Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Institut für Praktische Informatik und Medieninformatik

Hauptseminar SWTuPS
„Advanced Java Technologies -
Systemprogrammierung”

JINI

Bearbeiter: Martin Heise
Ingenieurinformatik M97

Betreuer: Dr.-Ing. Peter Jackisch
Dr.-Ing. Günter Hübel

Gliederung

1. Einführung und Motivation 3

2. Beispiele 4

3. Geschichte 5

4. Softwarelizenz 5

5. Aufbau 6

6. Basiskonzepte 6
6.1. Discovery 7
6.1.1. Unicast Discovery Protocol 7
6.1.2. Multicast Request Protocol 7
6.1.2.1 Details zum verwendeten Multicast 7
6.2. LookUp 8
6.3. Join 8
6.4. Leasing 8

7. Zusammenspiel 9

8. local service vs remote service vs smart proxy 10

9. Unterschiede zwischen Java RMI und JINI 10

10. Ausflug: Bluetooth 11

11. Weitere Begriffe 11

12. Resümee und Ausblick 12

Anhang 13
Weiterführendes 13
Quellenangaben 13

JINI – das „Kaffeemaschinen-Mikrowellen-Toaster"- Betriebssystem

So wurde es jedenfalls auf der CeBit vorgestellt.
Oder doch nicht? Was ist eigentlich „*JINI*" ?

1. Einführung und Motivation

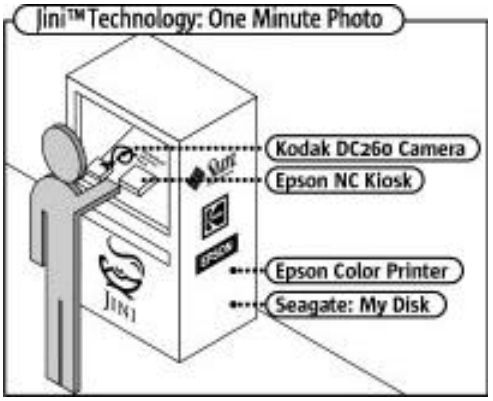
JINI ist die Abkürzung für „Java Intelligent Network Infrastructure" (ob diese Interpretation erst nachträglich entstand, darüber herrscht in den verschiedenen Publikation keine Einigkeit). Entgegen der landläufigen Meinung ist *JINI* jedoch kein Betriebssystem, sondern vielmehr ein Betriebssystemaufsatz. Es handelt sich um eine Technologie, welche auf Java basiert und eine neue Art der Infrastruktur für Netzwerke darstellt. Dabei werden größtenteils bereits bekannte Komponenten (z.B. „100% pure Java" und „Remote Method Invocation" [RMI]) genutzt. Dies erleichtert die Migration bestehender Strategien und Lösungen auf *JINI*, da nicht alles von Grund auf neu designed werden muss.

