Jini

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

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

Jini

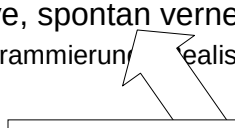
- Infrastruktur („Middleware“) für dynamische, kooperative, spontan ver[illegible]te Systeme
  - Zur Programmierung / Realisie[illegible]verteilter Anwendungen

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

- **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



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

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

Jini

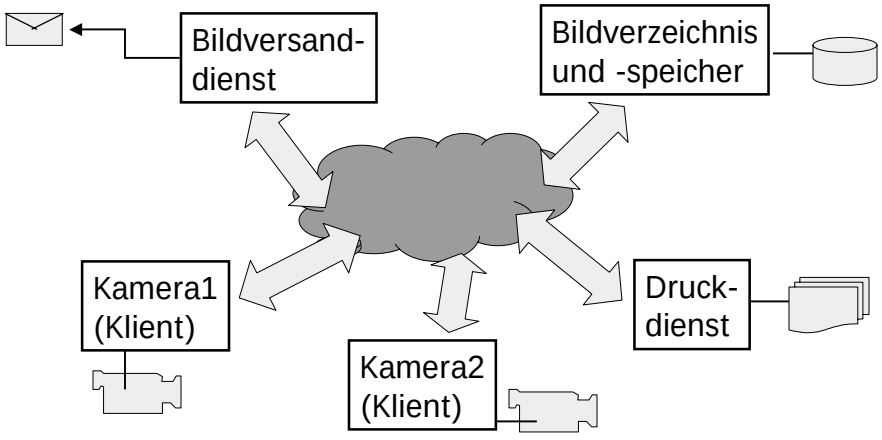
- Infrastruktur („Middleware“) für dynamische, kooperative, spontan vernetzte Systeme
  - Zur Programmierung / Realisierung verteilter Anwendungen
- Auf Java aufbauend und in Java implementiert
  - Typsichere (objektorientierte) Kommunikationsstruktur
  - Nutzt RMI (Remote Method Invocation)
  - „Überall“ 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, 6

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

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

Probleme mit klassischer 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

Trugschlüsse bei verteilten Systemen

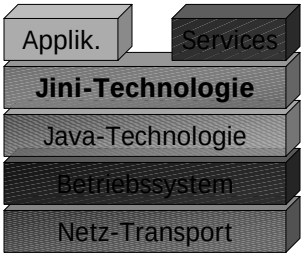
- Die ideale Sicht…
  - Das Netz ist verlässlich
  - Die Latenz ist null
  - Die Bandbreite ist unendlich
  - Das Netz ist sicher
  - Die Topologie ändert sich nie
  - Es gibt einen Systemverwalter

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



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?

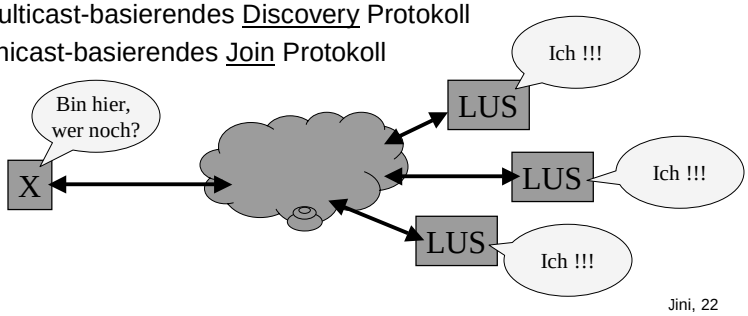
können Basisprotokoll für Discovery und Join; besitzen eine JVM; ...

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

Jini-Infrastruktur

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, 22

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 A-priori-Wissen über lokale Umgebung
- Wie bekommen neue Dienstleister und Dienstnutzer Kenntnis von den lokalen Gegebenheiten?

