Chapter 2
Architecture Overview

The greatest challenge to any thinker is stating the problem in a way that will allow a solution.

—Bertrand Russell

We are living in a fast growing era of computing. Processor speed has multiplied many times. Network bandwidth has increased at a rapid pace every year. The memory capacity of disks and RAM has increased significantly. Having a 1- or 2-gigabyte RAM on one's desktop is no longer a dream. The most positive feature of all this improvement is the cost of these components—which has been spiraling downward over the years.

On the other front, computer networks have been expanding beyond proportions. With the advent of the Internet, we are now dealing with networks of more than a million fixed nodes. Added to this is the recent gadget revolution—fancy handheld devices such as cellular phones, pocket PCs, and PDAs—which, through wireless or with dial-up connection, become dynamic nodes. There are not many systems that were designed for such scalability needs. Today, due to the availability of smaller and cheaper processors, mem-

ory, and network cards, almost all devices are becoming intelligent by adopting one of every component—processor, memory, and networking card. With a few dollars, you can network-enable any device in your home: from a power switch to a washing machine, TV set, VCR, audio equipment, or microwave oven. The day is not far away when all your 911 calls may be handled in a completely different way through your home network (see Figure 2–1). The security camera on your home network could be activated by the emergency support center from a remote location.

So the computing theme today is pervasive, ubiquitous, and dynamic distributed computing. Currently, there is no technology that can handle such a requirement. Microsoft's Millennium Edition, Sun's Jini, and Hewlett-Packard's e-Speak are envisioned to fill this solution space.

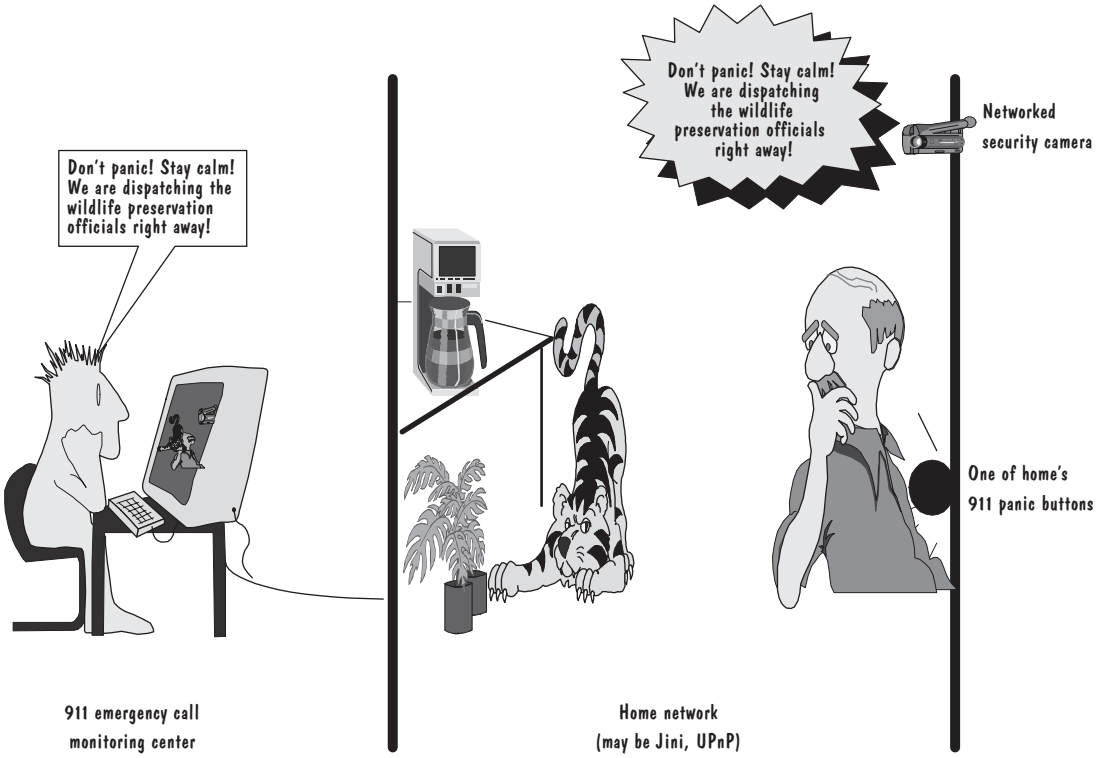

Figure 2–1 Future networking: looking beyond.

