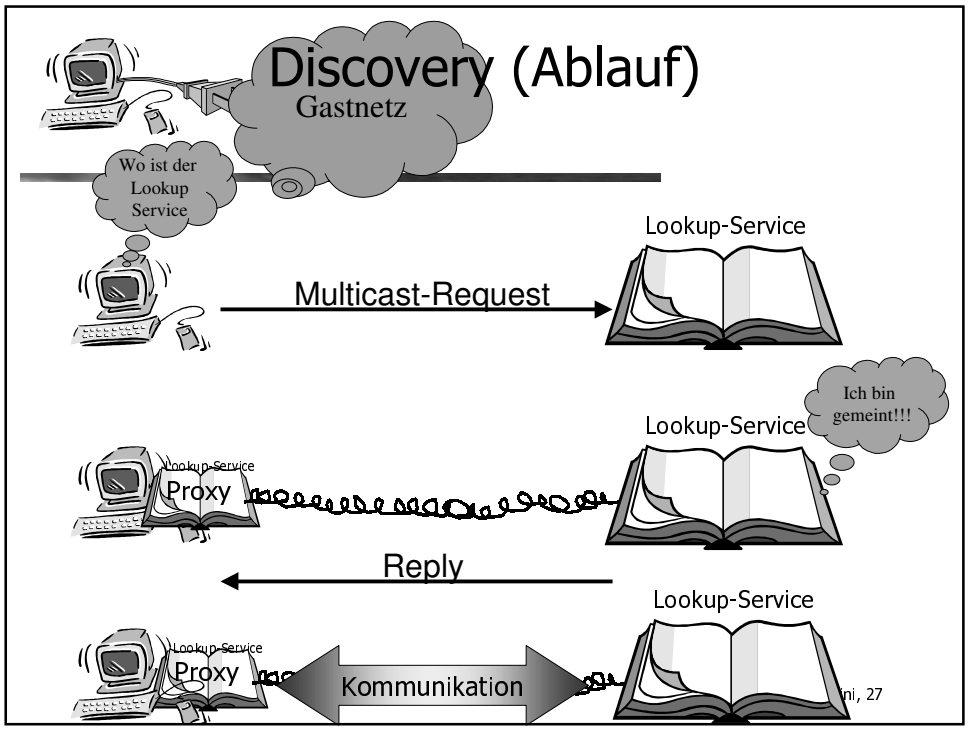
Jini, 24

LUS (4)

- Verwendet JavaRMI als Grundmechanismus
  - Daten und Code können im Netz „wandern“ (Code shipping)
- Dienstregistrierung:
  - Serialisierter **Service-Proxy** (Schnittstelle zum realen Dienst) und beschreibende Attribute werden im LUS abgelegt
- Im Unterschied zu klassischen Namensdiensten nicht nur Name / Adresse für einen Dienst, sondern auch
  - Menge von Attributen
    - Bsp: Printer(color: true, dpi: 600,...)
  - Können auch komplexere Klassen sein
    - Insbesondere unterschiedliche graphische Benutzungsschnittstellen

Discovery: Finden eines LUS

- Ziel: Ohne Kenntnis des Netzes den Lookup-Service finden zwecks
  - Bekanntmachen („Registrieren“) des eigenen Dienstes
  - Verwenden („Suchen“) eines vorhandenen Dienstes
- Discovery-Protokoll:
  - Multicast an bekannte Adresse („well-known address / port“)
  - Identifiziere Interesse an bestimmten Gruppen in Jini
  - Lookup-Service(s) antwortet mit serialisiertem JavaRMI-Objekt (Interface `ServiceRegistrar`)
    - Proxy-Objekt des Lookup-Service wird geladen
    - Kommunikation mit LUS über Proxy (ggf. proprietäres Protokoll)



