Jini™ Architecture Overview

Jim Waldo

A Jini system is a Java™-centric distributed system designed for simplicity, flexibility, and federation. The Jini architecture provides mechanisms for machines or programs to enter into a federation where each machine or program offers resources to other members of the federation and uses resources as needed. The design of the Jini architecture exploits the ability to move Java language code from machine to machine and unifies under the notion of a *service* everything from the user of a Jini system to the software available on the machines to the hardware components of the machines themselves.

Introduction 1

This document describes the high level architecture of a Jini software system, defines the different components that make up the system, characterizes the use of those components, and discusses some of the component interactions. This document identifies those parts of the system that are necessary infrastructure, those that are part of the programming model, and those that are optional services which can live within the system. This document also discusses the reasons behind particular design choices.

1.1 Goals of the System

A Jini system is a distributed system based on the idea of federating groups of users and the resources required by those users. The overall goal is to turn the network into a flexible, easily administered tool on which resources can be found by human and computational clients. Resources can be implemented as either hardware devices, software programs, or a combination of the two. The focus of the system is to make the network a more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly.

A Jini system consists of the following parts:

- a set of components that provide an infrastructure for federating services in a distributed system
- a programming model that supports and encourages the production of reliable distributed services

- services that can be made part of a Jini federation and which offer functionality to any other member of the federation

While these pieces are separable and distinct, they are interrelated, which can blur the distinction in practice. The components that make up the Jini infrastructure make use of the Jini programming model; services that reside within the infrastructure also use that model; and the programming model is well supported by components in the infrastructure.

The end goals of the system span a number of different audiences; these goals include the following:

- enabling users to share services and resources over a network
- providing users easy access to resources anywhere on the network while allowing the network location of the user to change
- providing programmers with tools and programming patterns which allow the development of robust and secure distributed systems
- simplifying the task of building, maintaining, and altering a network of devices, software, and users.

The Jini system extends the Java application environment from a single virtual machine to a network of machines. The Java application environment provides a good computing platform for distributed computing because both code and data can move from machine to machine. The environment has built-in security that allows the confidence to run code downloaded from another machine. Strong typing in the Java application environment enables identifying the class of an object to be run on a virtual machine even when the object did not originate on that machine. The result is a system in which the network supports a fluid configuration of objects which can move from place to place as needed and can call any part of the network to perform operations.

The Jini architecture exploits these characteristics of the Java application environment to simplify the construction of a distributed system. The Jini architecture adds mechanisms that allow fluidity of all components in a distributed system, extending the easy movement of objects to the entire networked system.

The Jini infrastructure provides mechanisms for devices, services, and users to join and detach from a network. Joining into and leaving a Jini grouping is an easy and natural, often automatic, occurrence. Jini groups are far more dynamic than is currently possible in networked groups where configuring a network is a centralized function done by hand.

1.2 Environmental Assumptions

The Jini system federates computers and computing devices into what appears to the user as a single system. It relies on the existence of a network of reasonable speed connecting those computers and devices—10mbps in the general case. Some devices require much higher bandwidth and others can do with much less—displays and printers are examples of extreme points. We assume the latency of the network is reasonable, measured, at most, in seconds rather than minutes.

We assume that each Jini-connected device has some memory and processing power. Devices without processing power or memory may be connected to a Jini system, but those devices are controlled by another piece of hardware and/or software, called a *proxy*, that presents the device to the Jini system and which itself contains both processing power and memory. The architecture for devices not equipped with a Java virtual machine is discussed more fully in a separate document.

The Jini system is Java-technology centered. The Jini architecture gains much of its simplicity from assuming that the Java programming language is the implementation language for components. The ability to dynamically download and run code is central to a number of the features of the Jini architecture. However, the Java-centric nature of the Jini architecture depends on the Java application environment rather than on the Java programming language. Any programming language can be supported by a Jini system if it has a compiler that produces compliant bytecodes for the Java programming language.

1.3 Related Documents

This document does not provide a full specification of the Jini system. Each of the Jini components is specified in a companion document. In particular, the reader is directed to the following documents (note: Please check `http://www.java.sun.com/Products/Jini` for availability):

- The Java Remote Method Invocation Specification
- The Java Object Serialization Specification
- The Jini Discovery and Join Specification
- The Distributed Event Specification
- The Distributed Leasing Specification
- The Transaction Specification
- The JavaSpaces™ Specification
- The Jini Lookup Service Specification
- The Jini Device Architecture Specification

System Overview 2

2.1 Key Concepts

The purpose of the Jini architecture is to *federate* groups of devices and software components into a single, dynamic distributed system. The resulting federation provides the simplicity of access, ease of administration, and support for sharing that are provided by a large monolithic system while retaining the flexibility, uniform response, and control provided by a personal computer or workstation.

