thema

Jini, geen afkorting overigens, is de naam van een gedistribueerde omgeving die 'Plug and Play in een netwerk' biedt. In feite kan Jini beschouwd worden als een Service Oriented Architecture (SOA) *avant la lettre*. Een grote doorbraak van Jini is vooralsnog uitgebleven. Een wijziging van het licentiemodel naar Apache Open Source en de goede kwaliteit van de Jini-middleware leidt de laatste tijd tot een hernieuwde interesse in de technologie. In dit artikel zet Willem Koppenol daarom de architectuur en werking van Jini op een rij.

Jini staat klaar

Gedistribueerde systeemontwikkeling

Jini dateert al uit 1998 en is ontstaan uit de behoefte gedistribueerde systeemontwikkeling te vereenvoudigen. Een device of een software service kondigt bij aansluiting op het netwerk zijn aanwezigheid aan. Clients die een dergelijke service willen gebruiken kunnen hem vervolgens lokaliseren en aanroepen om taken uit te voeren. Ook het oorspronkelijke Java-idee allerlei devices in een netwerk op een simpele manier met elkaar te verbinden is een inspiratiebron voor Jini. Jini is inmiddels een bewezen technologie met vele praktijktoepassingen.

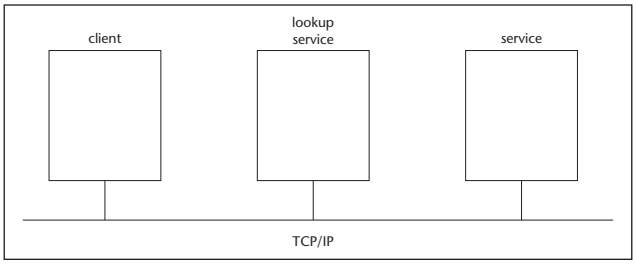
Introductie Jini is een Java technologie en biedt de infrastructuur voor het bouwen en uitrollen van gedistribueerde systemen die zijn georganiseerd als een federatie van services. Een service is daarbij iets wat zich in het netwerk bevindt en een nuttige functie kan vervullen. Hardware-devices, software-applicaties, communicatiekanalen en zelfs gebruikers kunnen een service leveren. Een met Jini uitgeruste printer kan bijvoorbeeld een printservice leveren. Een client kan gebruik maken van de federatie van services die zich op dat moment in het netwerk bevinden om een doel te bereiken.

De fundamenten van Jini zijn object serialisatie en RMI (Remote Method Invocation) die het mogelijk maken Java objecten over het netwerk van virtual machine naar virtual machine te transporteren. In de Jini architectuur wordt één van de belangrijkste fundamenten van objectorientatie, namelijk de scheiding tussen interface en implementatie, toegepast op services in een netwerk. Clients, zoals devices uitgerust met Jini kunnen, zonder overeenstemming over het gebruikte netwerk protocol, met elkaar communiceren via Java interfaces op objecten.

Bij de introductie van de Jini-technologie was er nogal een focus op devices, die met Jini konden communiceren in betrouwbare gedistribueerde applicaties. Jini vereiste echter een behoorlijk deel van de Java runtime stack en deze was vooralsnog onvoldoende aanwezig in de toenmalige devices. Niets in de specificaties zegt echter dat je Jini alleen voor devices kunt gebruiken. Gedistribueerde applicaties met software services, kunnen net zo goed hun voordeel doen met Jini. De eerste toepassingen van Jini kwamen dan ook op dit vlak tot stand bij organisaties die problemen met gedistribueerde systemen moesten oplossen, zoals telecombedrijven en financiële instellingen.

Componenten Bij een download van Jini krijg je een aantal verschillende zaken. Jini specificeert een reeks middleware-componenten en ontwikkelaars kunnen beschikken over een API waarmee je services kunt schrijven en de middleware kunt gebruiken. In de tweede plaats krijg je een implementatie, in Java, van deze middleware in de vorm van een aantal Java-packages. De source code wordt bijgeleverd. In de derde plaats vereist Jini een aantal standaardservices en levert Sun hiervoor een basisimplementatie. Strict genomen zijn deze services geen echt onderdeel van Jini, maar ze zijn opgenomen om de gebruiker op weg te helpen.