Multicast Request Protocol

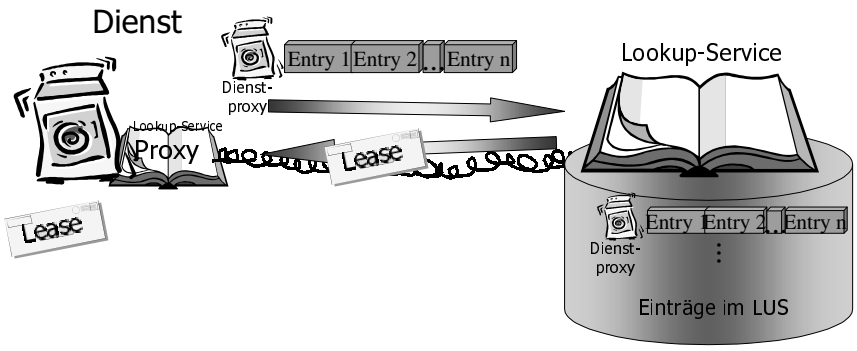
- Benötigt keine Kenntnisse über das Gastnetz
- Aktive Suche nach Lookup-Service(s)
- Discovery-Request basiert in IP-Netzen auf Multicast-UDP
  - „Empfohlen“ eine TTL von 15
  - Multicast-Gruppe für Discovery-Requests ist 224.0.1.85
  - Port-Nummer der Lookup-Services ist normalerweise 4160
    - Hexadezimale Subtraktion: $CAFE_{16}$-$BABE_{16}$=4160_{10}
- Discovery-Reply basiert auf TCP-Verbindung
  - Der Port für die Antwort wird in Multicast mitgesendet

Jini, 28

Join: Anmelden eines Dienstes

- Dienst hat bereits Schnittstelle zum Lookup-Service
  - Kann darüber eigene Dienstleistung anmelden (`register()`)
  - Teilt dem Lookup-Service dazu mit:
    - Eigenen Dienstproxy
    - Menge der genauer qualifizierenden Attribute
  - Erhält einen Lease für den Eintrag
  - Dienst kann dann im Jini-Verbund gefunden und verwendet werden

Join (Ablauf)



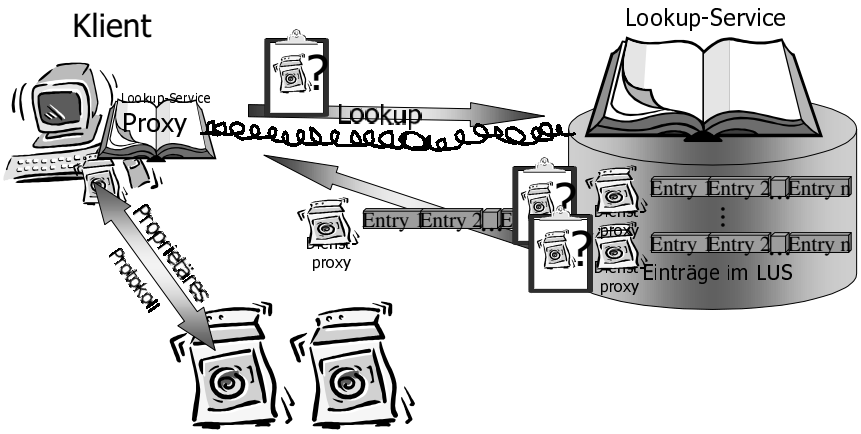
Join: Weitere Eigenschaften

- Dienst braucht für Registrierung folgende Informationen:
  - Seine ServiceID (falls vorhanden)
  - Menge der Attribute, Menge der Gruppen
  - Ggf. Liste von speziellen Lookup-Services
- Dienst wartet eine zufällige Zeit nach dem Start
  - Soll „Netzsturm“ nach Neustart des Netzes verhindern
- Registrierung in Lookup-Services ist immer mit Leases verbunden
  - Der Dienst muss den Lease regelmässig verlängern
- Registrierung wird über Klasse `JoinManager` gekapselt, die auch Lease-Erneuerung übernimmt

