



By Jim Waldo  
Sun Microsystems

# Mobile Code, Distributed Computing, and Agents

**B**ecause the Jini<sup>1</sup> system employs mobile objects, some have categorized it as a mobile agent system. However, such a categorization is based on an over-simplified view of mobile agent systems. In fact, mobile agent

systems have characteristics that clearly differentiate them from Jini technology. Unfortunately, these characteristics also make such systems highly unlikely to actually be useful, at least in my opinion.

In what follows, I will briefly describe some of the Jini networking system's features and attempt to show how this system's assumptions differ from those used to construct other distributed system frameworks. I will then characterize mobile agent systems, showing such systems' intrinsic differences from Jini and some of their basic problems. Finally, I will discuss how agent systems could use the Jini system approach to solve some of these intrinsic problems.

### The Jini networking system

A full discussion of the Jini networking system falls outside this article's scope. However, I will give a very quick overview of some of the system's major features to help with what follows.

The Jini networking system is a distributed infrastructure built around the Java programming language and environment. The basic communication model is based on the semantic model of the Java Remote Method Invocation system, in which objects in one Java virtual machine communicate with objects in another by receiving a proxy object that implements the same interface as the remote object. This proxy object deals with all communication details between the two processes. The proxy object can introduce new code into the process to which it is moved. This is possible because Java bytecodes are portable, and it is safe because of the the Java environment's built-in verification and security.

To this underlying communication model the Jini system adds some basic infrastructure and parts of a programming model. The infrastructure provides a mechanism by which

clients and services can join into the Jini network, while the programming constructs encapsulate mechanisms that allow reliable distributed systems to be built.

Java provides the Jini system with a mechanism to move mobile objects, including their code, safely and efficiently from a service to a client of that service. The Java type system forms the basis for identifying services, and its polymorphic nature lets it treat requests for service as requests for something that implements at least a certain type, although the service might offer more. However, the requirement for the Java language and platform is only at the network level; programmers can use non-Java objects to implement a Java network object that can live in the Jini world.

### Mobile code and mobile agents

Those who characterize the Jini system as a mobile agent system seem to rely on the assumption that any system that uses mobile objects must be an agent system. But while it seems obvious that moving objects (in a loose sense) is a necessary condition for a mobile agent system (since agents are objects), and while it might be true that most systems that use mobile objects have, in fact, been agent systems, object movement does not seem to be the only thing that characterizes a mobile agent system.

Let's look at a couple examples of mobile agent systems. The first of these has to do with shopping over the Internet. Rather than searching the Net for the best deal on, say, an airline ticket from Boston to London, I send an agent into the network to do the work for me. This agent will move from site to site, finding flight times, prices, and availability. Upon locating the best deal, the agent books my travel (using my credit card numbers, which it keeps securely) and returns to tell me what it has accomplished and when I need to have my bags packed.

A second example exploits the proximity networks looming on the horizon. When I enter a cab, my personal agent, which resides on my personal digital assistant device, joins into the cab's wireless proximity network. Consulting my schedule (also on my PDA), my agent determines that I'm going to the airport and moves to the location of the cab's

agent. My agent tells the cab's agent where I'm going. The two agents then negotiate a price (and perhaps a route), my agent pays (again using my credit card information), and my agent returns to my PDA, entering the transaction into my financial statement.

A final example returns us to the World Wide Web. In preparing a paper, I discover that I have neglected to do the scholarly research expected. I don't want to use the simple search engines, because I know that the number of false positive returns will far outweigh the number of good returns to my query. So, I send an agent out on the Web after telling it my paper's subject, and my agent jumps around the Web and returns to me the next morning, with the six papers most relevant to my subject.

I don't know if these are real examples of what mobile agent systems should do. Certainly, examples like these abound in the popular press, and researchers in the field have given exactly these examples when I ask what a mobile agent system could do.

The examples display what I take to be the characteristics of a mobile agent system, which include

- use of active objects—that is, objects that move from place to place and, when they arrive at a place, can obtain a thread of control automatically;
- interaction with various environments to find out information or perform actions on those machines;
- ability to obtain information or performs actions on behalf of the person who sent the agent out on the network;
- ability to, after doing its work, report the work's results to the requesting person; and
- ability to make decisions on behalf of the person it represents.