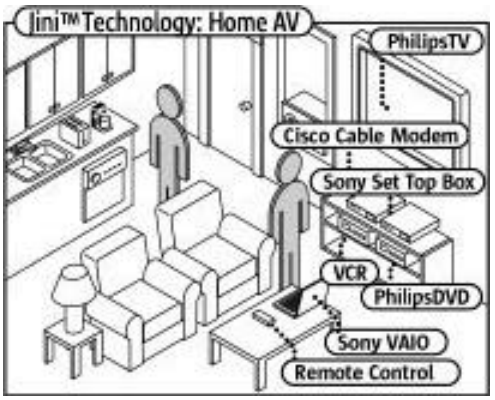
Warum aber *JINI*? Die immer weiterführende Expansion der vorhandenen Netzwerke und deren globaler Zusammenschluss führen zur Notwendigkeit, diese logisch wieder in kleinere, leicht administrierbare Teile zu zerlegen. *JINI* bietet diese Möglichkeit über eine noch höhere Abstraktion und Zusammenfassung von Netzwerkdiensten; sowohl auf Seiten der Anbieter (Server) als auch der Nutzer (Clients), wobei es bei *JINI* keine Rolle spielt, ob es sich dabei um PCs oder sonstige Hardware aus dem alltäglichen Leben handelt. Genau diese so verschiedenartigen Geräte in ein logisches Netz zu integrieren ist ein weiteres Ziel von *JINI*, welches durch die Kapselung von Hardware (erreicht durch die „Java Virtual Machine" [JavaVM]) und Protokollen erreicht wird.
Wie bereits erwähnt, setzt *JINI* auf Java auf. Diese Komponente muss schon zur Verfügung stehen; sie als Ganzes abstrahiert die darunterliegenden Layer (Betriebssystem und Hardware) und stellt *JINI* eine einheitliche Basis zur Verfügung (vgl. dazu „Aufbau von *JINI*" auf Seite 6). Bereits heute gibt es für fast alle erdenklichen Plattformen im Computerbereich (z.B. Apple, IBM-PCs mit verschiedensten Betriebssystemen, Solaris) und ansatzweise auch für proprietäre Hardware wie Videorecorder und andere Haushaltsgeräte eine Implementation des Java-Standards resp. eine zur Hardware / dem Betriebssystem passende JavaVM. Dadurch ist man in der Lage, schon jetzt *JINI*-fähige Geräte herzustellen bzw. Netzwerke damit aufzubauen, ohne noch vorher auf die Implementierung als „OneChip-Solution" (Stichwort „embedded systems") warten zu müssen. Damit wird der Dynamik und der rasanten Entwicklung in der IT-Branche allgemein und der Softwareindustrie im Besonderen Rechnung getragen.

2. Beispiele

JINI kann überall da eingesetzt werden, wo Geräte miteinander kommunizieren sollen. Eine totale Integration wird noch etwas auf sich warten lassen, aber einige Namenhafte Firmen – darunter z.B. 3com, Canon, Cisco, Kodak, Nokia, Philips, Quantum, Seagate, Sharp und Sony - haben schon jetzt einige Beispielszenarien entwickelt und mit ihrer Hardware ausgestattet.

So wäre z.B. eine „One Minute Photo"-Box denkbar, an der man sich Passfotos nun nicht mehr analog, sondern mittels DigiCam und Farbdrucker machen lassen könnte. Da die DigiCam via *JINI* direkt mit dem Drucker reden kann, erübrigt sich der bisherige Aufwand eines für Grafikbearbeitung ausreichenden PCs als Vermittler.

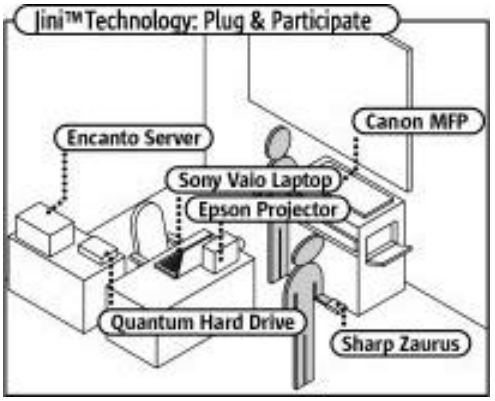




Im Heimbereich würde *JINI* nicht nur das Kabelgewirr verschwinden lassen, sondern sorgt z.B. auch für eine reibungslose Kommunikation (Stichwort hier: Senderabgleich) zwischen Videorecorder und Fernseher. Auch besteht langfristig die Möglichkeit, dass der PC in seiner gewohnten Form überflüssig wird, da der Fernseher mit Hilfe von *JINI* direkt mit dem Modem kommunizieren kann und somit internettauglich wird. Aber die Vision von *JINI* geht noch weiter – die Eingangs zur Diskussion gestellte Kaffeemaschine könnte auf Grund der eingebauten „Intelligenz" dann pünktlich zum Aufstehen am frühen Morgen heissen Kaffee bereitstellen.