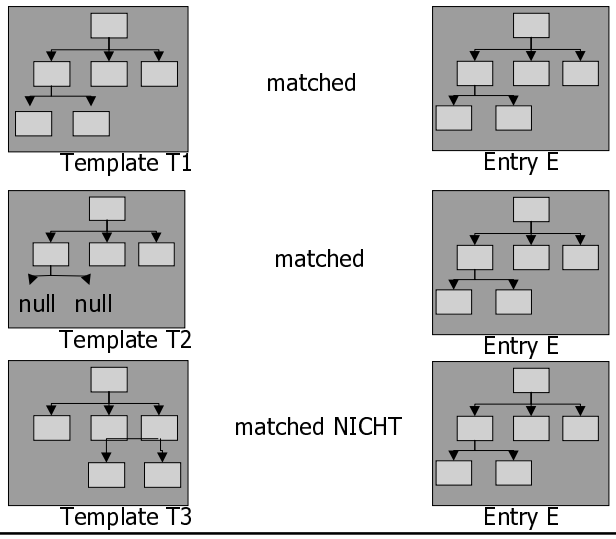
Lookup: Suchen eines Dienstes

- Klient kennt Lookup-Service (z.B. über Discovery-Protokoll)
- Sucht bestimmten Dienst
- Formuliert dazu Anfrage an den Lookup-Service
  - Mittels eines „Service-Template“
  - Matching nach Registrier-Nummer des Dienstes und / oder Typ und / oder Attributen
  - Wildcards erlaubt, ansonsten „exact-match“
- Lookup-Service liefert i.a. Menge aller Treffer zurück
- Auswahl des Dienstes lokal beim Klienten
- Verwendung des Dienstes erfolgt über Methodenaufrufe im Dienstproxy
- Protokoll Proxy $\leftrightarrow$ Dienst ist beliebig!!!

Lookup (Ablauf)

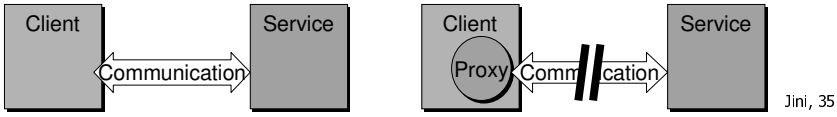


Lookup



Proxies und Smart-Proxies

- Proxies sind reine Stellvertreter des Dienstes
  - Reichen alle Daten zum Dienst durch
  - Sinnvoll bei kleinenDatenmengen, „schwachen“ Klienten
- Smart-Proxies erbringen Dienst teilweise oder ganz beim Klienten
  - Oftmals Vorverarbeitung von Daten möglich/sinnvoll
    - Video-Kompression oder Daten-Filter
  - Rechenleistung des Klienten notwendig/vorhanden
- Entscheidung liegt allein beim Dienst-Designer
  - Keine (direkte) Einflussmöglichkeit des Klienten



Jini, 35

Gruppen

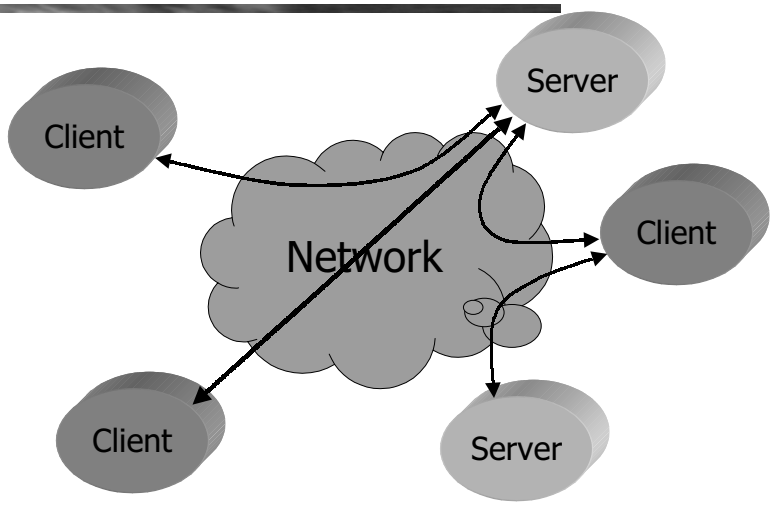
- In einem grösseren Jini-System kann es viele Lookup-Services geben
- Idee: Zuständigkeiten für bestimmte Gruppen von Diensten einführen
  - Sogenannte „Lookup-Groups“
  - Kontakt mit Lookup-Services immer mit Liste der interessanten Gruppen
  - „Fremde“ Gruppen werden ignoriert
  - Lediglich ein Freitextbezeichner
- Bsp: In einer Werkstatt existieren Lookup-Services für Werkstattbereich, Büro, Lager, etc.; Anmeldung eines Fahrzeugs soll begrenzt sein auf alle Lookup-Services, die der Gruppe „Werkstatt“ zugeordnet sind

