# Konzepte von Betriebssystem-Komponenten

# Jini

Philipp Sommer

09.11.2004

## 1. Einführung

### 1.1. Hintergrund

Da 1994 der erste, vollständig in Java geschriebene Webbrowser die Fähigkeit des Herunterladens ausführbarer Programme, den Applets, besaß, machten sich die Entwickler bei Sun Gedanken darüber, wie man mit geringem Verwaltungsaufwand das Austauschen von Code zwischen zusammenarbeitenden Gemeinschaften von Geräten ermöglichen kann. Dabei wollte man sich nicht nur auf Computersysteme beschränken, sondern allen Geräten (sei es eine Mikrowelle, ein Videorekorder, oder ein Toaster), die einen Prozessor und Speicher besitzen, die Möglichkeit geben an Netzwerkgemeinschaften teilzunehmen. Auch auf Java basierend erhielt das Projekt die Bezeichnung **Jini** (Java Intelligent Network Interface oder aber Jini Is Not Initials(Jini ist keine Abkürzung)).

### 1.2. Was ist Jini?

Jini ist also eine auf Java basierende Architektur, mit deren Hilfe sich **Dienste** im Netzwerk finden, miteinander kommunizieren und sich gegenseitig nutzen können.
Beispielsweise könnte eine Jini-fähige Kamera, die an ein Netzwerk angeschlossen wird, das über einen Jini-fähigen Drucker verfügt, diesen finden und nutzen, egal in welchen lokalen System sich die Geräte befinden.

### 1.3. Ziele von Jini

- Einfachheit: Jini nutzt Java und fügt selbst nur wenige Funktionen hinzu, um Geräten und Diensten die Zusammenarbeit zu erleichtern.
- Zuverlässigkeit: Beim Hinzufügen oder Entfernen von Dienste in einem Netzwerk („spontane Vernetzung“), soll dieses stabil bleiben.
- Skalierbarkeit: Jini-Dienste schließen sich zu einer Gemeinschaft (Djinn) zusammen, in der jeder Dienst den anderen kennt und nutzen kann. Jini besitzt die Fähigkeit auf Dienste anderer Gemeinschaften zuzugreifen, indem die Gemeinschaften zu größeren Einheiten föderiert werden.

# 2. Jini Technology

## 2.1. Komponenten in einer Jini-Architektur

In der Jini Architektur gibt es drei Hauptkomponenten. Die erste ist der **Dienst**, z.B. ein Drucker, der eine spezielle Funktion allen Teilnehmern im Netz zur Verfügung stellen will. Die zweite Komponente ist der **Client** (Dienstnutzer), welcher einen oder mehrere Dienste eines Dienstanbieters nutzt. Die dritte und wohl wichtigste Komponente ist der **Lookup-Service** (Service-Locator), der die Aufgabe hat, zwischen Dienst und Client zu vermitteln. Diese drei Komponenten sind Teil eines Netzwerkes, das auf TCP/IP läuft (Abb. 1).

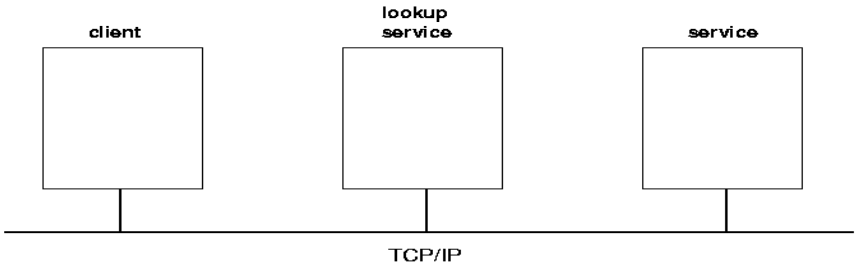


Abb.1

Zwischen diesen drei Komponenten wird nun Code ausgetauscht, indem die Objekte so serialisiert werden (siehe Vortrag über RMI), dass sie versendet werden können. Um später die Objekte verwenden zu können müssen sie zunächst deserialisiert werden. Nun kann es vorkommen, dass Objekte einer JVM (Java Virtual Machine), Methoden eines Objekts in einer anderen JVM aufrufen müssen. Das geschieht meistens mit RMI.
Die Jini-Protokollschichten (vgl. Abb.2) zeigen, dass Jini darauf hinzielt Komponenten eines Netzwerkes soweit zu abstrahieren, dass sie sich auf einer gemeinsamen Kommunikationsebene befinden.