Selbst unterwegs lässt einen *JINI* nicht alleine! Auf dem Flughafen könnte so z.B. ein mitgeführter „Personal Digital Assistent" (PDA) eine sog. „ad hoc"-Verbindung zum Server des Flughafen aufnehmen, um sich dort nach dem aktuellen Flugplan sowie Schaltern zu erkundigen, an denen man sich melden muss („Proximity Networking").

Auch den Büroalltag kann *JINI* erleichtern: so kann eine problemlose Kommunikation zwischen Kopierer, Datenserver der Firma, dem eigenen Arbeitsplatz und z.B. einer Mobilstation gewährleistet werden, die man ohne Konfigurationsaufwand mit Daten versorgt, die man als Aussendienstmitarbeiter dem Kunden präsentieren möchte.



3. Geschichte

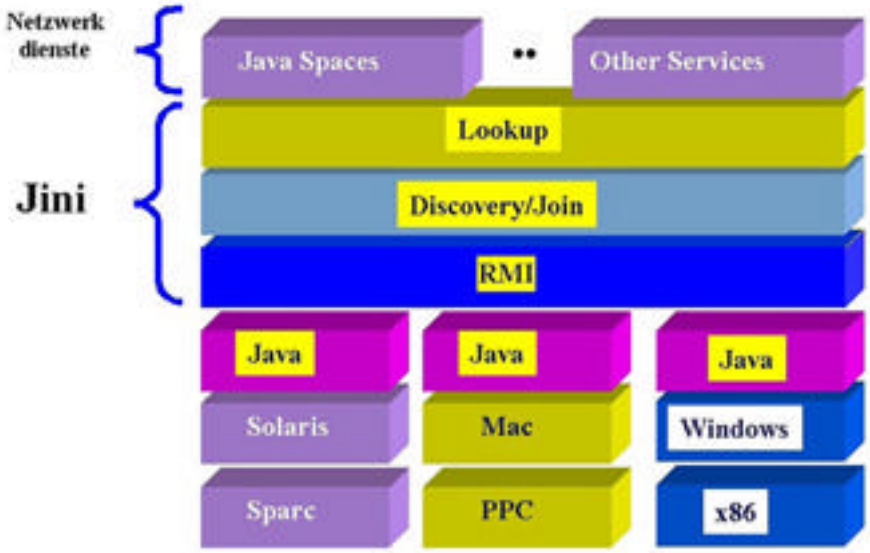
JINI ist eine noch recht junge Technologie, die sich zur Zeit noch im Entwicklungsstadium befindet.
Eine erste Erwähnung fand *JINI* im Jahre 1998 auf dem Titelblatt der „New York Times" – das zeigt, welche Beachtung (und auch Hoffnung) man dieser neuen Technologie für Netzwerke entgegenbringt.
Der Öffentlichkeit präsentiert wurde *JINI* erstmals am 25. Januar 1999 in San Francisco. Im folgenden Monat wurde auf der CeBit in Deutschland die allseits bekannte Kaffeemaschine mit *JINI* vorgeführt.
Dieses Jahr konnte *JINI* erstmals auch eine tatsächliche praktische Anwendung finden. Im April wurde ein tragbares EKG-Gerät mit *JINI* ausgestattet, welches dem Patienten und dem Arzt einen leichten Datenabgleich und deren Einsicht gewähren soll.

4. Softwarelizenz

JINI wurde von der Firma Sun entwickelt und wird von ihr unter der neu entstandenen „Sun Community Source Code License" (SCSL) als Open Source zur Verfügung gestellt (nähere Informationen sind unter http://developer.java.sun.com/developer/products/jini erhältlich). Durch die Freigabe des Sourcecodes soll *JINI* zu einer schnellen und unbürokratischen Weiterentwicklung verholfen werden, wie sie z.B. auch dem Betriebssystem Linux beschert war.