These characteristics distinguish the earlier examples as interesting and straightforward examples of mobile agents. They provide a way of marking a clear distinction between mobile agents and other distributed systems or simple Web crawling.

Given this characterization of mobile agent systems, the Jini system is not a mobile agent system. Although objects can be mobile in the Jini system, such mobile objects are not active in the sense of automatically getting a thread of control. Mobile objects in the Jini system can interact with the Java environment on the machine to

which they are sent, but they will more likely interact with other Jini services that reside in the network. A mobile object in the Jini system does not need the identity of some principal who sent the object or need to return to any point of origin. Finally, mobile objects in the Jini system don't need to be able to make decisions on behalf of their sender; indeed, one of the most common forms of mobile object in the Jini system simply forwards calls to the service it represents.

In fact, it is just these characteristics that have doomed mobile agent systems to the disreputable ghetto of distributed computing. The very characteristics that make a system a mobile agent system also make such systems difficult or impossible to deploy in the real world.

Using active objects means that mobile agent systems blur the line between the resources that a machine's owner controls

A system in which an agent can obtain information or perform actions on behalf of the person who sent it assumes a form of distributed authentication far more sophisticated than is generally available.

and the resources controlled by the agents that arrive at that machine. If an agent that arrives on a machine automatically gets a thread of control on that machine, an automatic allocation of computational resource follows. This makes little or no difference in cases when a small number of agents arrive and the machines are not resource-constrained. But this approach does not scale, either to resource-limited machines or to large numbers of agents.

A system in which an agent will interact with the environment on the machine to which it moves assumes a homogeneity of connected machines that is hard to take seriously in a large-scale network. When an agent arrives at a new destination, how does

it know with what to interact? Does it read the information that it needs from a file, and if so, where in the file system does that file reside? Does it contact some service on the machine? If so, it would need to know how to contact that service in a way that works on all the possible destinations to which it could be sent.

A system in which an agent can obtain information or perform actions on behalf of the person who sent it assumes a form of distributed authentication far more sophisticated than is generally available. For the agent to act on my behalf, I need to be able to delegate some or all of my identity to that agent. Although researchers have proposed security systems that have delegation, few have successfully implemented them, and the systems present difficult problems (such as canceling a delegation or insuring that delegations are bound in some fashion).

Furthermore, the security needed for a mobile agent system requires properties that general distributed-system security does not. Because the agent is being moved onto a host, it must be able to be authenticated. But when does that authentication take place? Because the agent is active, by the time the recipient knows of the agent's presence, the agent is running—perhaps able to damage the host machine. Also, how does the agent—which often carries sensitive information such as credit card numbers—establish trust in the machine on which it is going to run?

That the agent will return to its machine of origin after it has completed its work seems reasonable until we consider that networks—and the machines on them—fail. Suppose that the machine on which an agent was running crashes. How long does the entity that sent the agent wait for that agent's return before sending out another? Suppose that the network link from the current machine to the home machine is down. How long does an agent wait until it gives up the return trip? The failure models for agent systems do not seem well specified, yet any distributed system lacking a failure model that deals with these kinds of failures will not produce reliable applications.

Finally, the last characteristic of agent systems, that an agent can make decisions on behalf of its owner, seems to require the solution to many of artificial intelligence's problems. I have not seen many computer programs that make good decisions—certainly not any that I would be willing to have make decisions on my behalf. I am not saying that

such programs are impossible (I remain a skeptical agnostic on this subject). Furthermore, programs that come the closest to having this capability are all large. Yet, to be useful, mobile agents must be small entities that can easily move over a network. Even in moments when I feel most optimistic about AI's prospects, I don't see small, mobile objects that can run anywhere making such decisions.

For these reasons, I have never taken the claims of the mobile agents community terribly seriously. The Jini system was certainly not itself designed as a mobile agent system—or as a means for constructing such systems. It had much more modest goals. Ironically, however, systems based on the Jini technology and run in a Jini-enabled network can solve, in a rather straightforward way, a number of the problems that have plagued mobile agent systems.