Jini, 23

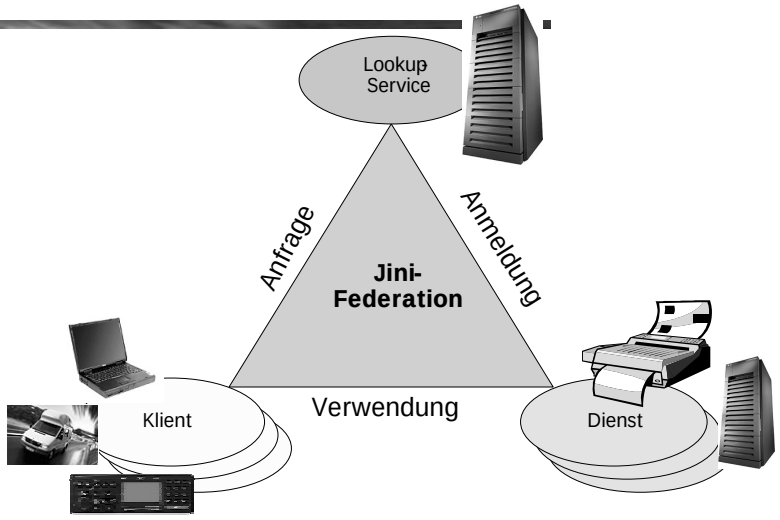
Lookup-Service (1)

- Ähnlich CORBA Naming Service und RMI Registry
- Repository für Dienstanbieter
- Kernelement jeder Jini-Föderation
- 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)

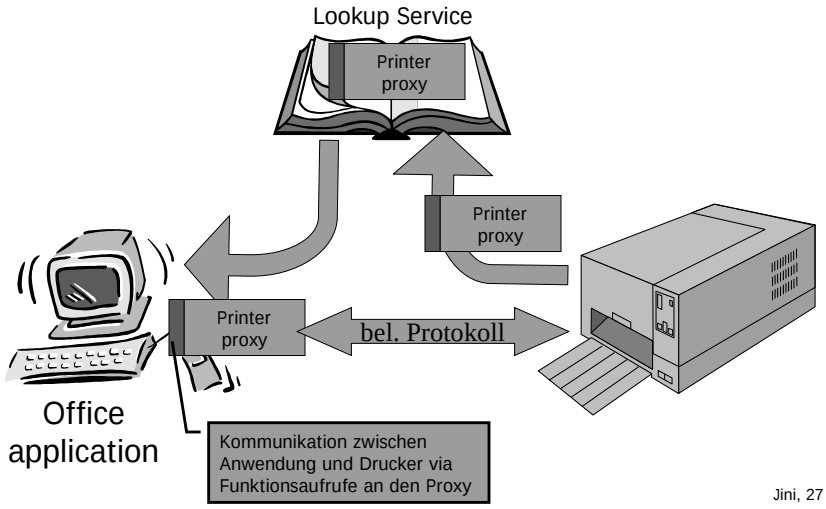
- Jini-Systeme gruppieren sich um einen oder mehrere Lookup-Service(s)
- Dienste registrieren Dienstleistung und Fähigkeiten beim Lookup-Service (Registrierung)
- Klienten finden Dienste über Lookup-Service („Lookup“)
- Weitere Möglichkeiten:
  - Erhöhung der Robustheit durch redundante Auslegung der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

LUS (3)



