Proseminar „Mobil Computing“: JINI und JavaSpaces

Von Bernd Dutkowski

Einleitung

SUN propagiert schon länger unter dem Werbespruch „The Network is the Computer“ die komplette Vernetzung aller Computer. In der Praxis ergeben sich dadurch eine ganze Menge Probleme. Die meisten entstehen dadurch, daß es keinen generischen Mechanismus gibt um Ressourcen zu verwalten. Genau hier will SUN mit JINI eine Lösung anbieten. JINI steht für „Java Intelligent Network Infrastructure“. Der Name ist dabei Programm: SUN setzt hier komplett auf Java und versucht die Verwaltung von Diensten und Ressourcen „intelligent“ zu gestalten. Der Benutzer soll so wenig wie möglich eingreifen müssen, wenn er nicht will.

Szenario

Ein oft strapaziertes Beispiel ist die Geschichte des Managers, der bei einer Konferenz den anwesenden ein Dokument, das er auf seinem Laptop mitführt, ausdrucken will.
Derzeit muß er zuerst einen Systemandministrator kontaktieren, der ihm einen Drucker zuweist, diesen anschließen oder über das Netzwerk eine Verbindung aufbauen, passende Treiber installieren und zu guter letzt noch vom Systemadministrator die passenden Rechte zugewiesen bekommen. Um zu guter letzt dann ein einfaches bedrucktes Blatt in Händen zu halten.
Dank JINI soll das ganze viel einfacher funktionieren. Es reicht eine Verbindung zum Netz aufzubauen und der Rest wird wie von Geisterhand erledigt. Er muß nur noch den richtigen Drucker auswählen.
JINI soll überall da zum Einsatz kommen, wo Ressourcen wie Drucker, Rechenkapazität, Speicher, ... verwaltet werden sollen. Dabei wird der Einsatzbereich nicht auf Haushaltsgeräte oder reine Systemadministration beschränkt.

Das JINI Universum

Das JINI Universum besteht aus Diensten, Clients, die Dienste Benutzen und LookUp-Services bei denen Sich Dienste anmelden und Clients nach Diensten suchen können. Es können sowohl von Clients, Diensten und LookUp-Services mehrere Instanzen existieren. Ein LookUp-Service sollte aber mindestens erreichbar sein. Damit das nicht zu einem Gefahrenpunkt wird, ist es durchaus denkbar LookUp-Services in einem Cluster ähnlichem Verbund zu betreiben.

Aufbau

Schichtmodel

(Siehe dazu Folie Nr. 6)

Discovery & Join

Auf der untersten Schicht des JINI Models befindet sich die Familie der „Discovery and Join“- Protokolle. Jedes dieser Protokolle setzt auf UDP auf.
Das „Multicast Request Protocol“ wird von Clients oder Diensten genutzt, um LookUp-Server zu lokalisieren. Dazu sendet der Dienst oder Client ein UDP-Broadcast-Packet an die Multicast-IP 224.0.1.85 Port 4160 mit der Liste aller bekannten LookUp-Services. Ein LookUp-Service, der noch nicht in dieser Liste steht, antwortet.
Mit Hilfe des „Multicast Announcement Protocol“ publizieren LookUp-Server ihre Verfügbarkeit nach einem Reboot oder bei einem Neustrat.
Da Mulicasts an die Adresse 224.0.1.85 nicht geroutet werden, definierte SUN ein weiteres Protokoll, das es ermöglicht, einen LookUP-Server direkt anzusprechen. Dieses „Unicast Discovery Protocol“ setzt jedoch voraus, daß die IP des LookUp_Services bekannt ist.
Die genaue Spezifikation der Protokolle wurde von SUN veröffentlicht, jedoch ist es nicht nötig, sich als Programmierer damit auseinander zu setzten, das die JINI-API dies komplett Kapselt und verdeckt.

Remote Method Invocation

„Remote Method Invocation“ oder kurz RMI baut im Gegensatz zu Discovery & Join auf TCP auf. RMI ist schon länger Bestandteil von Java und wurde zur Objektkommunikation über Prozeß und Netzwerkgrenzen hinweg geschaffen. Mit RMI können entfernte Mothodenaufrufe ausgeführt werden und Klassen wie auch Objekte transferiert werden. Die verwendeten Mechanismen erinnern sehr stark an IIOP/CORBA, sind jedoch bei weiten so mächtig und dadurch sehr viel einfacher.