Runtime-infrastructuur De Jini runtime infrastructuur leeft in het netwerk en voorziet in mechanismen waarmee services kunnen worden toegevoegd, verwijderd, gelocaliseerd en benaderd. Een belangrijk



FIGUUR 1. Componenten van een Jini systeem.

gegeven in Jini is dat het netwerk niet centraal wordt gecontroleerd. De services vormen een federatie en werken samen op basis van gelijkheid. Lookup services zijn het organiserende mechanisme is Jini systemen. Als een nieuwe service beschikbaar komt registreert deze zich bij lookup services. Als clients een service willen gebruiken raadplegen ze een lookup service. Hierin bevindt zich een directory van op dat moment beschikbare services. Als client en service elkaar eenmaal hebben gevonden, functioneren ze los van de lookup service.

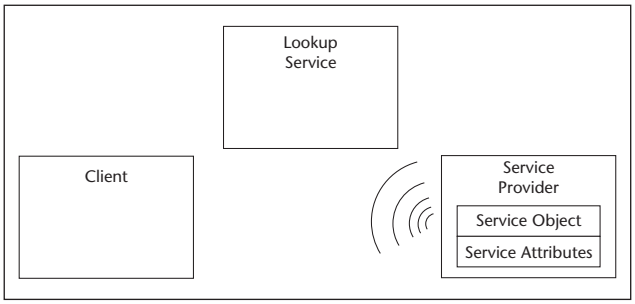
De runtime infrastructuur bevindt zich op drie plaatsen in het netwerk : in de lookup services, in de service providers zoals met Jini uitgeruste devices of software services, en in de clients. De runtime infrastructuur gebruikt een netwerk protocol, discovery genaamd, en twee objectprotocollen, join en lookup. Het discovery protocol stelt clients en services in staat lookup services te vinden. Met het join protocol kan een service zich registreren bij een lookup service. Een client gebruikt tenslotte het lookup protocol om een lookup service te vragen naar de services die hij nodig heeft.

DISCOVERY PROCES Zodra een Jini service provider, bijvoorbeeld een met Jini uitgerust device dat een print service aanbiedt, op het netwerk wordt aangesloten, volgt er een broadcast van een IP-pakketje op een bekend port nummer. De service kondigt hiermee zijn aanwezigheid aan, aan de lookup services. Onderdeel van het IP-pakketje is het IP-adres en portnummer waarop de service kan worden gecontacteerd door de lookup service. Lookup services luisteren naar dit soort pakketjes. Indien er verschillende lookup services in het netwerk aanwezig zijn, krijgen ze allen deze aankondiging. Bij ontvangst zal de lookup service contact zoeken met de afzender en RMI gebruiken om een service registrar object naar de afzender te sturen. De bedoeling van dit service registrar object is om verdere communicatie met de lookup service te vergemakkelijken.

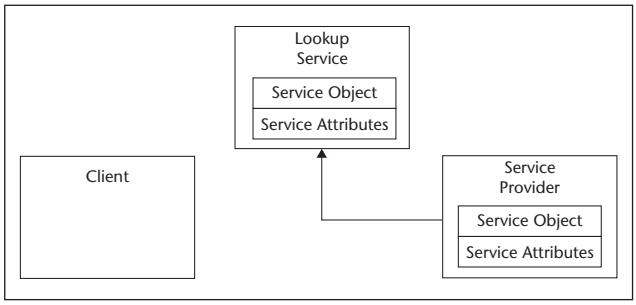
JOIN PROCES Voor een join operatie zal een service provider de register methode van het service registrar object aanroepen. Parameter is een service item object dat bestaat uit een bundel van objecten die de service beschrijven. Een copy van het service item object wordt naar de lookup service gestuurd, alwaar het wordt opgeslagen. De service provider is nu geregistreerd bij de lookup service. Een service item is een container voor verschillende objecten, waaronder een service object dat clients gebruiken om de service te benaderen. Het service item bevat mogelijk ook een aantal attributen. Dit kunnen willekeurige objecten zijn zoals iconen, classes voor een GUI interface op de service of classes die de service nader beschrijven. Service objecten implementeren gewoonlijk een of meer interfaces die clients gebruiken voor interactie met de service. Zo heeft de lookup service het service registrar object als service object. De register methode is gedeclareerd in het ServiceRegistrar interface.