Jini, 26

Beispiel



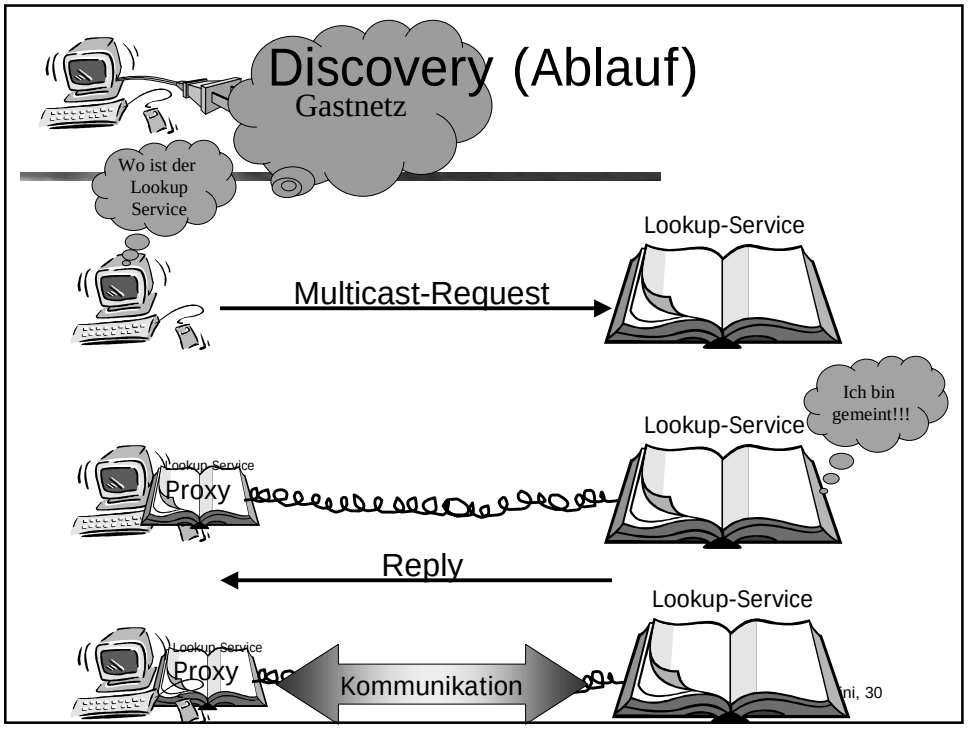
Jini, 27

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
  - Bekannt machen („Registrieren“) des eigenen Dienstes
  - Verwenden („Suchen“) eines vorhandenen Dienstes
- Discovery-Protokoll:
  - Multicast an bekannte Adresse („well-known address / port“)
  - 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

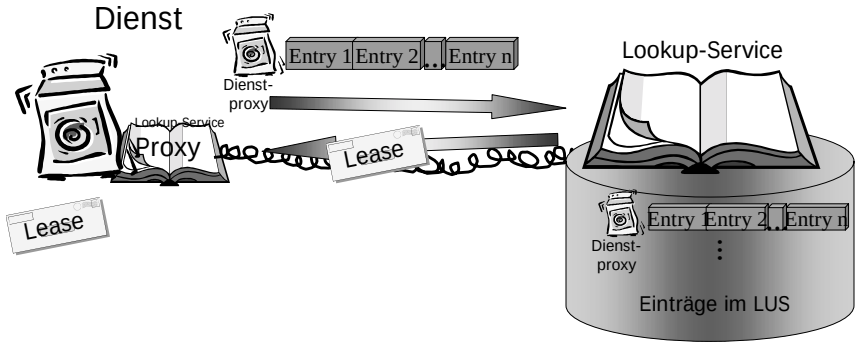
- Benötigt keine Kenntnisse über das Gastnetz
- Aktive Suche nach Lookup-Service(s)
- Discovery-Request basiert in IP-Netzen auf Multicast-UDP
  - „Empfohlen“ eine TTL von 15
  - Multicast-Gruppe für Discovery-Requests ist 224.0.1.85
  - Port-Nummer der Lookup-Services ist normalerweise 4160
- Discovery-Reply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

Jini, 31

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)