JavaSpaces

JavaSpaces ist ein Objektperistenzdienst der auch außerhalb der JINI Welt Verwendung finden soll. Die grundlegende Idee wurde von „Tuppelsystemen“ abgeschaut. Auch hier ist die API sehr einfach gehalten worden. Es gibt nur vier primäre Operationen. Mit „write“ wird ein Objekt im „JavaSpace“ abgelegt. „read“ liefert eine Objekt zurück, ohne es jedoch aus dem Speicher zu entfernen. „take“ hingegen nimmt es aus dem Speicher. Per „notify“ kann ein Client auf das eintreffen eines Objektes warten. Die Kommunikation zwischen Client und JavaSpace erfolgt über RMI und integriert sich damit nahtlos in Java.
In verteilten Systemen muß immer mit dem Ausfall von Komponenten gerechnet werden. In diesem Fall ist es fatal, wenn ein Client ein Objekt ablegt und danach die Verbindung verliert. Der Server kann nun nicht entscheiden, ob das Objekt noch gültig ist oder nicht. Um dies zu vermeiden, wird beim anlegen des Objekts im JavaSpace ein „Verfallsdatum“ mitgegeben.

Diese „Leases“ können beliebig verlängert werden und garantieren ohne großen technischen Aufwand eine für JINI hinreichende Konsistenz.
JavaSpaces sind auch transaktionsfähig und benutzen dazu den „Java Transaction Service“. Diese ist in der Verbindung mit JINI jedoch nicht von größerer Bedeutung.

LookUp-Service

Der LookUp-Service bildet den zentralen Punkt im JINI Konzept. Dienste melden sich an und Clients können nach Diensten suchen.
Zur Verwaltung der Informationen benutzt die Referezimplementierung von SUN JavaSpaces.
Auch hier kapselt die JINI-API das meiste.

JINI-API

Die JINI-API kapselt die darunter liegenden Schichten und bietet sowohl für Client wie auch Dienste eine schlanke API. Die gesamte JINI Klassenbibliothek ist derzeit 48 k Byte groß und somit auch Für Geräte mit wenig Speicher geeignet. Es wird nur eine VM nach Java 2.0 oder höher gefordert. Somit ist JINI auch auf PersonalJava lauffähig.

JINI-Service

Ein JINI-Service besteht einmal aus einem Java-Interface, das die Schnittstelle zu diesem Dienst definiert. Diese Schnittstele sollte für die jeweilige Gerät genormt sein. So sollten zum Beispiel alle Drucker über das gleiche Interface ansprechbar sein. Zum anderen muß dieses Interface implementiert werden und das Gerät angesteuert werden. Über welches Protokoll der Treiber und das gerät kommunizieren ist dabei den Entwicklern überlassen. Es kann RMI, IIOP-CORBA oder sonst ein Protokoll, das auf TCP/IP oder UDP/IP aufsetzt. Wie die Logik in dieser Implementierung zwischen Gerät und Proxyobjekt verteilt wird, bleibt dem Programmierer überlassen.

JINI-Client

Der Client sucht im LookUp-Service nach passenden Diensten. Um sie zu benutzen, muß er das Interface kennen. Von der Implementierung oder gar der Kommunikation zwischen Treiber und Gerät braucht er nichts zu wissen. Somit reduziert sich die Programmierung auf das Wesentliche.

Zusammenspiel

Das Zusammenspiel zwischen den einzelnen Komponenten ist im Prinzip recht einfach, jedoch durch die gegeben Flexibilität kann es unüberschaubar werden. Ich beschränke mich auf einen einfachen und allgemein gehaltenen Fall.
(Siehe dazu auch Folie Nr. 13)

LookUp-Service startet

Beim Start meldet der LookUp-Service seine Verfügbarkeit per „Multicast Announcement Protocol“ und wartet auf Anfragen von Clients und Diensten. Für erste Versuche liegt eine Referenzimplementation dem JINI-SDK bei.