### New directions in mobile agents

The first problem with mobile agent systems is that they employ active objects and therefore have automatic access to the resources of the machines to which their agents travel. A solution to this problem would be to move the agents from one machine to another as passive objects, but move them to a Jini service that would then oversee granting—at the appropriate time and priority—a thread of control to that agent.

Building such an agent host is simple. In fact, *The Java Tutorial Continued* gives it as an example for the Java Remote Method Invocation system,<sup>2</sup> under the guise of a compute server. The service itself is about 30 lines of code. Turning that service into a Jini service (which requires finding and registering with a lookup service and renewing leases on that registration) requires about 100 more lines of code.

Some might object that this is, in effect, what current active object agent systems do under the covers, so the stated approach does not differ from current practice. In one sense, this objection is correct; from a high-level view, the description of what happens is no different.

However, when some service—rather than a part of the underlying system—activates agents, the service's owner controls the policies the service uses. If I run an agent host service on my machine, I control the conditions under which it activates an agent. All the agent host must do is implement a known interface. How that interface is implemented

is up to me, not to a general system. If agent activation is a more general system activity, then I have given up control (at least in the agent systems I have seen). I could adopt a policy to activate agents immediately upon receipt, in which case I would fall prey to the kinds of agent storms discussed earlier. But I could also adopt a very different policy (likely after experiencing my first major service degradation) to ration the resources given to arriving agents. The main point is that the decision lies with me—the service's owner—and not with the agent system.

An agent system that runs in a Jini-enabled network also has a general technique for finding appropriate information when an agent moves from place to place. Because one of Jini's major goals is to allow service discovery over the network by programs, agents can use the mechanisms of lookup by Java type when they reach a new location to find the ser-

The first problem with mobile agent systems is that they employ active objects and therefore have automatic access to the resources of the machines to which their agents travel.

vices needed (or discover that the location does not have such services available). Once an agent gets a thread of control, the agent can find the local lookup service and then query it for objects that present the programmatic interface that the agent needs for its work.

Agent systems built within a Jini framework can also employ the Jini/Java RMI security model, which the Java Community Process is defining (<http://java.sun.com/aboutJava/communityprocess>). This security model's draft specification already includes mechanisms for mutual authentication of client and service and a time-based delegation model. More important for use within the agent community, the model provides a mechanism to let authentication occur before objects are reconstructed.

This model is layered on top of existing

Java security models, which (while not perfect) help insure that a machine will detect bad behavior by downloaded code before running it, and provide a mechanism to limit downloaded code's access to resources on the destination machine.

Finally, an agent system's implementation in the Jini technology model would begin to provide a failure model that the agent community might find useful. In particular, the Jini notion of a distributed object (that is, one that exists on multiple machines simultaneously) and the Jini programming model of a lease can provide solutions for the network failures current agent systems ignore.

Unfortunately, Jini technology does nothing to help with an agent's ability to make decisions on behalf of its sender. Indeed, much of the technology's design center was predicated on the supposition that software cannot make intelligent decisions. The system designers felt that the problems of distributed computing were hard enough and that they didn't need to add the AI problems into the requirements set. ■

### References

1. K. Arnold et al., *The Jini Specification*, Addison-Wesley, Reading, Mass., 1999.
2. M. Campione and K. Wallrath, *The Java Tutorial Continued*, Addison-Wesley, Reading, Mass., 1999.

**Jim Waldo** is a Distinguished Engineer with Sun Microsystems, where he is the lead architect for Jini, a distributed programming system based on Java. He is an adjunct faculty member of Harvard University, where he teaches distributed computing in the department of computer science. Prior to Jini, he worked in JavaSoft and Sun Microsystems Laboratories, where he did research in object-oriented programming and systems, distributed computing, and user environments. While working at Hewlett Packard, he led the design and development of the first Object Request Broker and was instrumental in getting that technology incorporated into the first OMG CORBA specification. He edited *The Evolution of C++: Language Design in the Marketplace of Ideas* (MIT Press, 1993), and was a coauthor of *The Jini Specification* (Addison-Wesley, 1999). He has MAs in linguistics and philosophy from the University of Utah and a PhD in philosophy from the University of Massachusetts, Amherst. He is a member of the IEEE and ACM. Contact him at Sun Microsystems, 1 Network Dr., MS UBUR02-306, Burlington, MA 01803; jim.waldo@east.sun.com.