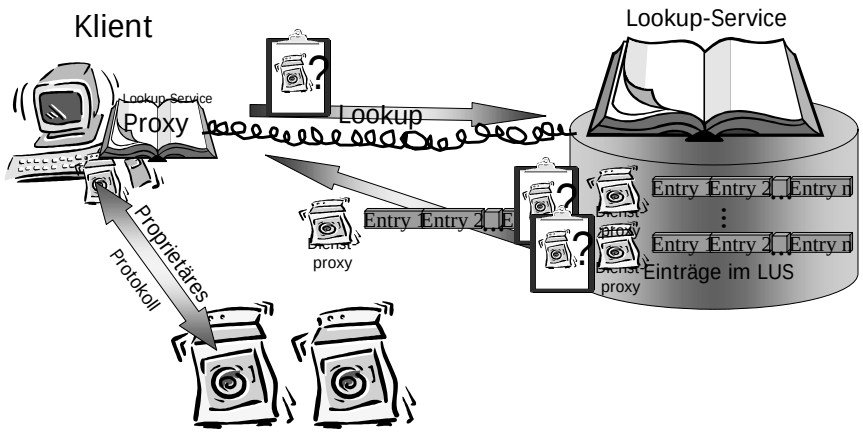
Join: Weitere Eigenschaften

- 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

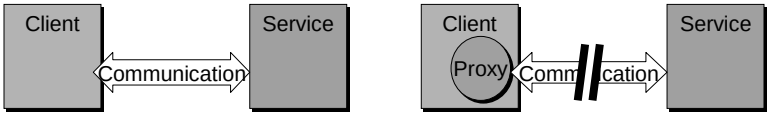
- Klient kennt Lookup-Service (z.B. über Discovery-Protokoll)
- Sucht bestimmten Dienst
- Formuliert dazu Anfrage an den Lookup-Service
  - Mittels eines „Service-Template“
  - Matching nach Registrier-Nummer des Dienstes und / oder Typ und / oder Attributen
  - Wildcards erlaubt, ansonsten „exact-match“
- 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)



Proxies und Smart-Proxies

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



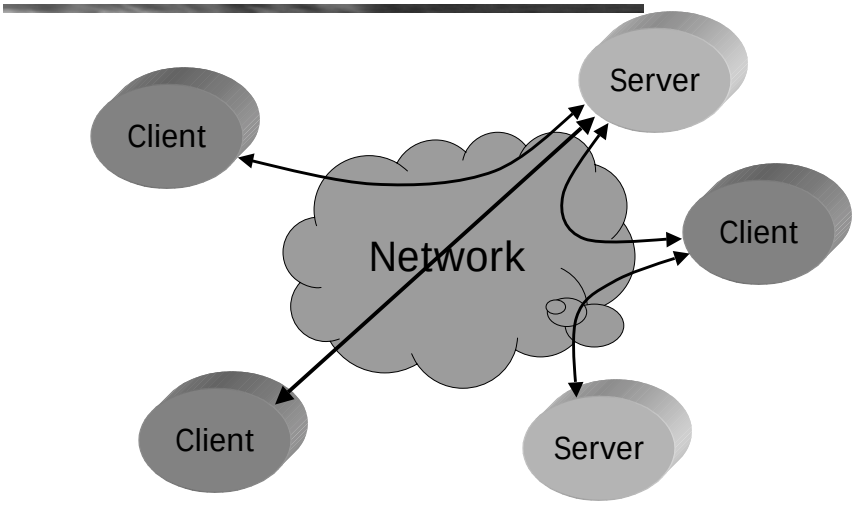
Gruppen

- 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 Textbezeichner
- 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-Programmiermodell

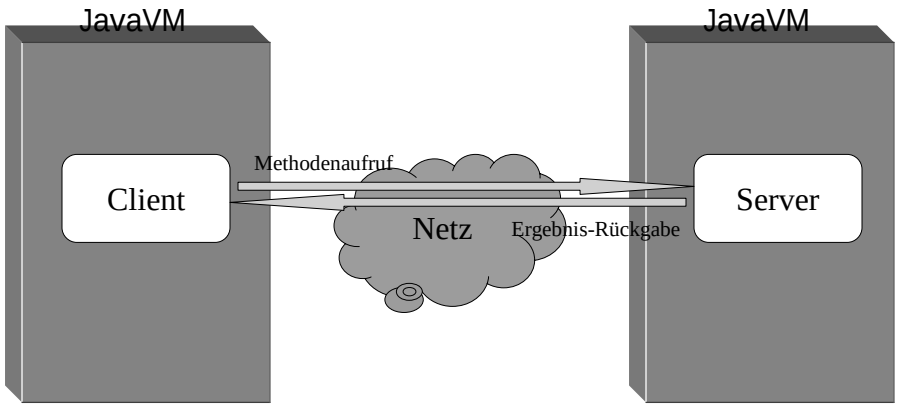
- Remote Method Invocation (RMI)
  - Entfernter Methodenaufruf
- Leases
  - Zeitlich begrenzte Nutzung von Diensten mit Hilfe von Time-outs
- Verteilte Events
  - Eine Event-Quelle, viele Event-Consumer
    - Publikation der möglichen Events
    - Subskriptionsphase, dann Lieferungsphase
  - Asynchrone Kommunikation
- Verteilte Transaktionen
  - Wie in Datenbanken, aber für den verteilten Fall
  - Allerdings keine mächtige Semantik auf Jini-Ebene