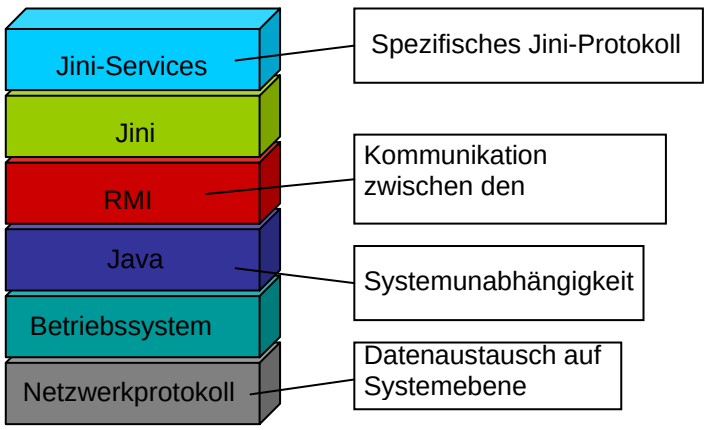


Abb. 2 Jini-Protokollschichten

## 2.2. Client , Server und Dienste

Jini ist ein Framework für verteilte Computersysteme. Die Teilnehmer heißen daher Clients und Server. Ein Server hat jedoch ein Interface, das er der Außenwelt

präsentiert. Dieses Interface nennt man auch Dienst-Interface, oder auch einfacher: **Dienst**.
Fast alle Jini-Dienste sind jedoch eine Mischform aus Client und Server, da sie einerseits ihren Dienst als Server anbieten, andererseits den Lookup-Service als Client nutzen. Lediglich der Lookup-Service ist ein reiner Server.
Damit ein Client einen Dienst nutzen kann muss der Server ein **Service-Objekt** erzeugen, welches den Dienst implementiert (vgl. Abb.3), und dieses beim Lookup-Service *registrieren*. Dieses Service-Objekt kann mit verschiedenen Attributen versehen werden, die den Dienst genauer beschreiben.
Beispielsweise kann ein Druckdienst, welcher u.a. die print()-Methode enthält, von mehreren Servern angeboten werden:

- Von einem Drucker, bei dem der Druckdienst direkt auf dem Drucker läuft.
- Von einem auf software-basierenden Dienst, der mit Hilfe der print()-Methode etwas auf dem Monitor ausgibt.
- Von einem Proxy, der einer Jini Gemeinschaft das Interface des Druckdienstes präsentiert, jedoch mit dem Drucker in einer anderen JVM über ein proprietäres Protokoll kommuniziert.

Das erzeugte Service-Objekt ist für alle Clients im Lookup-Service sichtbar, und wird von ihnen zur Nutzung des Dienstes heruntergeladen.

Abb.3: Dienstanbieter mit erzeugtem Service-Objekt

## 2.3. Discovery und Join

Bevor ein Server einen Dienst beim Lookup-Service registrieren kann, muss er zunächst den Lookup-Service finden (vgl. Abb. 4). Dieser Vorgang wird als **Discovery** bezeichnet und ist in der *Discovery and Join Specitfication* definiert. Sobald eine Jini-Gemeinschaft gefunden wurde, kann die Jini-fähige Einheit durch **Joining** der Gemeinschaft beitreten und ihre Dienste anbieten. Zwischen Gemeinschaften und Lookup-Diensten besteht in Jini grundsätzlich nicht ein Eins-zu-Eins-Verhältnis.
Es ist durchaus möglich, dass ein Lookup-Service mehreren Gemeinschaften dient. Lookup-Dienste müssen explizit von einem Administrator gestartet werden, damit sich Jini-Gemeinschaften überhaupt aufbauen können.
Discovery basiert auf drei, für jeweils eine bestimmte Situation passende Protokolle:

- **Unicast Request Protocol** wird verwendet, wenn der Rechner, auf dem der Lookup-Service läuft, bereits bekannt ist. Mit Hilfe von Unicast Discovery kann eine Verbindung zu einem Lookup-Service außerhalb des lokalen Netzwerkes hergestellt werden. Dies geschieht mit Hilfe von TCP/IP.
- **Multicast Request Protocol** wird verwendet, wenn kein Lookup-Service im Netzwerk bekannt ist. Dabei werden UDP Datagramme versendet, um benachbarte Lookup-Dienste aufzuspüren.
- **Multicast Announcement Protocol** wird von den Lookup-Diensten zur Ankündigung des eigenen Vorhandenseins benutzt. Sobald ein neuer Lookup-Service in einer Jini-Gemeinschaft gestartet wird, wird jeder

interessierte Beteiligte darüber über UDP informiert. Dieses Protokoll wird hauptsächlich verwendet, damit Lookup-Services ihre (erneute) Erreichbarkeit nach einem Ausfall oder Neustart (z.B. Serverabsturz) den Teilnehmern einer Jini-Gemeinschaft mitteilen können.

