The skinny on Jini

Jini and JavaSpaces bring true distributed computing to Java

Summary
Jini and JavaSpaces combine to turn your machines into a networked multicomputer. Together, these two technologies can even be used to create a personal supercomputer at work or in the home. Rawn Shah delves into the systems that will bring to life the applications of the next generation. *(3,200 words)*

By Rawn Shah

By now you've read about Sun Microsystems's July 20 announcement of the release of Jini, its new technology for distributed computing. Sun released the specifications for Jini to the public, giving everyone the chance to delve into distributed programming.

To understand Jini, you'll need to know about JavaSpaces, because the two are carefully woven together. Further on in this article, you'll see that JavaSpaces is actually more important than Jini, but let's start at the beginning: What are these two technologies and what do they do?

- Jini provides the distributed system services for look-up, registration, and leasing
- JavaSpaces manages features such as object processing, sharing, and migration

JavaSpaces needs Jini to perform its functions. To use a military metaphor, Jini acts as a quartermaster allocating equipment and armaments to the JavaSpaces troops. The troops use the allocated equipment to carry out their orders in battle.

In this article, we'll provide an architectural overview of what Jini and JavaSpaces do and how they work together. Then we'll compare and contrast the Jini-JavaSpaces technology with similar technologies. And, finally, we'll explain how this technology can be used and where it will be most beneficial.

The details on Jini

As Sun puts it, Jini allows you to create a "federation" of devices and software components in a single distributed computing space. Although the components work together to serve a common goal, they're still identified as separate components on a network. The federation provides services that are accessed by the various devices and software components. By services we mean communication mechanisms, users, hardware devices, storage devices, calculators (i.e., for your spreadsheet), notepads (for your BBS), and so on. The various members of a Jini federation share their services to get a job done. They communicate with each other through a set of Java interfaces known as the *service protocol,* and they locate each other using a special *look-up service.* The look-up system works in a multipoint network environment and the different components are ordered in a hierarchy.

The low-level communications used by Jini is the Java Remote Method Invocation (RMI) API as specified in JDK 1.2. There are a few basic changes between RMI in JDK 1.1 and 1.2 that better support the Jini system, particularly object activation, and distributed garbage collection. Jini requires RMI in the new JDK. The current specification documentation is slightly off when it says "extended RMI"; there are no extensions to RMI specifically for Jini other than what is already in JDK 1.2.

Each component has an ACL (access control list) describing which other components are allowed access to it. The component that sends out a service request, known as the *principal,* communicates with other components needed to perform a service.

The look-up service, distributed security system, RMI 1.2, and the discovery protocol (to register or locate a service using the look-up service) together constitute the *infrastructure* component of Jini.

The transaction system
This transaction system 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. The standard, however, does not indicate how this transaction system is implemented; that is left to the developer implementing the individual services.

The distributed event model
The distributed event model is an extension of the system used in JavaBeans. An *event* is a notification system for the services to indicate that an action is to be taken. The distributed event mechanism requires that Jini services register with others to receive notification. The Jini system handles the task of locating the services according to the hierarchy.

The leasing system, the transaction system, and the distributed events model together constitute the *programming model* for Jini. This, together with the infrastructure component and the services component describe the Jini component architecture. By itself, Jini works to create a sort of "plug-and-play" environment for all sorts of devices and software components on a network. JavaSpaces uses this architecture to create distributed computing systems.

JavaSpaces
JavaSpaces employs a simple mechanism for accessing and processing distributed objects. Simply said, a client application makes contact with a JavaSpaces server. The client asks for a certain type of object by sending a *template* describing what the object looks like. The space will then respond with the entry that best fits the template description. The client sends a `read` operation to make a copy of the entry within that space and work with it. Alternatively, the client can also do a `take` operation to take an existing entry from that space, effectively removing it from the list of entries. The client then processes the entry object as needed. Once completed, it can do a `write` operation to put that entry object back into the space, so that others can use it. If you have multiple clients looking for a particular entry, you may want to run `notify` to alert the system that you are waiting for the entry or to find out if someone else has written that entry.

That's it. A very simple concept in theory, but one that involves many services to handle the different parts. For example, each `read`, `write`, or `take` operation is a transaction and can be undone, so security is an issue. It's important to confirm that request clients have proper access rights to the entries. In addition, a mechanism needs to be in place to notify the different clients registered using the `notify` operation, and you need to be sure one client doesn't hog entries for itself. And guess what? Jini does it all.

Let's look at an example of how Jini works in an actual system. We'll use an inventory management system based on an idea brought up during the Conference on Electronic Commerce and Marketplace last April at the University of Austin. You're given an inventory management system in which each type of item in the inventory is managed by an agent. The agent is responsible for allocating ("selling") its items to the agents requesting ("buying") them. Essentially, with competing agents providing products of a similar type, you create a stock market-like exchange system with agents bidding for items to satisfy a request. A real-world example of this system might be the relationship between a car parts dealer and the various manufacturers selling the parts.

