Jini: Infrastruktur für dynamische Dienste in verteilten Systemen[1]

Joachim Posegga Deutsche Telekom AG, Technologiezentrum, D-64307 Darmstadt, Tel. 06151/83-7881, Joachim.Posegga@Telekom.de

Zieht man einen Vergleich zwischen dem Internet und dem GSM-Mobilfunk, wird ein Problem klar, das im Internet bisher nur wenig Beachtung fand, nämlich die problemlose Nutzung von Diensten in fremden Netzen: Für GSM gib es klar definierte Protokolle, die es jedem Nutzer erlauben, sich in ein beliebiges GSM-Netz einzuklinken und dort Dienste zu nutzen (sog. „Roaming").

Von dieser Situation ist das Internet in der Praxis noch weit entfernt: Die Aufgabe, einen Laptop in ein lokales IP-Netz (beispielsweise in einem Besprechungsraum) einzubinden und z.B. einen dort vorhandenen Drucker zu nutzen, ist ohne detaillierte Systemkenntnisse kaum lösbar.

Jini, eine auf Java aufbauende Infrastruktur für verteilte Systeme, soll hier Abhilfe schaffen und verspricht, Internet-Dienste genauso einfach handhabbar zu machen wie das Telefonieren: Dazu bietet Jini verschiedene Protokolle, die es erlauben, Netzkomponenten auf einfache Weise in ein IP-Netz einzubinden, die Schnittstellen dieser Komponenten im Netz bekannt zu machen, zu finden und zu nutzen. Sun bezeichnet dieses Prinzip mit „Network-Dialtone".

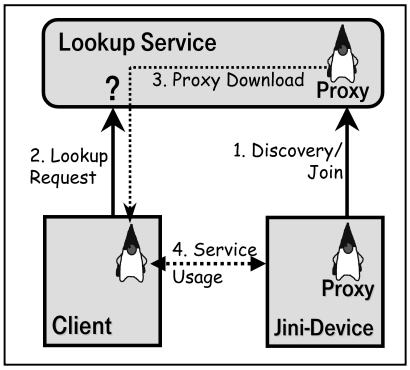
Funktion eines Jini-Netzes

Eine zentrale Komponente bei Jini ist der ***Lookup-Service*** [1], der Informationen über die in einem Jini-Netz vorhandenen Komponenten und Dienste (dem sog. „djinn") verwaltet. Betrachten wir an einem Beispiel wie ein Jini-fähiges Gerät (ein sog. „Jini-Device"), etwa ein Drucker, neu in ein Jini-Netz eingebunden und bekannt gemacht wird (vgl. Abbildung):

1. Das ***Discovery/Join-Potokoll*** [1] sorgt zunächst für die Einbindung des Gerätes, bzw. des Dienstes in den djinn: wird ein Jini-fähiges Gerät physikalisch mit dem Netz verbunden, so sendet dieses typischerweise als erstes ein sog. „multicast Dis-

[1] Ich danke Friedemann Mattern und Marco Gruteser für hilfreiche Kommentare zu diesem Beitrag.

covery“-Paket (quasi einen Rundruf ins Netz), um einen Lookup-Service zu finden. Dieser meldet sich daraufhin bei dem Gerät. Danach teilt das Gerät dem Lookup-Service Informationen über den bereitgestellten Dienst mit und sendet insbesondere ein Java-Applet (sog. „Proxy“) an den Lookup-Service, das später die Kommunikation mit dem Gerät selbst implementieren wird.[2] Damit sind die Dienste, die das Gerät anbietet, im Jini-Netz verfügbar.



2. Zur Nutzung von Diensten können Jini-Teilnehmer über einen sog. ***Lookup-Request*** Jini-Dienste nachfragen: Dabei erhält der Lookup-Service eine Spezifikation in Form eines Java-Interfaces und sucht nach passenden Kandidaten.

3. Konnte der Lookup-Service einen geeigneten Dienst identifizieren, so wird der zugehörige Proxy an den nachfragenden Client geschickt. Konkret ist ein solcher Proxy ein serialisierbares Java-Objekt.

4. Der Client startet den Proxy dann in der eigenen virtuellen Java-Maschine und erhält über ein Interface des Proxies Zugriff auf den Dienst.