In this chapter we will look into the details of Jini's architecture—its vision, assumptions, components, component details, and its solution for solving the pervasive, ubiquitous, and dynamic distributed comupting problems.

Vision and Goals for Jini

As a dynamic distributed technology, Jini has the following vision and goals (see Figure 2–2):

- *To provide an infrastructure to connect anything, anytime, anywhere.* The vision of Jini is to provide an infrastructure that can help different network users to discover, join, and participate in any network community spontaneously.
- *To provide an infrastructure to enable "network plug and work."* The goal of Jini is to make any service joining the network available for other users without installation and configuration hassles. The vision is 0% installation and 0% configuration. It should be as easy as plugging a telephone into a telephone jack and using it—but it is not there yet. In fact, today's services are more operating system- and driver-centric. Even after downloading appropriate drivers and appropriate configuring, it is more a scenario of "plug and pray" than of "plug and play."
- *To support a service-based architecture by abstracting the hardware/software distinction.* Jini's vision is to provide an architecture centered around a service network instead of a computer network or device network. Jini's architecture simplifies the pervasive nature of computing by treating everything as a service. This service can be provided through hardware, software, or a combination of both. The advantage in abstracting this way enables the infrastructure to be designed to accommodate a single type of entity—a service. All protocols, such as joining or leaving the network, can be defined with respect to this service type instead of individual types. Such abstraction also helps in hiding the implementation of the service provider from the service requester.
- *To provide an architecture to handle partial failure.* A distributed architecture is not complete until it provides a mechanism for handling partial failures. Jini's vision is to provide an infrastructure and an associated programming model that can handle partial failures and help in establishing a self-healing network of services.

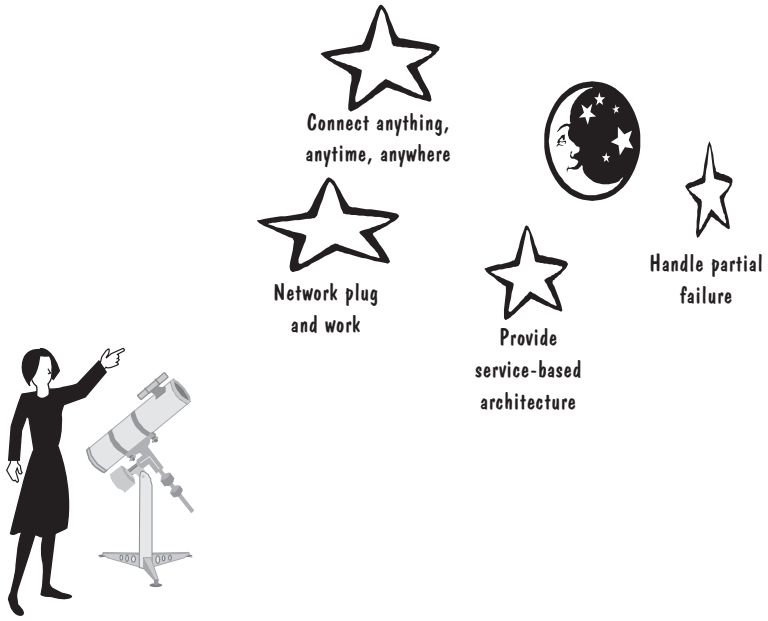

Figure 2–2 Jini's vision and goals.