In a JavaSpaces system, a customer would go to the car parts dealer and put in a request for a carburetor for a 1989 4-cylinder Chevy Beretta. The order goes into the distributed JavaSpaces system (a "space"). It first enters the subsection of the space represented by the warehouse, or the warehouse space. This warehouse space then negotiates with the subsections of the car manufacturers (the car manufacturer's spaces) or parts distributors (the parts distributors' space). It issues a template describing the part. The various spaces respond with a list of available entries. The warehouse space server object compares them for price, features, reliability, and so on, and then issues either a `read` operation in the case of a car manufacturer (since it can make any number of identical copies of the part), or a `take` operation in the case of a parts distributor (since it has a given supply of the part); in truth, the car manufacturer space doesn't have an unlimited supply, but this is just an example. Let's say the customer wants to give back its defective carburetor, which is usually sent back to the manufacturer because it may be recoverable. In this case, the client `reads` the part from the car parts store warehouse space and `writes` it to the car manufacturers space.

JavaSpaces-Jini versus everything else
During our analysis and in talking to other developers, it becomes clear that Jini and JavaSpaces share similarities with a number of existing technologies. To give you a firm understanding, we'll make a few comparisons.

First, let's compare JavaSpaces-Jini with some of the other technologies from Sun's Java Software Division. JavaSpaces is a system for developing distributed applications. A distributed application usually consists of components running on several machines that communicate with each other to process the application. Think of this as a server-side application, where the server application is broken into several parts, each running on a different machine but still working as a whole.

Sun JavaServer
The Sun JavaServer (a.k.a. Jeeves) is simply an extended Web server with a backend programming interface that allows you to create servlets. It is directly comparable to the traditional CGI model, whereby accessing a link on a Web page can execute a program. However, servlets do not break into separate applications that automatically run across several computers to execute your program. You can make them do that, but you'll have to build the mechanism yourself.

Enterprise JavaBeans
With Enterprise JavaBeans (EJB) things get a little more complicated. EJB is a server-side mechanism that allows you to build a managed

network automatically, you may have an abstract notion of what JavaSpaces-Jini can do.

CORBA
JavaSpaces may also be confused with CORBA (Common Object Request Broker Architecture). The Object Management Group, an association of over 800 computer industry members, designed CORBA to span several component and/or object computing models and allow interactivity across them. CORBA defines a method to write client and server objects in a number of programming languages, including Java, C++, Smalltalk, Ada, and, believe it or not, Cobol. CORBA makes it possible for these languages to run on a number of different platforms and still communicate and interact with each other through a common object brokering infrastructure. All this works through the industry-accepted standard known as the Internet Inter-ORB Protocol (IIOP).

JavaSpaces provides server-side processing in a distributed environment. You are still passing objects around the network, but unless you use a Java object request broker like Visigenic Visibroker, Sun's Java IDL, or IBM Component Broker, these objects can only work in a Java environment. CORBA allows software development teams to develop using several languages or new code to be integrated with existing code by wrapping legacy code in CORBA objects. JavaSpaces actually fits into a separate brokering system for objects on the server side. Jini, for its part, works similarly to *CORBAservices,* a set of system objects for resource identification, leasing, transactions, and so on.

PVM architecture
Also comparable to JavaSpaces-Jini is the Parallel Virtual Machine (PVM) architecture from Oak Ridge National Labs. PVM is actually a set of programming libraries that allow you to build *multicomputers,* a set of independent machines on a network that share application processing, creating true heterogeneous network computing. Using PVM, you can create a distributed application, with each PVM daemon (server) automatically handling the job of network processing synchronization, communications, and fault tolerance. PVM participants may be Unix or Windows machines communicating over TCP sockets. There are programming interfaces to PVM in many languages including Java, C, C++, Perl, and Python.

JavaSpaces is similar to PVM. JavaSpaces is based on *Tuple space,* a concept created by Dr. David Gelertner of Yale University in his public domain Linda project. Tuple space is an abstract concept of a location, whereby processes can communicate in a distributed shared memory. The memory may be on a single machine, but more realistically is on several computers on a network -- each possibly a completely different computing platform. PVM and JavaSpaces solve the same problem of building heterogeneous network computers using different means and mechanisms.

Lucent's Inferno
Finally, let's compare JavaSpaces-Jini with Lucent Technology's Inferno. Inferno is a combined programming language (Limbo), virtual machine (Dis), and communications protocol (Styx) for small computing device models. It was designed as the core operating system for small multimedia and embedded devices such as pagers, cell phones, and set-top boxes. You can see that it shares many things in common with Java. Inferno itself is designed for a single device in a networked environment. It can locate resources kept on other Inferno devices across the network and communicate with them to exchange data. In this respect, Inferno has capabilities similar to Jini. Yet, it is lacking a mechanism to distribute the application processing across multiple machines.