Alle Lookup-Services lauschen standardmäßig an Port 4160. Wenn der Lookup-Service eine Anfrage an diesem Port bekommt, sendet er ein *registrar*-Objekt an den Server zurück (vgl. Abb. 5), das diesem als Proxy zum Lookup-Service dient. Dadurch kann der Dienstanbieter sein Service-Objekt auf den Lookup-Service laden und ist somit bei ihm registriert (vgl. Abb. 6).

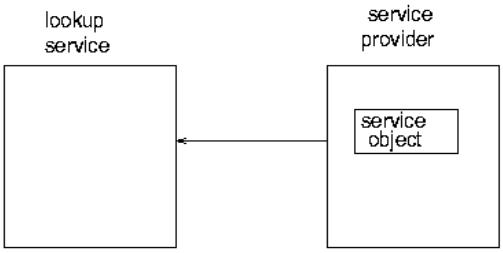


Abb. 4 Suche nach Lookup-Service via Unicast oder Multicast.

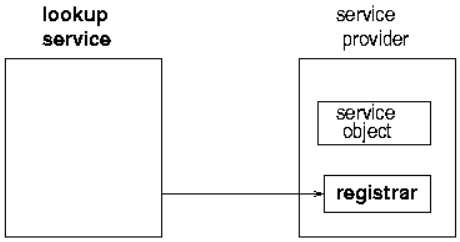


Abb. 5: Senden des registrar-Objekts

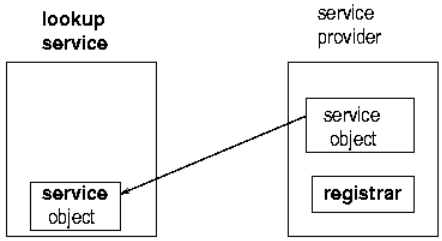


Abb. 6: Hochladen des Service-Objekts

## 2.4. Client Lookup

Der Client versucht, um einen Dienst nutzen zu können, eine Kopie des erwünschten Service-Objekts in seine JVM zu bekommen. Dabei durchläuft er für die Registrierung beim Lookup-Service dieselben Schritte wie der Server (vgl. Abb. 7 und 8). Nach erfolgreicher Registrierung stellt der Client eine Suchanfrage nach einem bestimmten Dienst (dazu können die Attribute zur genaueren Spezifikation genutzt werden). Hat der Lookup-Service einen passenden gefunden hat, kann der Client das Service-Objekt in seine JVM herunterladen (vgl. Abb. 9).

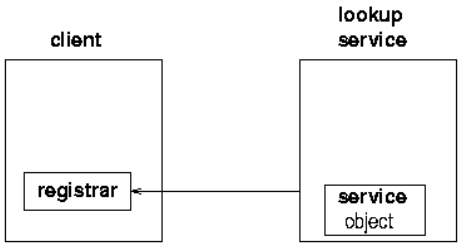


Abb. 7: Client erhält registrar-Objekt

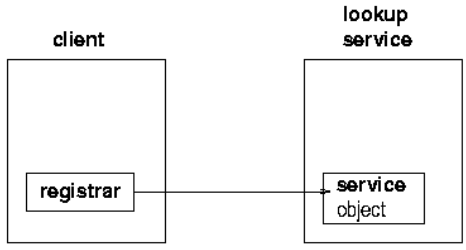


Abb. 8: Client registriert sich

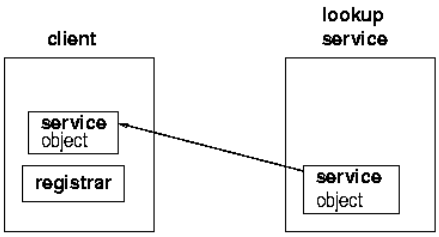

Abb. 9:Client lädt Service-Objekt herunter

Zu diesem Zeitpunkt hat der Dienstanbieter sein Service-Objekt, der Lookup-Service und der Client eine Kopie dieses Objekts in der jeweiligen JVM. Der Client kann nun direkt mit dem Service über das Service-Objekt kommunizieren.

## 2.5. Proxies

Das Service-Objekt dient dem Client also als **Proxy** zum Service und ist daher auch ein echter Proxy. Nun muss zwischen verschiedenen Situationen unterschieden werden.

- Der Proxy ist der Service. Der Service wird dann komplett auf dem Client ausgeführt. Der Service ist nur dafür da, um sich zu registrieren und seinen Eintrag im Lookup-Service zu erneuern.

Wenn der Dienstanbieter jedoch eine Hardware ist, und das Service-Objekt in einer entfernten JVM läuft, muss noch ein zweites Objekt im Dienstanbieter laufen.

- Der Proxy ist dann nur das Interface zum Service. Der Service führt alle Prozesse aus und der Proxy dient nur dazu, die Kommunikation zwischen Client und Service zu ermöglichen (vgl. Abb. 10). Der Lookup-Service wird von da an zur Kommunikation zwischen Client und Server nicht mehr gebraucht.