5. Aufbau

Der Aufbau von *JINI* bzw. eines *JINI*-Systems lässt sich – ähnlich dem ISO/OSI-Schichtenmodell - wie folgt darstellen:



Wie man auf der Abbildung erkennen kann, setzt *JINI* auf der Java-Plattform auf. Die „Komponente *JINI*" wird durch die Nutzung der bekannten RMI und dem, was *JINI* eigentlich ausmacht – die Services „Discovery/Join" und „Lookup" – gebildet. Darauf setzen dann die verschiedene Netzwerkdienste auf. Als Beispiel seien hier die sogenannten „JavaSpaces" genannt, auf die in Kapitel 11 (Seite 11) noch einmal kurz eingegangen wird.
Im Folgenden werden die einzelnen hier vorgestellten Basiskonzepte näher erläutert.

6. Basiskonzepte

Die vier grundlegenden Dienste von *JINI* sind

- Discovery
- LookUp
- Join
- Leasing

Über die Betrachtung dieser Dienste wird deren Beziehung untereinander und somit die Funktionsweise von *JINI* ersichtlich.

6.1. Discovery

Über das „Discovery" (dt. etwa „entdecken, auffinden") wird ein bestimmter Dienst oder eine Ressource im Netzwerk ausfindig gemacht. Das Besondere an *JINI* ist die Art der Suche: nicht nach IP-Adressen wie beim TCP/IP-Protokoll oder nach Namen wie bei herkömmlichen Verzeichnisservices (z.B. LDAP) wird gesucht, sondern nach bestimmten Attributen. Dadurch sind mehrere Service-Provider möglich. Als Beispiel hierfür wird gerne die Auswahl eines Druckers angegeben, auch wenn dies nicht das volle Spektrum der Leistungsfähigkeit von *JINI* wiederspiegelt. Über *JINI* sucht man nicht einen bestimmten Drucker im Netzwerk, sondern nach „einem Drucker, der A4 bedrucken kann, und das in Farbe mit mindestens 300dpi".
Dazu existieren zwei unterschiedliche Verfahrensweisen:

6.1.1. „Unicast Discovery Protocol"

Damit kann man einen spezifischen Service im Netz suchen. Als Beispiel wäre hier ein Drucker zu nennen, der im gleichen Raum steht, da man bei Suche nach einem Drucker mit bestimmten Eigenschaften sonst vielleicht einen als Ergebnis geliefert bekommt, der sich am anderen Ende des Gebäudes befindet

6.1.2. „Multicast Request Protocol"

Hierbei werden alle LookUp-Services, die zu einer (lokalen) Gruppe gehören, gefunden. Dies kann genutzt werden, wenn es egal ist, welches Gerät eine Aufgabe ausführt (z.B. eine langwierige Rechenoperation).

6.1.2.1. Details zum verwendeten Multicast

Permanente Multicast-Adressen werden von der „Internet Assigned Numbers Authority" (IANA) vergeben. Für *JINI* wurden zwei Multicast-Protokolle definiert:

- das „announcement-protocoll" wird verwendet, neue LookUp-Services im Netzwerk bekanntzugeben. Dafür wurde die „*JINI*-announcement"-multicast-IP-Adresse 224.0.0.84, Port 4160 reserviert.

- das „request-protocoll", welches den nächsten LookUp-Service findet, bekam die „*JINI*-request"-multicast-IP-Adresse 224.0.1.85, ebenfalls Port 4160, zugewiesen.

6.2. LookUp

Der LookUp-Service ist die zentrale Anlaufstelle (Registry) für alle Java-Dienste (z.B. JavaSpaces) und sollte daher immer redundant ausgelegt sein. Im Unterschied zu den bereits existierenden Verzeichnisdiensten (wie z.B. X.500, LDAP, Novell Directory Services [NDS] oder Microsoft´s ActiveDirectory) werden bei *JINI* jedoch nicht nur die Namen und die Beschreibung eines Objektes (hier im Speziellen: eines Dienstes) gespeichert, sondern auch der Dienst selber, nötige Schnittstellen und Treiber sowie dessen Attribute (vgl. das Beispiel mit dem Drucker weiter oben).

