The promise at the heart of Java has always been code that developers can write once and run anywhere but with “smart card to supercomputer” scalability.

Sun has recently encountered criticism of Jini’s performance, primarily from Microsoft (mainly for technical reasons, but ostensibly because it inconveniently clashes with Millennium, Microsoft’s quiet project to develop its own version of a distributed operating system).

Nevertheless, Jini represents a sterling attempt by Sun and its supporters to help make Java truly portable while simultaneously addressing many of Java’s limitations as a language capable of running on the smallest handheld PC to controlling huge pieces of equipment, yet with few developers actually using it in the real world.

**What Is Jini?**

At its very simplest, Jini is an architecture for a network operating system designed for an array of electronic devices and software services. This takes it out of the domain of the typical PC or network computer into the realm of the network appliance.

Appliances can be anything from a telephone or pocket organiser or even something less obviously a computer - such as a remote control unit (imagine a remote control that changes its functions depending on what room you are in). The dream of Jini is to run on any appliance or computer - or rather, any device that supports Java.

Sun tries not to define Jini as a network operating system, preferring instead to call it a “networking infrastructure” running on top of Java to create a “federation” of virtual machines.

Jini can thus be logically divided into three loose areas: an infrastructure; a programming model; and Jini services. To the rest of us this looks awfully like a network operating system, but Sun is justifiably playing down its role as a true network OS primarily because Jini only provides a limited selection of core services. Also, some of Sun’s partners (in particular companies with their own network OS products such as Novell) are wary of helping Sun develop a potentially self-competing network OS.

**Differences**

Jini supports only very simple basic control structures. Objects communicate via the standard Java RMI interface through basic operations in what Sun has dubbed JavaSpaces. JavaSpaces is a vital part of Java that manages lower-level features such as processing objects and sharing them with other processes and so on.

To understand Jini, then, you’ll need to know about JavaSpaces, because the two are carefully woven together. JavaSpaces is actually more important than Jini, because although JavaSpaces manages features such as object processing, sharing and migration, Jini provides the distributed system services for look-up, registration and leasing.

JavaSpaces needs Jini in order to perform its functions. Jini allocates features and services to the JavaSpaces model, which then uses its allocation to carry out its coded instructions.

However, the most fundamentally important feature associated with Jini is that it provides Java with plug-and-execute capabilities. Sun hopes that a new device will gain instant recogni-
tion by the network and be granted access to any services for which it has authority. Dubbed JavaTones, this is similar to the same way that a telephone has a dialling tone that a telephone, fax machine and other devices use to recognise a service being available.

Federations

As I highlighted earlier, a Jini federation provides services that are accessed by the various devices and software components. Services can be anything from calculators, communication mechanisms, users, storage devices hardware devices, notepads and so on. These members of a Jini federation share their services to perform their ordained functions, communicating with each other through a set of Java interfaces known as the service protocol. They find each other using look-up services where the various components are arranged in a hierarchy.

The low-level communications used by Jini is dubbed the Java Remote Method Invocation (or RMI). You’ll find a full specification for this in the JDK 1.2.

Each component has an access control list, (which effectively describes what other components are allowed to access it) while the component that sends out a service request, is known as a principal, and communicates with other components needed to perform a service.

JavaSpaces

JavaSpaces employs a simple mechanism for accessing and processing distributed objects where a client application basically 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 an entry that best fits the template description. The client makes a copy of the entry within that space and works with it.

Alternatively, a client can also take an existing entry from that space, effectively removing it from the list of entries. The client then processes the entry object as needed, so security is an important issue and is managed by a security server because 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 various registered clients and you need to be sure one client doesn’t hog all of the entries for itself. However, whereas a traditional network OS relies on various layers of both clients and servers to manage these functions, often with unpredictable results, Jini does it all.

Why It’s Important

Network computing has become far too complicated. A typical client OS such as Windows 98 or Windows NT Workstation consists of between 10 and 20 million lines of code. The gargantuan complexity of these systems requires costly and resource-intensive computing devices that not only need to support their own standards but those standards required to talk to larger, more open networks, such as the Internet.