System Assumptions

Jini's architecture is based on the following environmental assumptions (see Figure 2–3):

- *The existence of a network with reasonable network latency.* This is to ensure that network latency does not affect the performance of a Jini system because Jini relies heavily on Java's mobile-code feature.
- *Each Jini-enabled device has some memory and processing power.* For devices without processing power or memory, a proxy exists that contains both processing power and memory. This is a strong assumption because all network citizens are expected to have minimum computing capability, memory, and ability to communicate.
- *Each device should be equipped with a Java Virtual Machine* (JVM). The availability of different JVM footprints makes it easier to Java-enable any device.
- *Service components are implemented using Java*. This is an assumption for software components that would be joining a Jini community. All the service components should live as Java objects to facilitate the service requester to

download and run code dynamically. The point to note here is that Jini does not expect a Java service implementation but a Java wrapper.

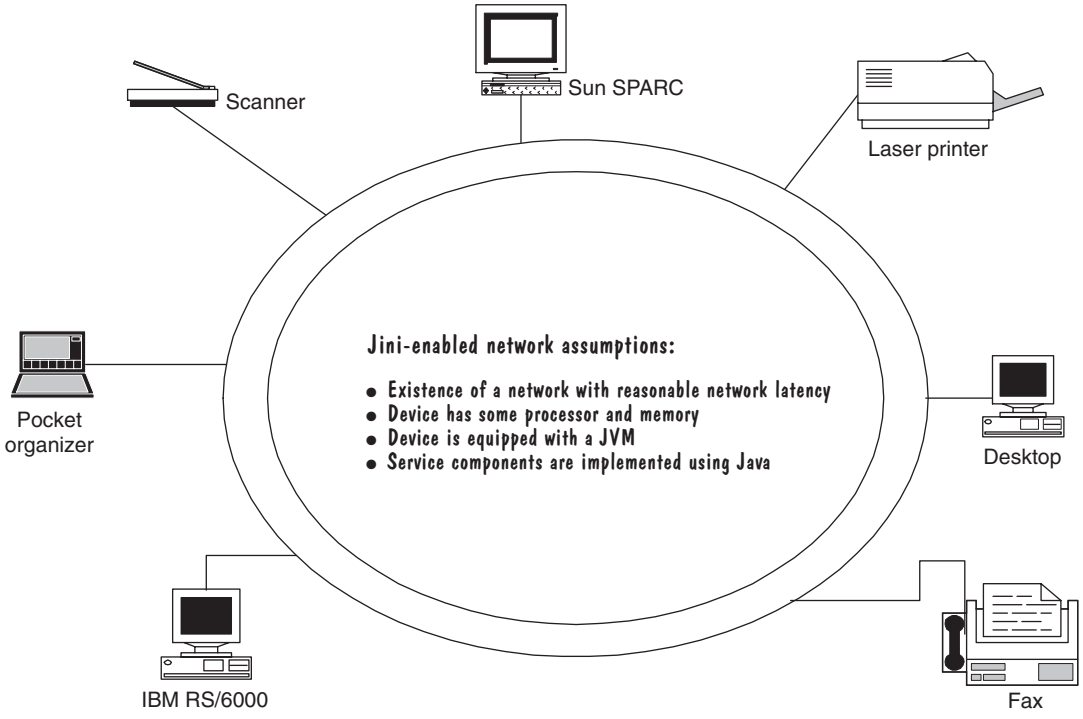

Figure 2–3 Jini's assumptions.

Are the system assumptions hard to meet?

The only assumption, which is very strong, is the expectation of Jini-enabled devices. These are devices with minimum computing capabilities, communicating capabilities, and memory that should host a Java Virtual Machine (JVM). This is fine for many devices, but it can cause problems for the numerous devices that are currently processor-less and driver-controlled. But the provision of a *proxy* (any device that has a processor, memory, and network capability willing to represent a processor-less device) makes this assumption easier to meet. By this approach, you can use a desktop computer to represent all your processor-less devices, such as printers, scanners, electric switches, washing machines, and microwave ovens, and also to control them.