Remote Method Invocation (RMI)



Jini, 42

RMI



Jini, 43

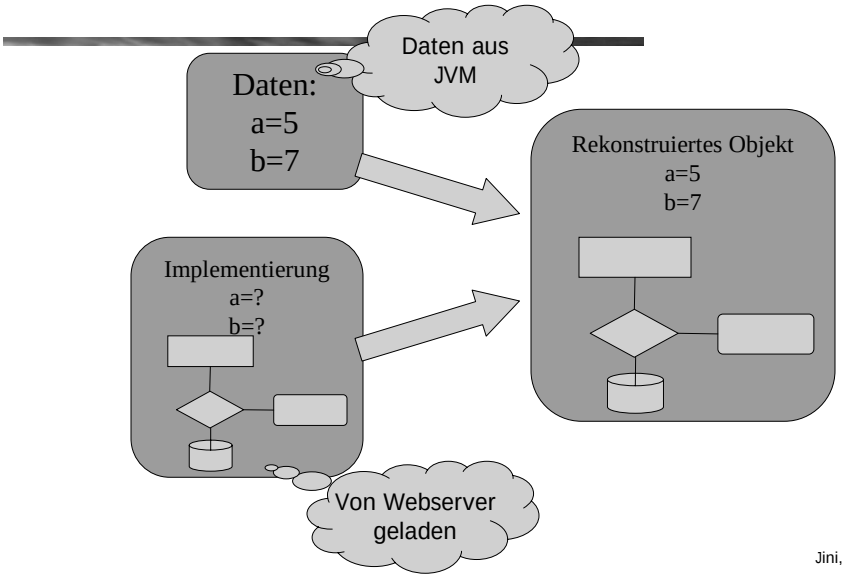
RMI

- 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
- Klassische Idee: Erzeuge Stellvertreter des entfernten Objektes in lokaler JVM
  - Client-Stub und Server-Skeleton

Code-Mobilität

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

Code-Mobilität



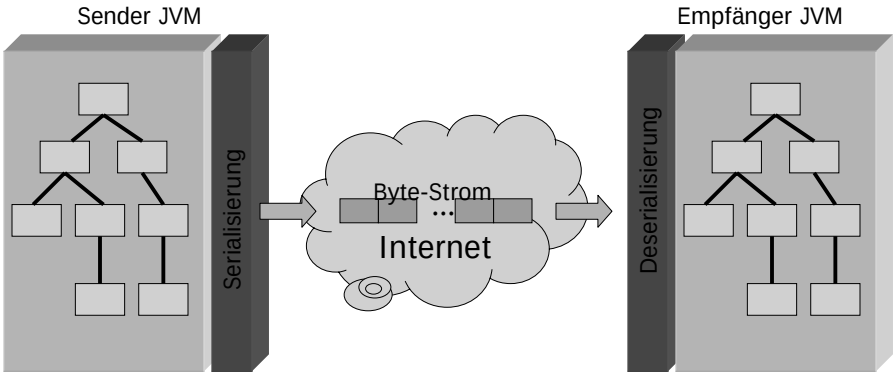
Jini, 50

Serialisierung

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

Jini, 51

Serialisierung



Jini, 52

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)

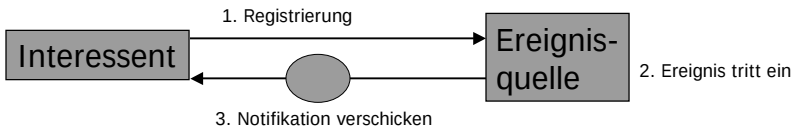
Jini, 55

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,...)
    - Einträge im LUS sind geleased
  - Verteilte “Garbage Collection”
    - Zuweisung von Ressourcen ist geleased
    - Lease ausgelaufen → “Garbage”
- Damit auch Abrechnung kostenpflichtiger Dienste ?