6.3. Join

Als „Join" wird das Registrieren eines Dienstes bei einem gefundenen Service bezeichnet. Damit wird der Dienste-Provider im Netzwerk bekanntgegeben. Der LookUp-Service weist dann dem Dienst eine interne, eindeutige ID („Universally Unique Identifier", kurz UUID) zu. Diese wird allerdings nur vom LookUp-Service selber benutzt, da – wie bereits vorstehend erwähnt – später nach den Attributen eines Dienstes und nicht nach dieser Nummer (die z.B. vergleichbar mit der Indexnummer eines Datensatzes in einer Datenbank ist) gesucht wird. Gleichzeitig werden auch die Schnittstellen (bei *JINI* als „Proxy" bezeichnet) auf den LookUp-Service kopiert, damit sie später vom Nutzer (Client) dort heruntergeladen werden können.

6.4. Leasing

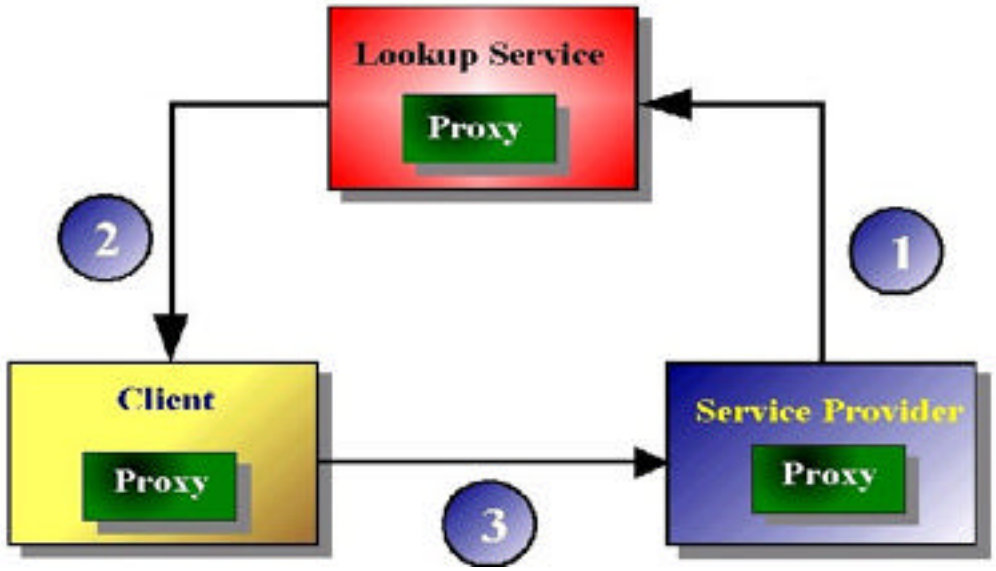
Nachdem ein Dienst im Netzwerk angemeldet wurde (Join) und somit zur Verfügung steht, kann ein Nutzer (Client) über das „Leasing" genannte Verfahren Zugriff auf die bereitgestellten Ressourcen erlangen. Dieser Zugriff ist jedoch zeitlich beschränkt auf die sogenannte „Lease-Time". Der Client muss in regelmäßigen Abständen eine Verlängerung der Nutzungserlaubnis beim Server beantragen. Geschieht dies nicht, erfolgt eine Freigabe der vom Client belegten Ressourcen nach Ablauf des Lease serverseitig durch den Garbage Collector. Sinn dieses Zugriffsverfahrens ist es, Netzwerkausfällen und Abstürzen einzelner Komponenten entgegenzuwirken.