The same applies when adding new devices to an existing network. Despite Microsoft’s attempts to simplify this process by adding plug-and-play capabilities to its operating systems, adding, configuring and installing new drivers for new hardware is still an ongoing problem for most IT departments, creating potentially expensive logistical nightmares for systems administrators who must maintain a consistent computing environment for users. Sadly Microsoft’s attempts have only compounded an already snowballing problem.

Jini attempts to side-step this whole problem, because Sun claims it is able to decrease the cost and complexity of maintaining distributed computing environments while increasing the availability of services for a much broader range of electronic devices.

Reducing the total cost of ownership (TCO) of business networks is also becoming more important, with more businesses requiring access to networks and computing services. Indeed, after the resolution of many corporations’ Y2K problems, reducing TCO will be the next and equally-important task for IT departments over the next decade - and it is something Microsoft hopes to address independently with Windows 2000.

But beyond mere cost-cutting, Jini introduces a whole new generation of
network services and appliances to the home and the workplace. Services can be developed that can conform to the capabilities of a remote device and, where capabilities are limited, supply them dynamically from the network (such as moving from the office to the home). This is prohibitively difficult with the current generation of OSes - networked or otherwise - as it attempts to connect traditionally diverse embedded operating systems and indeed share the processing between them.

The common analogy Sun uses is an intelligent home network that controls everything from your computer to toaster - a distributed supercomputer, a telephone network that automatically routes calls to you at your computer, telephone, television and cellular device. To extend Sun’s analogy, we don’t always use the washing machine, central heating, VCR and so on, so such processing will be shared by Jini services amongst each device as and when it needs it.

Sun is not alone in trying to bring this level of simplified networking to the world’s networks however, and although Jini and JavaSpaces are best regarded as a double act that supports Java, they share certain similarities with a number of existing technologies. Intriguingly some of these competing technologies are from Sun.

**JavaBeans Server**

With JavaBeans (in this case, Enterprise JavaBeans), things get more complicated. Enterprise JavaBeans are client- and server-side reusable components, however the way they describe a server-side mechanism that allows developers to build managed server applications is analogous to Jini. Components are pre-developed pieces of application code that can be assembled into working application systems. Java technology currently has a component model called JavaBeans, which supports reusable development components. The Enterprise JavaBeans architecture logically extends the JavaBeans component model to support server components.

Each Enterprise JavaBeans server handles a pool of JavaBeans applications (each of which handles the requests from network clients). Enterprise JavaBeans builds an interface between a transaction processing server and its client applications that communicate over the network. Although an Enterprise JavaBeans server does not automatically distribute applications across several machines to do the processing, each Enterprise JavaBeans “entity object” (or session) is processed on the local machine. If you extend the Enterprise JavaBeans server

---

**Understanding Jini**

“The common analogy Sun uses is an intelligent home network that controls everything from your computer to toaster.”

---

**JavaServer**

The Sun JavaServer (a.k.a. Jeeves) is a set of extensions to a Web server employing a back-end interface that allows programmers to create what Sun has engagingly dubbed serverlets. Designed to supplant and eventually replace the traditional and difficult to standardise CGI model, where clicking on a link on a Web page executes a program, serverlets do not break into separate applications that automatically run across several computers to execute your program.

**Figure 1 - Clients perform operations that map entries (or templates) onto JavaSpaces applications. These can be singleton operations (as is shown with the upper client) or they can be contained in transactions (as with the lower client) so that all or none of the operations take place. A single client can interact with as many spaces as it needs to. Identities are accessed from the security subsystem and passed as parameters to the method invocation mechanism. Notifications go to “event catchers” which may be clients themselves, or proxies for a client (such as a store-and-forward mailbox).**
such that these EJB sessions or entities can span several computers on a network automatically, you may have an abstract idea of what Jini is capable of doing.