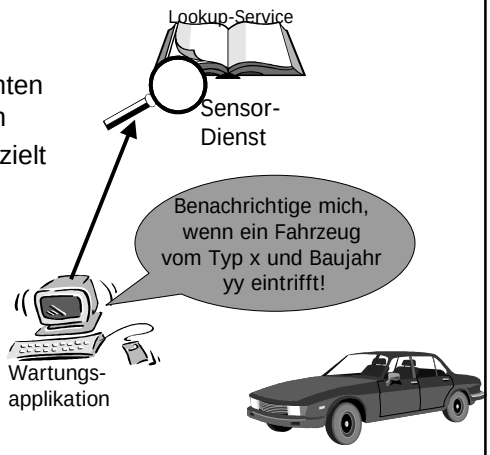
Verteilte Ereignisse

- Erlaubt es einem Objekt in einer Java-VM, Interesse an Ereignissen eines Objektes in einer anderen JVM zu registrieren
- „Publisher / Subscriber“ Modell
- Allgemeiner Ablauf:



Verteilte Ereignisse - Beispielszenario Autowerkstatt

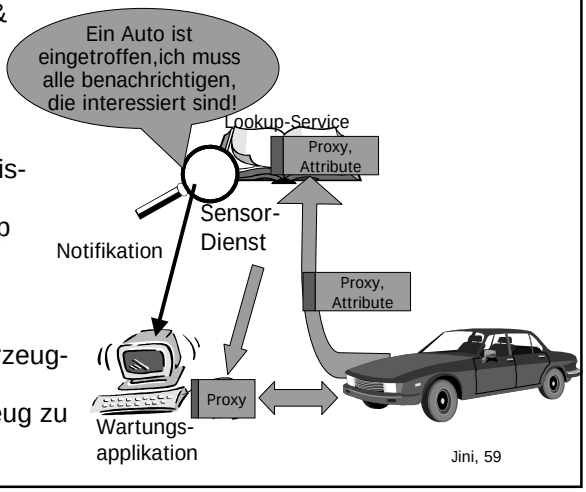
- Applikation registriert Interesse an Events, die das Eintreffen eines bestimmten Typs von Fahrzeug anzeigen
- Sensor-Dienst überwacht gezielt den Lookup-Service
- Benachrichtigt Applikation, wenn passendes Fahrzeug eintrifft
- Modellierung durch verteilte Ereignisse



Jini, 58

Verteilte Ereignisse - Beispielszenario

- Fahrzeug trifft ein, registriert sich in der Werkstatt
  - Es führt Discovery & Join aus
  - Sensor registriert neues Fahrzeug im Lookup-Service
  - Überprüft, ob Ereignis-Registrierungen für diesen Fahrzeug-Typ vorliegen
  - Benachrichtigt alle Interessenten
  - Applikation lädt Fahrzeug-Proxy zu sich und greift auf das Fahrzeug zu



Jini, 59

Jini: Zusammenfassung

- Die Vision:
  - Alles wird vernetzt sein
  - Alles wird kommunizieren (können)
  - Kommunikation wird kaum etwas kosten
  - Mobilität als dominantes Schema
- Das Problem:
  - Die Infrastruktur muss sich an die Devices anpassen und umgekehrt
  - Einbindung kleinster Devices
  - Spontaneität als Paradigma
  - Partielle Fehler durch Verteiltheit möglich

Jini: Zusammenfassung

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

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
- Skalierbarkeit?
  - Skaliert Jini in „globalem“ Massstab?

Jini-Konkurrenten

- Neben Jini existieren einige ähnliche Technologien und Systeme (vor allem zur Dienstvermittlung)
- Unterschiede in
  - Dienstbeschreibungen
    - z.B. Strings, XML-Dokumente, Java-Typen, UUID,...
  - Dienstvermittlung
    - zentral, directory/lookup service, dezentral
  - Abstraktionsebene
- Beispiele
  - Service Location Protocol (SLP)
  - Universal Plug & Play (UPnP)
  - Bluetooth (Service Discovery Protocol – SDP)
  - Home Audio Video Interface (HAVi)
  - Salutation
  - HP e-speak, HP Chai

Universal Plug and Play (UPnP)

- Initiatoren: Microsoft und Hewlett-Packard
- Dienstvermittlungsarchitektur insbesondere für den Home/Office-Bereich
- Ziele:
  - Plug und Play von Geräten in einem Netzwerk
  - Möglichkeit zur Fernsteuerung dieser Geräte
- UPnP nutzt Standard-Internet-Protokolle zur Interaktion
  - HTTP-analog, TCP, XML