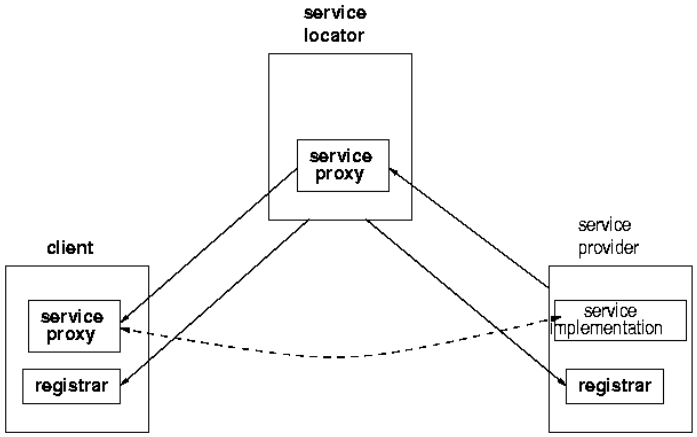

Abb. 10: Kommunikation über das Proxy-Objekt

## 2.6 Leasing

Ein großer Nachteil von verteilten gegenüber nicht-verteilten Systemen ist die Tatsache, dass einzelne Dienste ausfallen können und dadurch die Kommunikation zwischen den Komponenten unterbrochen ist. Der Lookup-Service erfährt den

Ausfall einer Komponente nicht und stellt den Clients weiterhin deren Service-Objekt zum download bereit. Versucht der Client darauf den nicht vorhandenen Dienst zu nutzen, tritt ein Fehler auf.
Um einer solchen Situation vorzubeugen, muss ein Dienst seine Verbindung zum Lookup-Service aktiv verwalten. Jeder Dienst erhält bei der Registrierung beim Lookup-Service von diesem ein **Lease**, das er vor dessen Ablauf erneuern muss. Andernfalls nimmt der Lookup-Service an, dass der Dienst nicht mehr existiert, und löscht dessen Service-Objekt aus seiner Liste der verfügbaren Dienste. Dabei kann der Dienst absichtlich beendet oder entfernt worden sein, er kann aber auch ausgefallen sein oder ein Netzwerkproblem unterbindet die Kommunikation zwischen Dienst und Lookup-Service.
Die Jini-API stellt zur Vereinfachung und zur Entlastung der Dienste einen Lease-Manager zur Verfügung, der alle ihm zugeteilten Leases in den notwendigen Zeitintervallen erneuert. Mißglückt die Erneuerung, oder läuft das Lease trotzdem ab wird ein Event generiert, das dem Dienst bescheid gibt.
Der Zeitpunkt, an dem das Lease abläuft wird nicht in absoluter (Sonntag 10.00), sondern in relativer Zeit (5 Minuten) angegeben, um Konflikte mit nicht synchron laufenden Systemuhren zu vermeiden.
Durch das Prinzip des Leasings wird also gewährleistet, dass keine „toten“ Dienste existieren und das Gesamtsystem wird dank dieser sich automatischen anpassenden Umgebung stabiler.

# 3 Abschließende Bewertung

Jini ist bei weitem nicht die einzige Architektur, die die Vernetzung von Komponenten in einem Netzwerk verbessern und vereinfachen soll. Jedoch bietet Jini gegenüber anderer Architekturen einige Vorteile:

- Jini ist eine elegante Architektur für spontane Netzwerke;
- Für Jini-Dienste müssen keine Treiber installiert werden;
- Jini ist ein „selbstheilendes“ System, da nicht verfügbare Dienste gelöscht werden.

An dieser Stelle sollen einige der Konkurrenzprodukte zu SUN’s Jini aufgeführt werden.

- Universal Plug and Play (UPnP von Microsoft) hat das gleiche Ziel wie Jini, ist aber plattformabhängig.
- T Spaces (IBM): Javabasiertes System zum Nachrichtenaustausch
- CORBA (Common Object Request Broker Architekture) ist eine von Java unabhängige Architektur, die auch eine Suche nach Interfaces im Netzwerk ermöglicht.

# 4 Literatur


- W. Keith Edwards, Core Jini; Verlag: Prentice Hall 2000
  ISBN 3-8272-9592-0

- Scott Oaks und Henry Wong, Jini in a Nutshell; Verlag: O’Reilly 2001
  ISBN 3-89721-194-7

- http://www.htw-saarland.de/fb/gis/people/wpauly/vortraege_internet/ws99_00/JINI/vortrag.html
- http://www.cs.bgu.ac.il/~elhadad/se021.html
- http://pandonia.canberra.edu.au/java/jini/tutorial/Jini.xml