Jini, 36

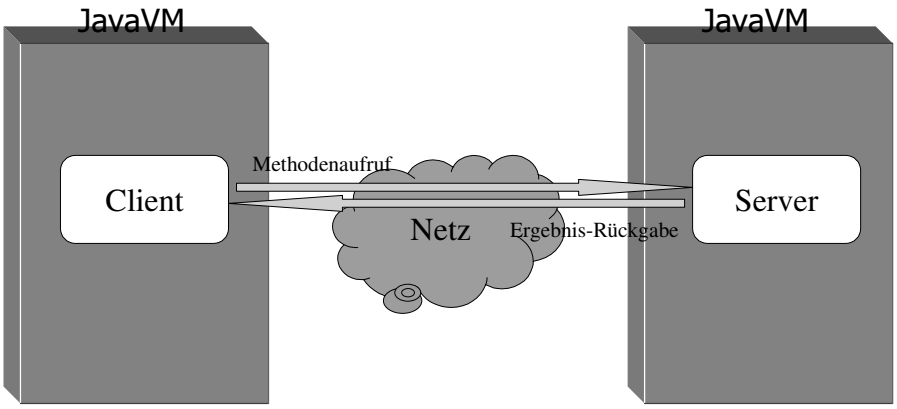
Jini-Programmiermodelle

- Leases
  - Zeitlich begrenzte Nutzung von Diensten mit Hilfe von Time-outs
- Verteilte Events
  - Eine Event-Quelle, viele Event-Consumer
    - Publikation der möglichen Events
    - Subskriptionsphase, dann Lieferungsphase
  - Asynchrone Kommunikation
    - Oftmals durch synchrone Mechanismen (RMI) realisiert
- Verteilte Transaktionen
  - Wie in Datenbanken, aber für den verteilten Fall, d.h. nicht geschlossene Systeme

Remote Method Invocation (RMI)



RMI



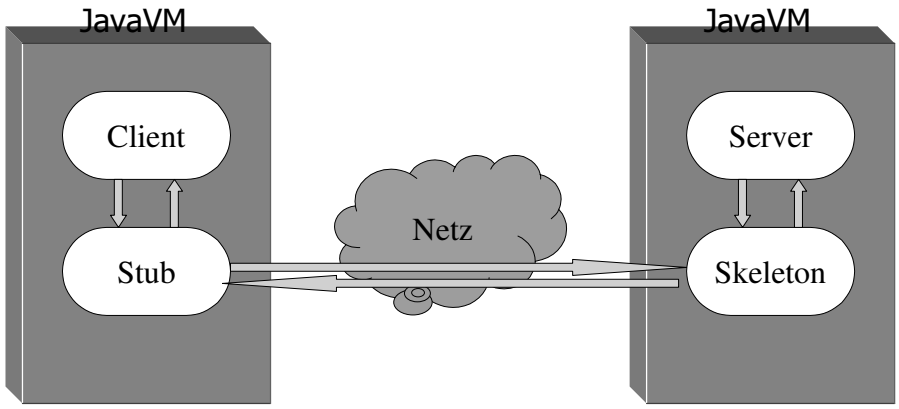
Jini, 39

RMI

- RMI=„Entfernter Methodenaufruf“
- „Klassisches“ Java: Methodenaufrufe innerhalb einer einzelnen Java Virtuellen Maschine (JVM)
- Mit RMI können Objekte in einer JVM Objekte in einer entfernten JVM aufrufen
- JVMs können auf unterschiedlichen Rechnern im Netz ausgeführt werden
- Bekannte Idee: Erzeuge Stellvertreter des entfernten Objektes in lokaler JVM
  - Client-Stub und Server-Skeleton