De runtime infrastructuur gebruikt een netwerk protocol en de objectprotocollen join en lookup

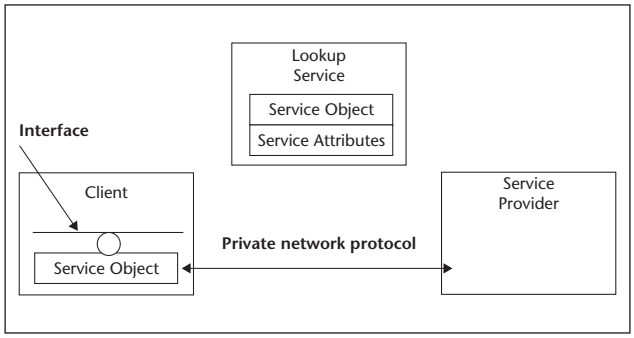
LOOKUP PROCES Clients doorlopen hetzelfde proces om een registrar object van de lookup service te krijgen maar doen daar vervolgens iets anders mee. Clients die een bepaalde service nodig hebben, raadplegen lookup services met de lookup methode van het service registrar object. Parameter voor lookup is een service template, waarin de zoek criteria voor de



FIGUUR 2. Broadcast van de aanwezigheid van de service.



FIGUUR 3. Service stuurt zijn Service Object naar de Lookup Service.

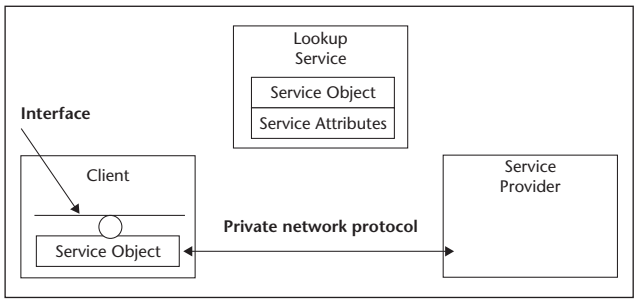


FIGUUR 4. Client krijgt een proxy naar de service via het lookup proces.

query staan. Meestal zal een client een service opzoeken via een Java type, zoals een interface. Maar het is ook mogelijk een service op te vragen op basis van een unieke service ID of de waarden van attributen. De lookup methode retourneert vervolgens een service object dat aan de zoek criteria voldoet. Ook is het mogelijk dat de service niet gevonden wordt of voldoen er meer services aan de zoek criteria. De client krijgt een copy van het service object dat fungeert als proxy voor het benaderen van de service en dat het service interface implementeert. De client kan de service nu gaan gebruiken.

Een service kan zich niet voor onbeperkte tijd bij een lookup service registreren

SERVICE-GEBRUIK Consequentie van de Jini architectuur is dat het network protocol dat wordt gebruikt om te communiceren tussen een proxy service object en een remote server niet bekend hoeft te zijn bij de client. Het netwerk protocol is immers onderdeel van de service implementatie en is bepaald door de ontwerper van de service. De client kan communiceren via het private service protocol omdat de service zijn eigen proxy code,



FIGUUR 5. Client communiceert via interface met service

het service object, injecteert in de client. Het geinjecteerde service object zou RMI, CORBA of een eigen protocol gebaseerd op sockets en streams kunnen gebruiken. Het maakt voor de client niet uit welk protocol wordt gebruikt want de client praat met het interface dat door de service object wordt geïmplementeerd.