Dabei ist es besonders wichtig, ein Transaktionsmanagement zu implementieren, damit z.B. durch Buchungen keine logischen Inkonsistenzen im Datenbestand auftreten. Im Gegensatz zu anderen Netzwerkmodellen und Java RMI bietet *JINI* ein solches „Transaction Tracking System" (TTS), wie es auch von diversen Serverbetriebssystemen (z.B. Novell) oder Datenbanksystemen angeboten wird. Es ermöglicht, dass bei einem Ausfall das sogenannte „Rollback"

ausgelöst wird und bereits vollzogene Datentransfers, die zu diesem Vorgang (Transaktion) gehören, automatisch wieder rückgängig gemacht werden.

7. Zusammenspiel

Das im vorangegangenen Abschnitt angerissene Prinzip des Zusammenspiels der einzelnen Dienste von *JINI* soll folgende Grafik noch einmal verdeutlichen:



Der erste Schritt (1) entspricht dem „Join" – ein beliebiger Service (z.B. Drucker) macht sich dem Netzwerk bekannt; registriert sich beim LookUp-Service nach Auffinden desselben über das „discovery" und übergibt diesem die notwendigen Parameter sowie ggf. einen Proxy für den Zugriff auf den Service.

Schritt (2) beinhaltet sowohl das von Seiten des Client initiierte „Discovery" als auch das anschliessende „Leasing" der Ressource (im Beispiel des Druckers), welches mit dem download des für den Zugriff nötigen Proxies vom LookUp-Service auf den Client verbunden ist.

Anschliessend (3) kann der Client direkt mit dem Service Provider (Drucker) kommunizieren, ohne über einen dritten Service zu gehen – wie es in bisherigen Netzwerken i.d.R. der Fall ist. Einzig und allein die Softwarekomponente „Proxy" (welche sich natürlich der verschiedensten Funktionen von *JINI* und des darunterliegenden Netzwerkbetriebssystems bedient) ist nun als Übermittler der Nachrichten zuständig. Die Koordination von z.B. PrintQueues obliegt somit direkt dem ausführenden Service, der Statusmeldungen usw. direkt über das Proxy-Konzept an alle angeschlossenen Clients übermitteln kann.

8. local service vs remote service vs smart proxy

Wie kann man sich einen solchen Proxy vorstellen? Es gibt drei Möglichkeiten, wie der Datenaustausch zwischen Client und Server ablaufen kann, wenn erst einmal nach dem beschriebenen Konzept eine Verbindung aufgebaut wurde.

- beim „local service" handelt es sich um ein „serializable object", welches dem Client übergeben wird und auf dessen VM läuft.

- der „remote service" stellt das Gegenstück dazu dar – es läuft auf der serverseitigen VM und nutzt ein Protokoll (z.B. RMI), um mit dem Client zu kommunizieren.

- eine Kombination von beidem wird dann „smart proxy" genannt und erlaubt somit den bidirektionalen Datenaustausch

9. Unterschiede zwischen Java RMI und *JINI*

Java RMI	***JINI***
RMI clients nutzen die Klasse Naming.Lookup(), um den angeforderten RMI-Service zu finden	*JINI* clients nutzen den „discovery"-Prozess, um Lookup-Services zu finden.
der die Informationen über andere Services speichernde Dienst ist die „RMI registry"	in *JINI* ist das der Lookup-Service
der RMI-Client muss den „RMI registry host" sowie den RMI-Server genau kennen	der *JINI*-Client kann ohne genaues Wissen den Service/Server finden
die Methode ist starr, da ein Client von einem bestimmten Service abhängig ist	die Methode ist toleranter gegenüber Fehlern im Netzwerk und maximiert die Unabhängigkeit von einem bestimmten Service-Provider (Server)
Das RMI-Proxy-Konzept muss strikt eingehalten werden	Das *JINI* Proxy-Konzept ist protokollunabhängig und der Proxy führt selber requests aus oder nutzt einen RMI-call oder einen eigenen Proxy-Provider, um einen request auszuführen
Es existieren keine vorgefertigen Konzepte für Transaktionen, verteilte Ereignisse oder die zeitabhängige Nutzung (Leasing)	Bei *JINI* wurde u.a. genau auf solche Dinge wert gelegt, um dem Programmierer weitere Arbeit abzunehmen