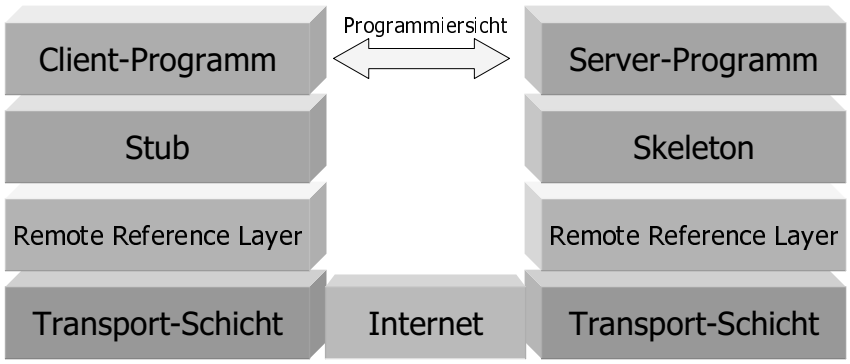
Jini, 40

RMI: Stubs und Skeleton



Jini, 41

Detailierte Darstellung



- „Remote Reference Layer“: Mappt entfernte Referenzen auf Rechnernamen und Objekte

Jini, 42

RMI-Programmierung

- Entfernte Methoden werden durch „remote interfaces“ definiert:

```
import java.rmi.*;

public interface Hello extends Remote {
   public String sayHello() throws java.rmi.RemoteException;
}
```

Methoden können immer `RemoteException` auslösen

- Das entfernte Objekt kann nicht mehr verfügbar sein
- Das Netz kann gestört sein

RMI-Programmierung

- Server-Programme impl. Interface und erweitern i.a. `java.rmi.UnicastRemoteObject`
  - Implementierung der entfernten Klassen
  - `UnicastRemoteObject` stellt die wichtigsten Methoden für die Verwendung von RMI bereit

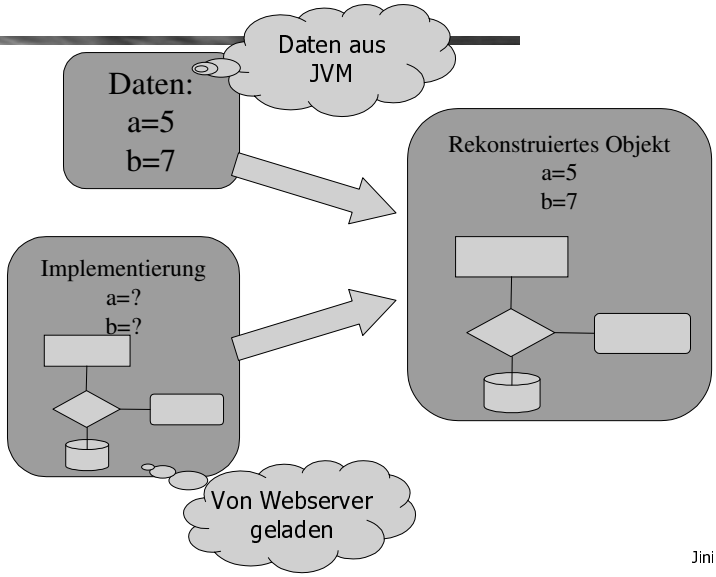
```
import java.rmi.*;
import java.rmi.server.*;
import java.net.*;

public class HelloImpl implements Hello extends UnicastRemoteObject {
   public HelloImpl() throws RemoteException {
     super();
   }
   public String sayHello() throws java.rmi.RemoteException {
     return „hallo“;
   }
}
```