System Architecture

The Jini architecture consists of the following components (see Figure 2–4):

- An *infrastructure component*, which enables building a federation of JVM
- A *programming model component*, which provides a set of interfaces for constructing reliable distributed services
- The *services component*, which forms the living entities and represents the offered functionality within the federation

Although the system has three component parts, the boundary between the parts is blurred. All three parts collaborate with each other, like a set of gears within a machine, to achieve the overall system objectives. In fact, the infrastructure and the services components are built using the programming model component interfaces.

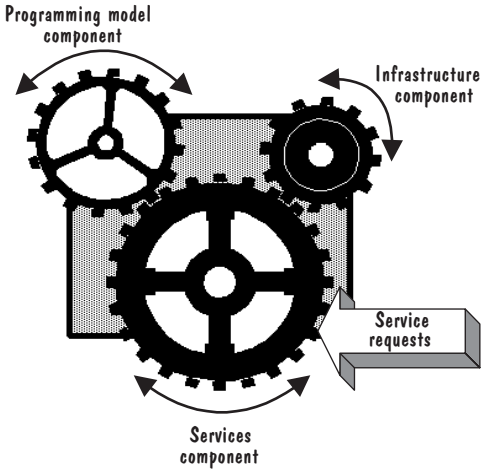

Figure 2–4 Jini architecture components.

Jini architecture is a Java-based solution for dynamic distributed computing. The Jini system extends the Java application environment from a single JVM to a network of JVMs. From that perspective, Jini can be seen as a network extension of the infrastructure, programming model, and services of Java application environment (see Figure 2–5). Jini utilizes most of the core Java technologies, such as RMI and JavaBeans, while adding additional functionality to meet the distributed/network nature of the system.

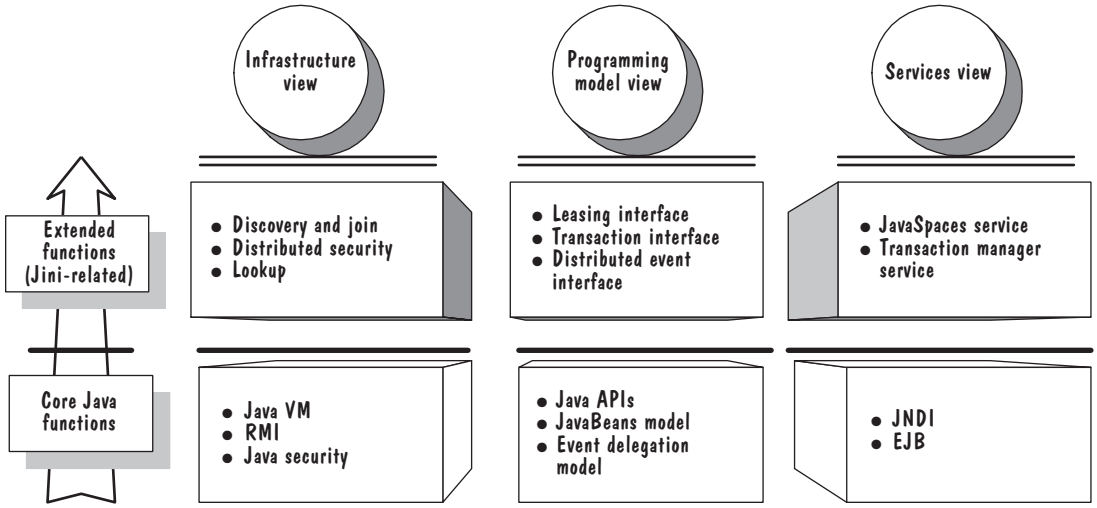

Figure 2–5 Java's extension for Jini.

How tightly are Jini architecture and Java coupled?

Two-part answer:

- Jini is tightly coupled with Java as an application environment and a programming model.
- Jini is not coupled with Java as a language.