SYSTEEMFALEN Een gedistribueerd systeem dat gebaseerd is op een federatie van services kent natuurlijk vele gedeeltelijke fout condities. Jini voorziet dit en heeft standaard recepten hoe hiermee om te gaan. De grote waarschijnlijkheid van het optreden van het gedeeltelijk falen van het systeem is in Jini een hoeksteen van de architectuur. Een typisch faal scenario is dat een client een proxy heeft naar een service die inmiddels ter ziele is omdat de host van de service is gecrashed. Wanneer de client vervolgens een methode aanroept op de proxy volgt een RemoteException. In het Jini programmeer model zal de client deze exceptie opvangen en vervolgens een nieuwe lookup doen van de service. Als we ervan uit mogen gaan dat de service redundant is en ook nog elders op het netwerk aanwezig, krijgt de client vervolgens van de lookup service een proxy naar een ander instantie van de service. De client functioneert met de nieuwe proxy gewoon door alsof er niets is gebeurd.

LEASE-CONCEPT Hoe weet Jini echter dat een service niet meer beschikbaar is? En hoe weten we dat we bij een nieuwe lookup geen proxy krijgen van een service die niet meer bestaat? De sleutel tot de reactiesnelheid van Jini ligt in het feit dat Jini erop rekent dat de service na verloop van tijd niet meer beschikbaar is. Een service kan zich niet voor onbeperkte tijd bij een lookup service registreren. Iedere service krijgt van de lookup service een lease, een in de tijd beperkte registratie. Als de service beschikbaar wil blijven als actieve service, moet hij de lease van tijd tot tijd hernieuwen. En als de host van de service gecrashed is kan dat natuurlijk niet meer. De gecrashte service kan dan niet meer worden gevonden door clients. Het lease interval is configureerbaar en kan een grote rol spelen in hoe snel een systeem op fout condities reageert.

JINI-VOORDELEN In vergelijking met andere technogieën voor gedistribueerde systemen heeft Jini een aantal voordelen. Als ontwerper van de service kan je zelf beslissen welke methodes op de server en welke eventueel lokaal worden uitgevoerd. Dit wordt bepaald in de proxy die de client ter beschikking krijgt en kan zelfs dynamisch gebeuren. Het is verder niet nodig clients te voorzien van veranderingen in de service code omdat ze die automatisch gedownload krijgen tijdens het lookup proces. En qua schaal-

baarheid is het in vele gevallen voldoende om een nieuwe service instantie te starten om meer clients te kunnen bedienen.

JAVA ONLY Vaak wordt verondersteld dat Jini systemen geheel in Java moeten worden geschreven. Dit is echter niet helemaal waar. De proxy moet weliswaar uit JVM bytecode bestaan, maar een client moet slechts in staat zijn om een Java object aan te roepen via een JVM. Tegenwoordige Jini clients kunnen daarom ook geschreven zijn in talen als C/C++ en .Net. Een volledige Java stack is niet nodig. Aan de service kant ben je nog flexibeler omdat een proxy tegen een willekeurige service kan praten en een willekeurig protocol kan gebruiken. De proxy zou bijvoorbeeld SOAP kunnen gebruiken om een Web Service te benaderen. De client is onwetend van de service taal en het gebruikte protocol.

SLOTWOORD Jini is een volwassen gedistribueerde programmeer omgeving met een schat aan mogelijkheden. Jini kan werken op kleine embedded devices met een memory footprint van enkele tientallen kilobytes als ook op grote parallelle super computers. Jini heeft niet dezelfde bekendheid in vergelijking met een concurrerende technologie als Web Services, maar heeft een aantal interessante voordelen. Een belangrijk aspect van Jini is dat het omgaan met systeem falen door middel van het lease concept zit ingebakken in de architectuur. Een ander belangrijk aspect van Jini is dat het niet gebonden is aan een bepaald protocol. Clients en services moeten uitsluitend overeenstemming hebben over de gebruikte Java interfaces om te kunnen communiceren. In het verleden was het licentie model nogal ongebruikelijk en was er aanvankelijk onzekerheid of Sun de ondersteuning zou continueren. Tegenwoordig is Jini in versie 2.1 open source onder de Apache 2.0 license en een blijvende optie.

drs. Willem Koppenol is Senior Trainer en Product Specialist Software Development bij Twice IT Training (wkoppenol@twice.nl).