Interessant dabei ist insbesondere die automatische Übertragung des Dienste-Proxies zu dem Client: Zum einen kann man so die Installation von Plattform-spezifischer Software auf einem Client vermeiden (z.B. die eines Druckertreibers), zum anderen kann ein Jini-Device damit berechnungsintensive Vorgänge (z.B. die Interpretation von Postscript) auf den Client auslagern.

Jini zielt jedoch nicht nur auf „klassische“ Client/Server-Szenarien, sondern bietet auch Konzepte zur Realisierung verteilter, objektorientierter Systeme:

Mit dem ***Distributed Leasing***-Konzept kann der Zugriff auf Ressourcen verwaltet werden, ***Distributed Transactions*** dienen dazu, verteilte, aber zusammengehörende Schritte atomar zu gestalten und so die Konsistenz in einem System sicherzustellen und durch ***Distributed Events*** läßt sich das Verhalten von verteilten Objekte koordinieren, also lokalen Zusandsänderungen auch globale Auswirkungen geben. Letztlich sind die ***JavaSpaces*** ein Dienst, der auf Linda Tuple Spaces [3] beruht und Java-Objekte verwaltet; damit ist es möglich, Anwendungen mit Hilfe von persistenten, verteilten Objekte zu realisieren.

Ausblick

Jini ist eine interessante Technologie, verspricht sie doch eine wesentliche Vereinfachung bei der Nutzung Internet-basierter Dienste und der Realisierung verteilter Systeme in der Praxis. Eine solche Entwicklung ist, angesichts der sich abzeichnenden Entwicklung im Bereich kleiner, spezialisierter Internet-Endgeräte (PDAs, Personal Appliances, etc). unzweifelhaft notwendig.

---

[2] Im Falle eines Druckers würde dieser Proxy u.a. den Druckertreiber realisieren.

Jini ist nicht *die* Lösung für das anvisierte Szenario, es ist lediglich *eine* denkbare Variante, nämlich eine Java-zentrierte Lösung: das Konzept fußt auf dem Java-spezifischen Kommunikationskonzept RMI („Remote Method Invocation“) [3] und serialisierbaren Java-Objekten; jede Jini-Komponente, die einen Dienst nutzen möchte, muß eine eigene Java-Umgebung (Java-VM und entsprechende Laufzeit-bibliotheken) besitzen. Jini-Devices, die lediglich Dienste anbieten, können zwar etwas einfacher gestaltet werden (vgl. *Jini Device Architecture Specification* [1]), müssen aber in der Lage sein, an den Jini-Protokollen zu partizipieren und das CORBA-Protokoll IIOP [3] beherrschen.

In mancher Hinsicht versucht Jini das Rad, neu zu erfinden: Etwas mehr Offenheit zu CORBA und den CORBAservices [3], die sehr ähnliche Lösungen bieten, oder auch TINA [4] wäre sicher wünschenswert.

Aus technologischer Sicht baut Jini auf weitgehend bekannten Konzepten auf, das Gesamtkonzept läßt jedoch noch einige Fragen offen: unklar bleibt beispielsweise, inwieweit Java Tuple-Spaces skalierbar sind, wie das (statische) Sicherheitskonzept von Java mit der dynamischen Jini-Welt zusammenpaßt, wie Key-Management konkret funktionieren soll, usw.

Der Jini-Geist erfüllt zwar noch nicht alle Wünsche, weist aber hinsichtlich einer Infrastruktur für dynamisch vernetzte Internet-basierter Geräte in die richtige Richtung. Insofern ist die Entwicklung vielversprechend und spannend.

[1] Sun Microsystems, Inc. (1998). *Jini-Spezifikationen, Rev. 1.0. beta* www.javasoft.com/products/jini/specs

[2] Gelernter, David (1985). *Generative Communication in Linda*. ACM TOPLAS, 7(1).

[3] Curtis, David (1997). *Java, RMI and CORBA*. Object Management Group, www.omg.org/news/wpjava.htm

[4] Lill Kristiansen, Ed. (1997): *TINA C Deliverable: Service Architecture*. www.tinac.com