This means that the service can be implemented in any language: C, C++, or JPython. But to participate in the architecture, it should be subjected to a compiler that can produce Java-compliant byte code. If not, it can be Java wrapped/Java-tized using Java native interface (JNI). In this way, even a legacy application can be Java-wrapped and can be made into a Jini service. To summarize: Jini architecture is not Java language-centric but Java application-centric.

System Components

The Jini system comprises three components: (1) the infrastructure, (2) a programming model, and (3) services (see Figure 2–6).

Infrastructure Component

The Infrastructure component is a core part of the architecture and its goal is to provide mechanisms for devices, services, and users to discover, join, and detach from the network. The Infrastructure component is composed of the following subcomponents:

- *Discovery and join protocol*, which defines the way that services discover, become part of, and advertise services to the other members of the federation.
- *Remote method invocation* (RMI), the distributed architecture environment that enables service proxies to be downloaded.
- *Distributed security model*, which provides the concept of security within the network. The distributed security model is an extension of Java's security model for distributed systems.
- *Lookup service*, which serves as a repository of services and helps network members to find each other within the Jini community. Entries in the repository are Java-compliant byte-code objects, which can be written in Java or wrapped by Java.

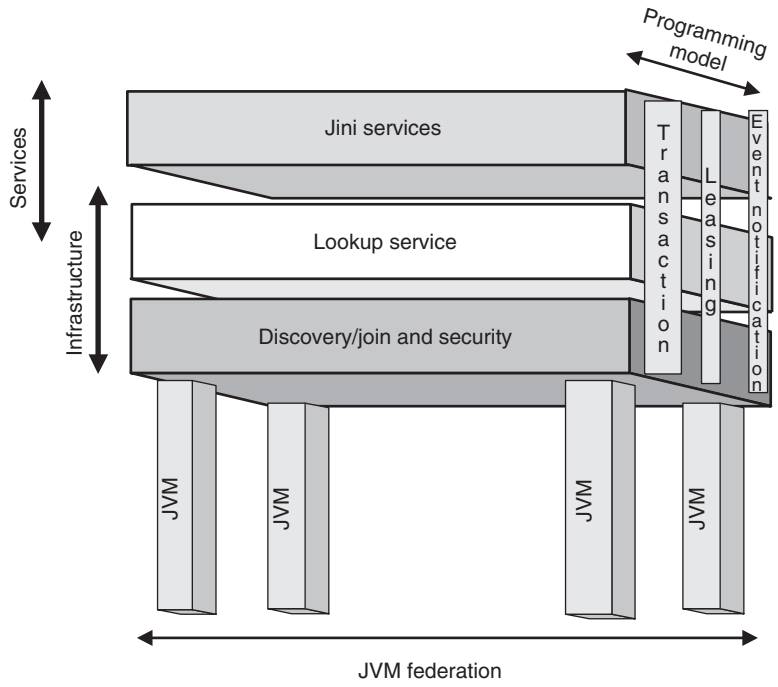

Figure 2–6 Jini components' detail.

Programming Model Component

The programming model is based on the Java application platform and its ability to move code between nodes. The programming model defines a set of interfaces, which taken together become the distributed extension of the Java programming model to form the Jini programming model. The programming model supports the following interfaces:

- *Lease interface*, which extends the Java programming model by adding time to the notion of holding a reference. This approach provides a renewable, duration-based model for allocating and freeing the resource references.
- *Event notification interface*, which extends the popular JavaBeans component event delegation model. This model allows an event to be handled by third-party objects and recognizes that the delivery of the distributed notification may be delayed.
- *Transaction interface*, which allows the system to handle object-oriented transaction handling. The interface does not define the actual mechanisms involved in the transaction but provides rules for the objects involved in the transaction. This approach provides freedom in choosing the preferred mechanics and individual object implementation.