10. Ausflug: Bluetooth

JINI soll die Kommunikation zwischen Geräten unterschiedlichster Art ermöglichen. Wie jedoch im Aufbau eines solchen Gerätes (vgl. Kapitel 5) bereits zu ersehen ist, existiert diese Plattformunabhängigkeit erst relativ weit „oben" im Schichtenmodell. Abhilfe dazu soll eine neue Technologie mit dem Namen „Bluetooth" schaffen. Neben der Interoperabilität verschiedenster Geräte hat Bluetooth auch die Aufgabe, die bisherige „straight trough"-Verkabelung (ein Gerät ist direkt mit dem nächsten verbunden) durch eine Funkraumlösung zu ersetzen. Bluetooth ist jedoch – im Gegensatz zu *JINI* – nicht von Sun. Vielmehr wird es in einer offenen Initiave gefördert, zu der bereits Unternehmen wie 3Com, Ericsson, Intel, IBM, Lucent, Microsoft, Motorola, Nokia und Toshiba gehören.
Bei Bluetooth werden mehrere sich in räumlicher Nähe zueinander befindliche Geräte zu einem sog. „Piconet" zusammengefasst und einigen sich untereinander auf eine gemeinsame Kommunikationsbasis, die ebenfalls Redundanzen und Ausfallsicherheiten beinhaltet. Antwortanforderungs-Algorithmen und Verschlüsselungsfunktionen ergänzen sinnvoll das Sicherheitskonzept von Bluetooth.
Ähnlich der weltweit stark verbreiteten TCP/IP-Protokollsuite unterstützt auch diese neue Technologie zwei Linktypen, die festlegen, welche Paketart transportiert werden kann. Der „Synchronous Connection Oriented Link" (SCO) wird überwiegend für Sprache verwendet, während der „Asynchronous Connectionsless Link" (ACL) hauptsächlich für Datenpakete genutzt wird.
Die derzeitige Konzeption des Standards beschränkt den maximalen Datendurchsatz allerdings auf 1MBit/s. Somit kann Bluetooth (vorerst) nicht als Ersatz für herkömmliche Netzwerke oder z.B. qualitativ hochwertiges Streaming verwendet werden.

11. weitere Begriffe

JavaSpaces:
Dabei handelt es sich um einen einfachen, schnellen und universellen Mechanismus, um verteilte Ressourcen, Services und Objekte zu teilen (sharing), koordinieren und zur Kommunikation dieser Objekte untereinander (-> Object-repository)

Lookup-Group:
So bezeichnet man eine Gruppe von (logisch oder physisch) zusammengefassten Services

Djinn:
Für alle Geräte, Ressourcen und Dienste, welche über *JINI* durch einen Lookup-Server/-Gruppe verwaltet werden, verwendet man „Djinn" als Sammelbezeichnung

12. Resümee und Ausblick

So euphorisch es teilweise klingen mag – neu ist der Ansatz von Sun nicht. Auch Softwaregigant Microsoft hat mit seiner „.NET"-Technologie etwas durchaus Ähnliches im Sinn. Dessen erste Ursprünge finden sich sogar schon in mancher .DLL von Windows Version 3. Jedoch ist bis heute kein komplettes Konzept wie bei *JINI* veröffentlicht. Mit der sich gerade in der Einführung befindlichen Programmiersprache „C#" will Microsoft seine Eigenentwicklung jedoch wieder aufnehmen und weiter forcieren. Analog zu den in Java gebräuchlichen „Java-Binaries" hat Microsoft eine sogenannte „Interface Language" (IL) als Ebene zwischen Quelltext und Hardware-kompatiblem Binary eingeführt. Insgesamt wird das gesamte Konzept von Microsoft wohl unter dem Namen „Universal Plug and Play" vermarktet werden – so der heutige Stand.
Aber auch andere Firmen haben schon an einer einheitlichen Kommunikationsplattform gearbeitet. So seien hier stellvertretend für noch einige kleinere Ansätze die Konzepte der grossen Firmen HP („Chai") und Lucent mit ihrem „Inferno" genannt.

