



UNIVERSITY OF
CALGARY

A tutorial report for SENG 609.22
Agent Based Software Engineering
Course Instructor: Dr. Behrouz H. Far

Tutorial

An Introduction to Jini

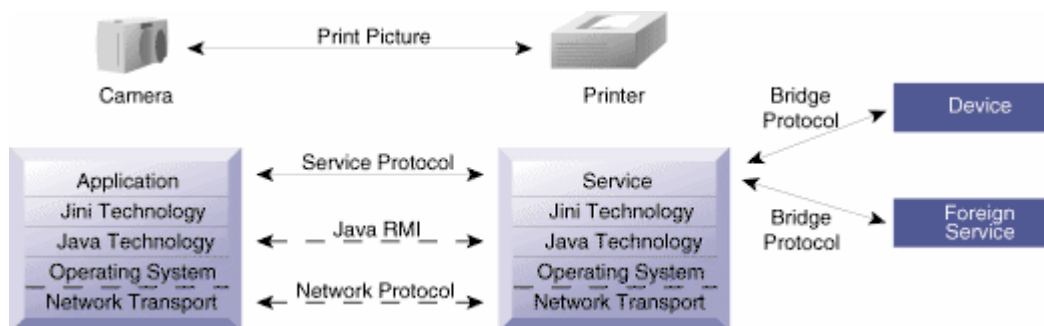
Baoqin Jenny Ye

What is Jini?

Today, with the rapid development of Internet or Intranet technology, distributed computing has become the mainstream of information technology. Compared with the traditional centralized computing environment, communication and coordination of distributed computing is more volatile, more difficult while less reliable. To address the need for a flexible, self-adaptive, reliable but also simple distributed computing mechanism, Sun has announced the release of its Jini technology in 1998. So far, Jini has entered into its 2.0 release and attracted more and more attention from the industry.

Currently, when a network environment is built, there must be lots of planning, installation and human intervention. Such heavy overhead has hindered the rapid progress of distributed computing. In a hope of solving this problem, Jini network technology has provided a simple mechanism to enable multiple devices in a network to deliver services among each other freely and trigger spontaneous interactions in a flexible manner. As a result, devices and software components in a network is formed into a federation where entities are able to join and leave dynamically, look up one another easily and use those services exposed to the federation by each entity.

Let's look at an example which illustrates the usage of Jini:



In the above picture, suppose that the printer (it probably has a built-in JVM) offers a service of printing pictures. Meanwhile another device in the network, the camera, needs to have a picture printed. It then has to first find a device provides such service. The lookup process is implemented in Jini (We will explain the details later). Once the camera has located the printer, the camera downloads and runs some code (implemented in Java programming language) supplied by the printing service. This code uses the underlying network transport, and perhaps RMI technology, to implement the printing service protocol needed to transmit the picture to the printer.

In this example, the camera and the printer doesn't need to know the existence of each other ahead of time. Jini is the underlying infrastructure which makes dynamic joint, runtime discovery and interaction and code downloading possible. The whole idea here is

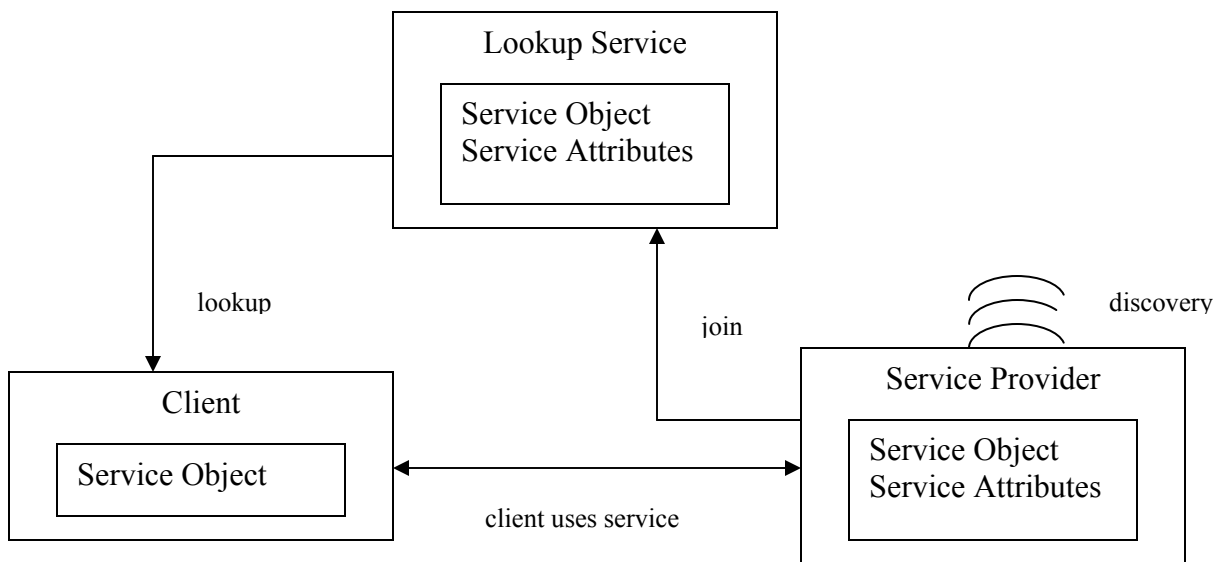
to have a plug-and-play network and get rid of the overhead of maintaining a network as much as possible.