Services Component

The services component represents an important concept within Jini architecture, and it denotes the entities that have come together to form the Jini community. The entities could be hardware, software, or a combination of hardware and software. The services are identified as Java objects within the system. Each service has an interface, which defines the operations that can be requested of that service. The interface also reflects the service type. A service is a composite entity and can be composed of other subservices. In fact, the lookup service—one of the subcomponents of the core Jini infrastructure—is implemented as a Jini service. Other constituents that form a part of Jini architecture and implemented as Jini services are:

- *JavaSpaces service*, which provides an optional distributed persistence mechanism for the objects within a Jini community
- *Transaction manager service*, which provides distributed transactions for the distributed objects

Interaction and Interdependence between Components

As stated above, although the system has three parts, each part has a specific role in the overall architecture and they work in tandem to achieve the overall system objective (see Figure 2–7). Both infrastructure and service components rely heavily on the programming model. For example, the lookup service makes use of leasing and event interfaces: JavaSpaces utilizes leasing, event, and transaction interfaces. In short, any component within the Jini system has to adopt the programming model recommended.

Theoretically, any service component is not forced to implement the programming model interfaces, but that is required for interaction with the infrastructure. For example, whether or not a service implements a leased service when it registers with a lookup service, it is leased. In scenarios where a service requester just invokes the service provider's method without sharing any resources or maintaining session information, leasing can be optional. An example of valid Jini service that does not use leasing and transaction is Jiro's log service. (Jiro technology provides tools and technology to reduce interoperability issues between storage systems, management software, and network devices.) Thus, the combination of infrastructure, services, and a programming model makes this architecture more reliable, dependable, and dynamic and helps to overcome the known issues with distributed computing.

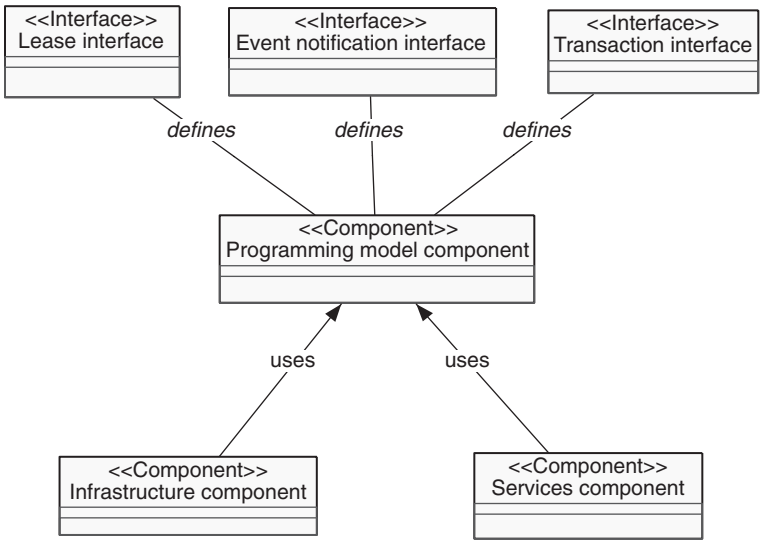

Figure 2–7 Component dependence: UML class diagram.

System Service Architecture

Let us now look at the way that Jini components work together to provide a dynamic distributed service network (see Figure 2–8).