Code-Mobilität

- RMI überträgt Zustand eines Objekts, nicht aber Implementierung der zugehörigen Klasse
- Problem: Klassen sind i.a. beim Client nicht lokal verfügbar (im Klassenpfad aufgelistet)
  - Klassen-Implementierungen müssen zur Laufzeit des Client dynamisch aus dem Netz nachgeladen werden
  - Auf Empfängerseite werden Zustand und Implementierung zu gültigem Objekt zusammengefügt

Jini, 45

Code-Mobilität

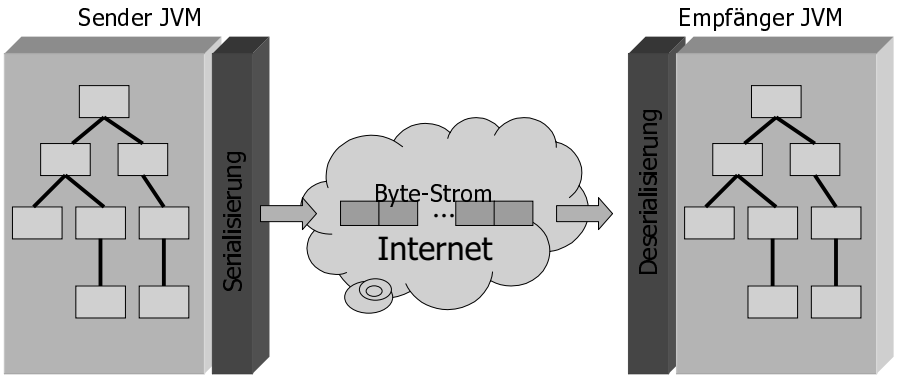


Jini, 46

Serialisierung

- Methoden-Parameter können auf zwei Arten verschickt werden:
  - „Remote object“ wird als „remote reference“ transportiert („verlässt“ JVM nicht)
  - Als Kopie des Objektes, sonst („call by value“)
- Kopien werden zu fremder JVM übertragen
  - Datenstrukturen müssen in Datenstrom umgewandelt werden
  - Transportierter Datenstrom muss beim Empfänger in Datenstruktur verwandelt werden
  - Serialisierung-API in Java

Serialisierung



Serialisierung

- Transformation eines Objektes in einen Bytestrom
  - Objekt besteht aus seinem Zustand und der Implementierung in Form einer `class`-Datei
  - Zustand ist dynamisch und nur zur Laufzeit bekannt
  - Übertrage einen „Schnappschuss“ dieses Zustands
- Problem: Referenzen auf andere Objekte
  - Nicht nur Referenzen auf primitive Typen
  - Alle referenzierten Objekte müssen ebenfalls rekursiv serialisiert werden
  - Bilde transitive Hülle über alle referenzierten Objekte

Serialisierung - Verwendungs-möglichkeiten

- Zustand kann serialisiert in Datei gespeichert werden
  - Konfigurationen
  - (Enterprise) Java Beans
  - Spätere Wiederaufnahme einer Bearbeitung eines Zustands
- Zustand kann serialisiert auf einen „Socket“ geschrieben werden
  - Übertragung des Zustands von einer JVM zu einer anderen

Leases

- Leases sind ein Vertrag zwischen zwei Parteien
- Leases führen eine Art Zeitmodell in Jini ein
  - Zuordnung von Ressourcen an Zeitraum gekoppelt
  - Zugehörigkeit von Objekten über wiederholte Interessenbekundung modelliert
    - „Ich bin weiterhin an X interessiert“
      - Lease wird periodisch erneuert
      - !!! Lease kann auch verweigert werden !!!
    - “Kein Interesse mehr an X”
      - Lease wird nicht erneuert
      - Lease-Herausgeber kann Objekt X anderweitig verwenden
        - » Löschen oder anderem Interessenten geben

Wozu Leases?