On the other hand, Jini is built upon Java™ 2 Platform which, as promised, has provided a cross-platform infrastructure to distributed computing. As Java gaining more and more popularity in the industry, people have come to know more about Jini by and by.

In the upcoming sections, we will give a little more detailed introduction on the core technologies of Jini and also take a look at how Jini and agent-based system will benefit each other.

Discovery and Join & Lookup

The first question we will probably ask about Jini is “Yeah, it’s great to have a plug-and-play network. But how? How can the other people know of my existence? What should I tell people about myself? Who is going to look after that if some kind of system administration mechanism is not available?” The answer Jini has to this question is through a pair of protocols called *discovery and join* --- first the service locates an appropriate lookup service (by using the *discovery* protocol), and then it joins it (by using the *join* protocol). Discovery, join, and lookup form the heart of a Jini system.



From the above picture, we can see that in a Jini system, the lookup service acts as a central hub. It is where a service registers itself and where a client looks up the service it is requesting. So the first step takes place after a service provider is plugged into the network would be notifying the lookup service of its existence and the information about the service it provides. To do this, the service provider sends out multicasting requests on the local network for any lookup services to identify themselves. This is very different

from a traditional network where such kinds of lookup services must be known ahead of time (for example, the IP address of a DNS server). Once the service provider has found the lookup service, a service object for the service is loaded into there. This service object contains the Java programming language interface for the service, including the methods that users and applications will invoke to execute the service along with any other descriptive attributes. By now, the service is ready to be looked up and used. And this is the process of *discovery and join*.

So what happens when a client wants to use a service in the network? No doubt it will need to find the lookup service first too. Usually it uses the same discovery protocol to locate the lookup service as the service provider. After that, the client tries to locate an appropriate service by its type – i.e. its interface written in the Java programming language – along with descriptive attributes that are used in a user interface for the lookup service. Here the descriptive attributes usually serve as the criterion for this searching and matching process. Depends on the how the lookup service is implemented, the search mechanism can vary from simple, straight forward linear lookup to more complicated, intelligent search engines.

Once the communication between lookup service and client is established, the service object will be loaded into the client. The client then uses this service object to interact directly with the service provider and gets the task done.

In the Jini system, the ability to move objects and code from the service provider to the lookup service and from there to the client of the service gives the service provider great freedom in the communication patterns between the service and its clients. It is a good way to make sure that the information the client finds is in synchronized with the that held by service provider (it is not that easy to guarantee this in network environment where communication is unreliable). The client knows only the interface of the service through the service object and the service provider has all the flexibilities for implementation or changes of the implementation.

The leasing system

We all know that things are unreliable in a distributed network environment. When central administration is not available, how can the federation know that a service used to be ready is no longer there or which was not there a while ago comes backup again? Jini's answer to this is a leasing system.

In the Jini system, access to many of the services is lease based. A lease is a grant of guaranteed access over a time period. Each lease is negotiated between the user of the service and the provider of the service. Client asks for some period of time for a service; the service provider then grants some period to that, presumably taking what the client has asked for into consideration. If a lease is not freed – either because the resource is no longer needed the client or network fails, or the lease is not permitted to be renewed –

then both the client and the service provider of the resource may conclude that the resource can be freed.

The Transaction System

We are all familiar with the concept of transaction in database system. It is something you either complete or totally roll back. In a Jini system, we also have this type of challenge. Sometimes a series of operations, either within a single service or spanning multiple services needs to be carried forward together and should be wrapped in a transaction. In the Jini system, transaction handling is based on the common *two-phased commit* model. The system keeps records of the transaction environment before and after the transaction is processed so that it can be undone if necessary. Here Jini defines only a standard, a series of interfaces and leaves the implementation details to the service using those interfaces.

The Distributed Event Model

The Jini architecture supports distributed events. In a conventional computing environment, an object usually can receive events raised upon itself and trigger up the corresponding operations. Jini extends this model to allow other objects to register interest in events in a particular object and receive a notification of the occurrence of such an event. This is very useful for distributed event-based programs.

Jini and Agent System

With all the nice features of Jini, how can an agent system benefit from it? And so far to what extent the industry has adopted or utilized the Jini technology for agent systems?

From our introduction of Jini, we can see that Jini has provided an infrastructure for dynamic and self-adaptive distributed computing. The Jini technology itself concentrates more on low-level communication and infrastructure implementation. While an agent system focuses on high level interactions such as cooperation, coordination etc.

Dr. Far mentioned in class that an agent has 3 components: reasoning; communication and documentation engines. While reasoning and documentation engine can be localized inside agents, we definitely need a platform to carry on inter-actions. The fully asynchronous and anonymous environment provided by Jini will be a great foundation to build an agent system. With such features as discovery and join, lookup, leasing management, transaction management and distributed event handling, we can build an agent platform in an easier way. Actually, if we look at some of the features mentioned in the FIPA specification, we may find a few similarities (such as directory service, locator) between the two platforms.

So far, some effort has already been made in the industry to build agent frameworks upon Jini. Among them, Ronie (<http://gentoo.cs.umbc.edu/ronin>) and Paradigma (<http://www.ecs.soton.ac.uk/~ra00r/paradigma/>) are two well-known platforms.

Reference

- <http://www.sun.com/jini/whitepapers/technologies.html>
- *Jini Technology Core Platform Specification*; Sun Microsystems; May 2000
- <http://www.ecs.soton.ac.uk/~ra00r/paradigma/>
- <http://gentoo.cs.umbc.edu/ronin/>