Service meldet sich an

Ein Service sucht per „Multicast Request Protocol“ oder „Unicast Discovery Protocol“ nach einem lookUp-Servide und meldet sich bei diesem an. Auch lauscht der Dienst und reagiert auf Packete des „Multicast Announcement Protocol“. Beim LookUp-Service hinterlegt er sein Proxy-Objekt.

Client sucht Service

Ähnlich verhalten sich Clients. Sie suchen sich einen LookUP-Service und fragen diesen nach passenden Diensten und benutzen die Proxy-Objekte dank RMI wie lokale Objekte.

Probleme die durch JINI entstehen oder nicht gelöst werden können

Da die jetzigen VMs noch recht ressourcenhungrig sind ist es noch nicht möglich JINI in jedes Gerät einzubauen. Aber die Entwicklung schreitet rasch voran und mit PersonalJava ist schon eine Lösung in sichtweite.
Auch stellt das Fehlen von fertigen Produkten ein Problem dar, da die noch recht junge Technologie nicht in der Praxis erprobt werden konnte.
Durch die Multicasts und den mobilen Code entsteht eine hohe Netzlast, die gerade bei mobilen Geräten mit schmaler Netzanbindung das ganze Konzept unbrauchbar machen oder zumindest stark einschränkt. Hier sind die Programmierer der Services gefragt. Es muß darauf geachtet werden, daß die Proxy-Objekte nicht zu groß werden und die Kommunikation zwischen Proxy-Objekt und eigentlichem Dienst nicht zu aufwendig ist.
Ein weiterer Schwachpunkt ist die Festlegung auf IP-Netze. Bei der komunikation über Infrarot oder Bluetooth bläht dies die Implementierung unnötig auf.
Auch Firewalls können im praktischen Einsatz Probleme bereiten, da sowohl UDP wie auch TCP basierende Protokolle eingesetzt werden. RMI ist Dank der Möglichkeit es durch HTTP zu tunneln unproblematisch. Jedoch läßt JINI den Dienstentwicklern freie Hand bei der Wahl ihres Protokolls und damit werden Firewals immer eine potentielle Problemquelle sein.
Das größte Problem jedoch stellt das Fehlen einer Organisation dar, die sich um die Normung der Dienst-Schnittstellen kümmert. Derzeit kocht jeder Hersteller sein eigenes Süpchen und ruiniert dadurch JINI. SUN selbst sieht bis jetzt noch keinen Handlungsbedarf.

Eine Alternative: „Universal Plug and Play“ (UPnP)

Eine alternative zu JINI soll „Universal Plug and Play“ darstellen. Dieser Standart setzt komplett auf XML zur Ressourcenbeschreibung und http über UDP. Das nicht zwingend auf IP aufsetzen muß und somit auch direkt Infraror oder ähnlichem betrieben werden kann. Im

Gegensatz zu JINI verzichtet man auf mobilen Code. UPnP wird von ca. 60 Firmen unterstützt, allen voran Microsoft. Microsoft sieht hier die Möglichkeit eine Alternative zu JINI zu etablieren und so der unbeliebten Sprache den Nährboden zu nehmen. Auch hier fehlen noch Praktische Erfahrungen.

Fazit

JINI stellt nichts neues dar. Alles war schon mal da. Jedoch ist die Art und Weise der Zusammensetzung gut gelungen. Auch wenn an manchen Stellen noch nachgebessert werden muß, stellt es doch schon eine gute Basis für die Entwicklung von Produkten dar. Gerade in Hochschulen wird intensiv an vielen Projekten gearbeitet. Ob sich JINI letztendlich durchsetzt ist jedoch nicht eine Frage der Technik. Die junge Geschichte der Computer lehrt uns, daß andere Parameter wichtiger sind.

Literatur

http://www.jini.org/
http://java.sun.com/
http://www.cetus-links.de/
„Vom Suchen und Finden“ iX 7/1999 S. 108 ff
„Unter dem Hammer“ iX 3/2000 S. 174 ff
„Bezaubernde Geräte“ iX 4/1999 S. 144 ff
„Geist aus dem Netz“ iX 11/1998 S. 166 ff