**CORBA**

JavaSpaces is often confused with CORBA (or Common Object Request Broker Architecture). CORBA defines a method to write client and server objects in a number of programming languages, (including C++, Java, Smalltalk, Ada and even Cobol). CORBA enabled these languages to run on various different platforms and still communicate and interact with each other through a common interface - or common object brokering infrastructure. This is managed by a standard interface known as the Internet Inter-ORB Protocol (IIOP).

Jini (and in particular JavaSpaces) provides Java with server-side processing in a distributed environment where objects are being passed around the network - however, unless you use a third-party Java object request broker, these objects can only work in a Java environment. The beauty of CORBA is that it allows software developers to develop using several languages and integrate it with existing lines of code by “wrapping” legacy CORBA objects. Jini, for its part, works similarly to CORBA services, a set of system objects for resource identification, leasing, transactions and so on. More details about the Common Object Request Broker Architecture can be found at http://www.omg.org.

**Inferno**

Inferno shares many similarities with Java. Designed by Lucent Technology, Inferno is a combined programming language (called “Limbo”), virtual machine (called “Dis”), and communications protocol (called “Styx”) and is intended for small computing device models.

Designed as a core OS for small multimedia and embedded devices (such as the new generation of video pagers, cell phones and set-top boxes), Inferno itself is designed for a single device in a networked environment. It can locate resources kept on other Inferno devices across a network and communicate with it to exchange data. In this respect, Inferno has similar capabilities to Jini, but Inferno lacks a mechanism to distribute the application processing across multiple machines. More details about Inferno can be found at www.lucent-inferno.com/Pages/Developers/index.html.

**Conclusion**

Jini does not create communication links between common commercial databases (such as Sybase, Oracle or DB2 databases), and object models like CORBA, DCOM and SOM. Instead, Jini is the underlying technology that allows developers to build the new generation of applications that can be distributed across a network of computers. This is the essence of network computing and it’s a vision Sun is keen to push.

But at present this is all formative and very much pie in the sky theorising. Jini has tremendous potential, but it still has to make the important conceptual leap from being a good idea to a product reality. To its credit it is already making leaps and bounds. Already Jini shares code across different versions of a JVM; this means that Jini may provide opportunities to innovate and still protect the infrastructure against incompatibilities (something that every network OS has so far failed to achieve).

Jini isn’t a complete plug-and-play mechanism ready to order, and JavaSpaces isn’t an existing vertical product (along the lines of a networked inventory control system).

Within Jini and Java there are language and OS tools to be used by programmers to create applications, but that’s all there is - the rest is up to programmers to make the most of Jini’s features. Its simplicity and its small footprint makes Jini (and thus Java) a good basis for the next generation of smart appliances.

As a component model, JavaSpaces brings distributed transaction processing forward in a way most programmers can understand and use. It is also important to remember that although Jini may steal the limelight from Java, it is nowhere near as powerful as JavaSpaces, but in order to use JavaSpaces you have to use Jini.

Further Reading

Basic information about Jini can be found at: java.sun.com/products/jini


Sun’s JavaSpaces home page can be found at: http://java.sun.com/products/javaspaces/

The Author

Clive Grace is a freelance writer, and can be contacted at clive.grace@itp-journals.com.
New Reviews from Tech Support Alert

**Anti-Trojan Software Reviews**
A detailed review of six of the best anti trojan software programs. Two products were impressive with a clear gap between these and other contenders in their ability to detect and remove dangerous modern trojans.

**Inkjet Printer Cartridge Suppliers**
Everyone gets inundated by hundreds of ads for inkjet printer cartridges, all claiming to be the cheapest or best. But which vendor do you believe? Our editors decided to put them to the test by anonymously buying printer cartridges and testing them in our office inkjet printers. Many suppliers disappointed but we came up with several web sites that offer good quality cheap inkjet cartridges with impressive customer service.

**Windows Backup Software**
In this review we looked at 18 different backup software products for home or SOHO use. In the end we could only recommend six though only two were good enough to get our “Editor's Choice” award.

**The 46 Best Freeware Programs**
There are many free utilities that perform as well or better than expensive commercial products. Our Editor Ian Richards picks out his selection of the very best freeware programs and he comes up with some real gems.