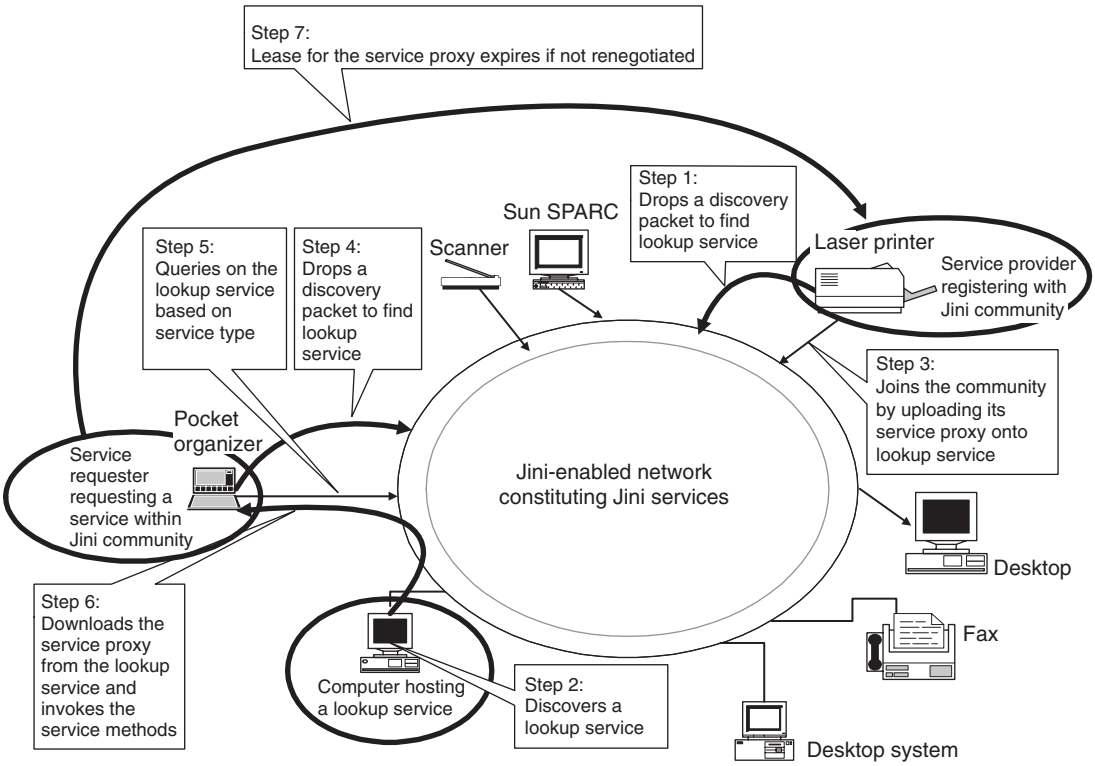

Figure 2–8 Technology at work.

Following is a walk-through of the steps that occur when a service provider registers with a Jini community, and when a service requester requests service.

Service Provider Registering within Jini Community

1. When a service is initiated into the network, it drops a discovery packet on the network, with a reference back to itself. The goal is to find one or more lookup services.
2. Any lookup service within the Jini community listens on a well-known port for the discovery packet and appropriately responds to the service provider.

3. When a lookup service within a network is discovered, the service joins the network by uploading all its characteristics into the lookup service. The service characteristics, its description, and its type are encapsulated as a proxy (Java) object, which is uploaded into the lookup service. This service is now available to any network citizen joining the community using the discovery and join protocol.

Service Requester Requesting Service within Jini Community

4. Any client (service requester) needing a service joins the community using the discovery protocol. In that process it locates one or more lookup services within the community.
5. After locating a lookup service, the client looks for the service in the lookup service based on its service type (Java interfaces).
6. Once the service is found, the client invokes the service, which involves moving the proxy code on to the client. Now the client can perform any operation on the service by calling its methods. This movement of the code between the lookup service and the client gives the service provider greater freedom in the communication pattern and makes it possible to maintain the integrity of the proxy code as it is supplied by the service provider.
7. Once the service proxy is downloaded, a service requester, depending upon its requirements, creates, negotiates, or terminates its lease with the service provider.

Summing Up

The Jini architecture consists of a core infrastructure component, a programming model, and service components that collaborate to provide a dynamic, distributed, self-healing network where services can discover and join spontaneously. It is a Java-based solution and can be considered as a network extension of the core Java application model. It is a simple, elegant solution for the complex dynamic distributed computing problem.

With this introduction to the architecture, let us now move to Chapters 3 to 5 to delve into the architectural component details.