Ob *JINI* der Durchbruch bzw. die Behauptung am Markt neben den Systemen der Konkurrenz gelingen wird, ist fraglich. Jedoch ermöglicht die Verwendung von Java – welches wie erwähnt für zahlreiche Plattformen bereits erhätlich ist – eine leichte Migration schon bestehender Netzwerkstrukturen auf *JINI*. Ausserdem dürfte die SCSL *JINI* dazu prädestinieren, dass neben den dann vielleicht etablierten kommerziellen Systemen der anderen Anbieter *JINI* als Plattform für neue Konzepte innerhalb der neuen Netzwerkstruktur genutzt werden kann, da hier eine breite Masse von Programmierern eigene Ideen recht einfach einbringen, implementieren und ausprobieren kann.

Jedoch soll auch ein grosser Nachteil aller dieser Ansätze nicht verschwiegen werden: durch die immer stärkere Abstraktion bzw. Objektorientierung gehen natürlich immer mehr spezifische Eigenschaften des jeweils angrenzenden Layers verloren.
Ausserdem wird durch diese Struktur immer mehr Speicherplatz verbraucht und Rechenleistung von Nöten. Da diese sich jedoch recht unabhängig ständig fast wie von selber steigern (derzeit ist eine Verdopplung von Speicherplatz bzw. Geschwindigkeit ca. alle ein- bis anderthalb Jahre der Fall), kann man diesen Punkt als Softwareentwickler beinahe vernachlässigen.

Insgesamt zeichnet sich jedoch eine Verschmelzung bestehender Lösungen (z.B. Directory-Services, Server, Netzwerkprotokolle) ab, die unsere derzeit recht heterogene Netzwerklandschaft wieder überschaubarer machen könnte. Insofern ist *JINI* ein weiterer, guter Ansatz, der – gerade im Bereich der Forschung und Weiterentwicklung von Netzwerken und Kommunikation – in unserer Informationsgesellschaft nötig ist und weiter verfolgt werden sollte.

Anhang

Weiterführendes

- The *JINI* community: http://www.jini.org
- Mailingliste „*JINI*-USERS“: http://archives.java.sun.com/archives/jini-users.html
- FAQs: http://www.sun.com/jini/faqs , http://www.artima.com/jini/faq.html
- Jiro – Implementation der „Federated Management Architecture“ (FMA): http://www.jiro.com
- Ein „public-Lookup-Service“ zum Testen von eigenem Code: http://www.jini.canberra.edu.au
- „*JINI* Specification“, (06/1999), Addison Wesley, ISBN: 0201616343 (385 Seiten, enthält trotz des Namens nur wenige Spezifikationen)
- „Professional Java Server Programming“, (08/1999), WROX Press, ISBN: 1861002777 (1121 Seiten, enthält Informationen über *JINI* und JavaSpaces)

Quellenangaben

- *JINI* bei Sun: http://www2.sun.com/jini
- Java Magazin; Ausgabe 3.1999: „Die Geister, die ich rief”
- The *JINI* FAQ from jGuru: http://www.jguru.com/jguru/faq
- „core*JINI*” (07/1999), Prentice Hall, ISBN: 013014469x (722 Seiten, sehr zu empfehlen, inkl. Schnittstellen)
- „*JINI* – die intelligente Netzwerkarchitektur in Theorie und Praxis“, Addison-Wesley, ISBN 3-8273-1556-5 (170 Seiten)
- Bluetooth Website: http://www.bluetooth.com
- Microsoft´s „.NET“
  Source: Microsoft; C# Programmer´s Reference & Language Reference
  http://msdn.microsoft.com/vstudio/nextgen/technology/csharpintro.asp