- Für Allokation von Hard- und Software
  - Beispiele: Persistenter Speicher, Ein-/Ausgabegeräte
  - Beispiel: Gruppenkommunikation: Teilnahme wird geleased
- Innerhalb von Jini
  - Registrierung für Ereignisse (Event-Notifikationen)
  - Verbreitung von Informationen (LUS, Namensdienst,...)
  - Verteilte “Garbage Collection”
    - Einträge im LUS sind geleased
    - Zuweisung von Ressourcen ist geleased
    - Lease ausgelaufen → “Garbage”
- Abrechnung kostenpflichtiger Dienste (vielleicht...)

Jini: Zusammenfassung

- Die Vision:
  - Alles wird vernetzt sein
  - Alles wird kommunizieren (können)
  - Kommunikation wird nichts oder wenig kosten
  - Mobilität als dominantes Schema
- Das Problem:
  - Die Infrastruktur muss sich an die Devices anpassen und umgekehrt
  - Einbindung kleiner Devices
  - Spontanität als Paradigma
  - Verteiltheit
    - Partielle Fehler
    - Kommunikation über Netze

Jini: Zusammenfassung

- Die Lösung (?)
  - Jini als Infrastruktur für dienstbasierende, ubiquitäre Netze
  - RMI als Abstraktion vom Netz
  - Discovery&Join, Lookup
  - Leases, Verteilte Events, Transaktionen
  - Der Dienst als zentrale Abstraktion

Fazit: Jini

- Richtige Richtung
  - Ubiquitäre Netze
  - Mobilität
- An vielen Stellen gute Ideen
  - Einfachheit
  - Discovery&Join
  - „Weniger ist mehr“ bringt viel Flexibilität
  - Erweiterung des Namensdienstes um beschreibende Attribute
  - Leases, Transaktionen -> immer gleiches Designpattern
- Insgesamt: Alles separat betrachtet nicht neu, das Zusammenspiel bringt aber Neues

Aber...

- Ressourcenverbrauch
  - Ein Dienst ist i.a. eine JVM
- Performanz
  - Java/RMI
- Einbindung kleiner Geräte
  - JVM und RMI auf Gerät benötigt
  - Anpassung an limitierte Verhältnisse nötig (wie?)
  - Proxy-Objekte wandern zum Gerät (Platz...)
- Es gibt konkurrierende Ansätze (z.B. SLP, UPnP)

Probleme

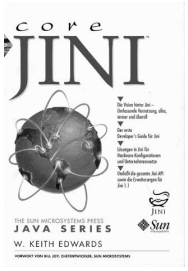
- Sicherheit
  - Gerade in dynamischen Umgebungen wichtig
  - Benutzer braucht Vertraulichkeit
    - Kommunikation
    - Daten
    - E-Commerce
  - Dienste verwenden andere Dienste im Namen des Benutzers
    - Was ist mit kostenpflichtigen Diensten?
- Skalierbarkeit?
  - Skaliert Jini in „globalem“ Massstab?

Informationen zu Jini

- Im Internet
  - Jini Homepage:
    `http://www.sun.com/jini`
  - Jini Community:
    `http://www.jini.org`
- Mailinglisten
  - Englischsprachige Mailingliste bei Sun:
    `JINI-USERS@JAVA.SUN.COM`
    `http://www.jini.org/openforum.html`
  - Deutschsprachige Mailingliste
    `DE-JINI-USERS@informatik.tu-darmstadt.de`
    `http://www.informatik.tu-darmstadt.de/de-jini-users/`

Bücher zu Jini

- W. Keith Edwards: **Core Jini**, Prentice Hall, 1999
  - Gut motiviert, detailiert,
  - Keine Angst vor den ca. 800 Seiten
    - Oft Wiederholungen
    - Langsame Steigerung des Detailierungsgrades
  - Auf deutsch im Markt&Techik Verlag erhältlich
- H. Bader, W. Huber: **Jini**, Addison-Wesley, 1999



Jini, 59