For all other comparisons, it's important to note that JavaSpaces-Jini is not middleware. It does not create communication links between common commercial databases such as Oracle, Sybase, DB2, and so on, and object models like CORBA, DCOM, and SOM. It is, instead, the underlying technology that will allow us to build the new generation of application; one that can be distributed across a network of computers. This is the essence of network computing.

What can JavaSpaces-Jini do?
Together the JavaSpaces-Jini system allows you to create a network of semi-intelligent devices that can share resources and processing. It's easier to understand this concept with computers, printers, modems, and routers than it is with consumer devices like TVs, cable boxes, stereo systems, or toasters. In the consumer electronics arena, however, JavaSpaces-Jini can create new versions of these devices that plug-and-play on a network.

So what's the benefit in that? Well, admittedly, it doesn't make sense for all devices, but its use in TV, stereo, HVAC (heating, ventilation, and air cooling) systems could allow you to build an intelligent home. Your home computer could then dictate how to regulate the temperature in different rooms based on which ones are being used; your stereo could be preprogrammed to deliver music according to your mood; your coffee maker could automatically dispense coffee when you need it. In the commercial space, a manufacturing robot could be programmed to weld in different patterns; a building power management system could predict activity in different offices and manage power accordingly; a fax machine could be taught how to direct faxes by e-mail. Yes, there are mechanisms for doing these things with existing technologies such as X10, CEBus, and other specialized systems, but most aren't operating in a dynamically programmable environment, nor are they compatible with many other systems.

Application developers will find it possible to turn a LAN of hundreds of PCs into a parallel super-multicomputer and put the processing power of idle machines to use executing applications. You may, for example, have heard the recent news about the Avalon supercomputer built from 68 Compaq/Digital PCs with the Alpha processor. A more conventional example is the system used by HotBot; the search engine (promoted by *Wired*) is based on a specialized multicomputer system of over one hundred individual Unix workstations. Incidentally, HotBot is ranked as one of the fastest search engines in the world.

Don't think that Jini is a complete plug-and-play mechanism ready to order or that JavaSpaces is an existing product along the lines of a packet inventory control system. They are primitives to be used by programmers to create real applications for powerful, yet JavaSpaces brings distributed transaction processing forward in a way most programmers can understand and use.

Furthermore, not all applications flow in the nonsequential order available with JavaSpaces-Jini. While it could be said that most programmers are still unfamiliar with writing code in such a way, universities are catching up and bringing such advanced topics to the undergraduate level.

One important fact to remember is that although Jini has a cool name, by itself it is nowhere near as powerful as the JavaSpaces metaphor. Then again, to use JavaSpaces you have to use Jini.

For those who worry that this dual technology will cause fractures in CORBA, you shouldn't. Consider this a specific means to create a distributed server-side. You can still interface this powerful multicomputer server to the CORBA environment with Java IDL.

If Sun were to combine the power of JavaSpaces-Jini and Enterprise JavaBeans, you would have an EJB-JavaSpaces-Jini Server that would be a full generation ahead of anything else on the market. The power of such a system would bring multicomputing to the masses. As it is, Sun's partnerships with embedded systems developers and vendors could expand the role of the computer in the workplace and the home significantly. ■

Also this month in JavaWorld Go

Resources

- Sun's Jini home page
  http://java.sun.com/products/jini/index.html
- The real meat of the matter, the Jini specification
  http://java.sun.com/products/jini/specs/index.html
- Sun's JavaSpaces home page
  http://java.sun.com/products/javaspaces/
- The JavaSpaces specification
  http://java.sun.com/products/javaspaces/specs/index.html
- Read all about the Object Management Group and CORBA
  http://www.omg.org
- Lucent Technology's Inferno system
  http://www.lucent-inferno.com/Pages/Developers/index.html
- The PVM home page
  http://www.epm.ornl.gov/pvm/pvm_home.html
- This online book will familiarize you with PVM in detail
  http://www.netlib.org/pvm3/book/pvm-book.html
- "Do-It-Yourself Supercomputers", *Wired News,* July 3, 1998
  http://www.wired.com/news/news/technology/story/13440.html
- DATEK, an enterprise resource planning application vendor, announces the first distributed application integrated with Javaspaces and Jini
  http://www.internetwire.com/technews/tn/tn980741.htx
- Check out *News.com*'s report on Jini, "Sun makes another run at Windows," which debuted just before its release
  http://www.news.com/News/Item/0,4,24214,00.html

About the author

Rawn Shah is an independent developer and consultant in network computing systems, based in Tucson, Arizona. Reach Rawn at rawn.shah@javaworld.com.

-Very worth reading -Too long -Too technical
-Worth reading -Just right -Just right
-Not worth reading -Too short -Not technical enough

Comments:

Name:

Email:

Company Name:

Send data

If you have problems with this magazine, contact webmaster@javaworld.com
URL: http://www.javaworld.com/javaworld/jw-08-1998/jw-08-jini.html
Last modified: Monday, October 19, 1998