PATCHES Patches PATCHES Patches PATCHES Patches PATCHES

Artikelen over onderwerpen als software-ontwikkeling, Java, UML, eXteme Programming en nog veel meer vindt u in het Online Archief van Array Publications. Vaktijdschriften als Software Release, Java Magazine, Database Magazine en ons Oracle vakblad Optimize hebben hun artikelenarchief online gezet. Dankzij de heldere zoekstructuur vindt u snel wat u zoekt op www.release.nl.

Future Group onderzoekt tijdsbesteding softwareontwikkelaar

Uit recent onderzoek van The Future Group blijkt dat ontwikkelaars gemiddeld 60 procent van hun tijd besteden aan het oplossen van applicatieproblemen. Andere betrokkenen in het proces, zoals support en testers, besteden ook veel tijd aan het verzamelen van informatie over het probleem, het vervolgens reconstrueren van de probleemsituatie en het uiteindelijk vinden van de oorzaak. Dit staat nog los van de onderlinge communicatie over het probleem.

Gaat het om een fout van een gebruiker? Is er een fout in de configuratie, het netwerk of de database? Of gaat het om een fout in de programmacode? Oorzaken van problemen in applicaties zijn vaak lastig te doorgronden. Veruit de meeste tijd gaat zitten in het zoeken naar de oorsprong. Is die eenmaal gevonden, dan is het probleem snel opgelost. Sterker nog: volgens The Future Group kost het vinden van de oorzaak 80 procent van de tijd en het werkelijk oplossen van het probleem maar 20 procent.

Ontwikkelaars die zich op applicatieproblemen richten besteden hieraan meer dan de helft (60 procent) van hun tijd. Testers besteden gemiddeld nog meer (65 procent) van hun tijd aan het documenteren en hercreëren van problemen.

Ook de helpdesk (of eerstelijns support) besteedt veel tijd (69 procent) aan applicatieproblemen. Het gaat dan vaak om nieuwe wachtwoorden en standaard gebruikersproblemen. Tweedelijns support houdt zich bezig met het vinden en oplossen van configuratiefouten, installatieproblemen en applicatieproblemen waarbij het niet noodzakelijk is de applicatiecode zelf te begrijpen (67 procent van de tijd). Derdelijns support lost complexe applicatieproblemen op, waaronder fouten in de applicatiecode (70 procent van de tijd).

BEA Systems presenteert tool voor real-time Java-applicaties

WebLogic Real Time Core Edition 1.1 is ontwikkeld om Java-applicaties in staat te stellen om de snelle en voorspelbare responstijden te leveren die nodig zijn in real-time omgevingen (bijvoorbeeld in de financiële sector). WebLogic brengt dus voor het eerst de productiviteitsvoordelen van ontwikkelen in Java naar de veeleisende wereld van real-time applicaties. Door gebruik te maken van Java, kunnen ontwikkelaars real-time requirements implementeren die makkelijker in een SOA geïntegreerd kunnen worden.

Door een bijna drievoudige verkorting van de responstijd presteerde WebLogic Real Time maximaal 30 milliseconden latency op zijn benchmark applicatie. Bedrijven die hun real-time infrastructuur-code nog in C of C++ beschikbaar hebben, kunnen dat voortaan in Java ontwikkelen.

Andere componenten in WebLogic Real Time zijn een versie van BEA JRockit voor garbage collection, BEA JRockit Runtime Analyzer (JRA) voor het meten en analyseren van applicatie latency, en BEA WebLogic Express Basic Edition, een Java Servlet engine die ontwikkelaars helpt om de BEA WebLogic productlijn kosteneffectief te gebruiken. WLRT ondersteunt tevens de Sun SPARC architectuur.