The architecture of a single Jini system is targeted to the workgroup, which can range in size from two or three users in small offices or homes to groups of up to 1,000. Members of the federation are assumed to agree on basic notions of trust, administration, identification, and policy. It is possible to federate Jini systems themselves for larger organizations.

2.1.1 Services

The most important concept within the Jini architecture is that of a *service*. A service is an entity that can be used by a person, a program, or another service. A service may be a computation, storage, a communication channel to another user, a software filter, a hardware device, or another user. A service may be more specific than this, for example providing a translation from one word-processor format to some other or translating a document from one language to some other.

Members of a Jini system federate in order to share access to services. A Jini federation should not be thought of as sets of clients and servers, or users and programs, or even programs and files. Instead, a Jini federation consists of services that can be composed together for the performance of a particular task. Services may make use of other services, and a client of one service may itself be a service with clients of its own. The dynamic nature of a Jini system allows services to be added or withdrawn from a federation at any time according to demand, need, or the changing requirements of the workgroup using it.

Jini systems provide mechanisms for service construction, lookup, communication, and use in a distributed system. Examples of services include devices such as printers, displays, or disks; software such as applications or utilities; information such as databases and files; and users of the system.

Services in a Jini system communicate with each other by using a *service protocol*, which is a set of Java interfaces. The set of such protocols is open ended. The base Jini system defines a small number of such protocols which define critical service interactions. For some of these protocols we will provide example implementations of services that use them.

2.1.2 Lookup Service

Services are found and resolved by a *lookup service*. The lookup service is the central bootstrapping mechanism for the system and provides the major point of contact between the system and users of the system. In precise terms, a lookup service maps interfaces indicating the functionality provided by a service to sets of objects that implement the service. In addition, descriptive entries associated with a service allow more fine-grained selection of services based on properties understandable to people.

Objects in a lookup service may include other lookup services; this provides hierarchical lookup. Further, a lookup service may contain objects that encapsulate other naming or directory services, providing a way for bridges to be built between a Jini lookup service and other forms of lookup service. Of course, references to a Jini lookup service may be placed in these other naming and directory services, providing a means for clients of those services to gain access to a Jini federation.

A service is added to a lookup service by a process called *discovery*, indicating that the presence of the service is discovered by the Jini system.

2.1.3 Java Remote Method Invocation (RMI™)

Communication between services is accomplished using *Java Remote Method Invocation* (RMI™). The infrastructure to support communication between services is not itself a service that is discovered and used but is, rather, a part of the Jini infrastructure. RMI provides mechanisms to find, activate, and garbage collect object groups. RMI also provides the infrastructure for multicast, replication, and the mechanisms for basic security and confidentiality.

Fundamentally, RMI is a Java-programming-language-enabled extension to traditional remote procedure call mechanisms. RMI allows not only data to be passed from object to object around the network but full objects, including code. Much of the simplicity of the Jini system is enabled by this ability to move code around the network in a form that is encapsulated as an object.

2.1.4 Security

The Jini security model is built on the twin notions of a *principal* and an *access control list*. Jini services are accessed on behalf of some entity—the principal—which generally traces back to a particular user of the system. Services themselves may request access to other services based on the identity of the object that implements the service. Whether access to a service is allowed depends on the contents of an access control list that is associated with the object.

2.1.5 Leasing

Access to many of the services in the Jini environment 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 as part of the service protocol: A service is requested for some period; access is granted for some period, presumably taking the request period into account. If a lease is not renewed before it is 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 user and the provider of the resource can conclude the resource can be freed.

Leases are either exclusive or non-exclusive. Exclusive leases insure that no one else may access the resource during the period of the lease; non-exclusive leases allow multiple users to share a resource.

2.1.6 Transactions

A series of operations, either within a single service or spanning multiple services, can be wrapped in a *transaction*. The Jini transaction interfaces supply a service protocol needed to coordinate a *two-phase commit*. How transactions are implemented—and indeed, the very semantics of the notion of a transaction—is left up to the service using the interfaces.

2.1.7 Events

The Jini architecture supports distributed *events*. An object may allow other objects to register interest in events in the object and receive a notification of the occurrence of such an event. This enables distributed event-based programs to be written with a variety of reliability and scalability guarantees.

2.2 Component Overview

The components of the Jini system can be segmented into three categories: *infrastructure*, *programming model*, and *services*. The infrastructure is the set of components that enable building a Jini federation, while the services are the entities within the federation. The programming model is a set of interfaces that enable the construction of reliable services, including those that are part of the infrastructure and those that join into the federation.

These three categories, though distinct and separable, are entangled to such an extent that the distinction between them can seem blurred. Moreover, it is possible to build Jini-like systems with variants on the categories or without all three of them. But a Jini system gains its full power because it is a *system* built with the particular infrastructure and programming models described, based on the notion of a service. Decoupling the segments within the architecture allows legacy code to be changed minimally to take part in a Jini federation. Nevertheless, the full power of a Jini system will be available only to new services that are constructed using the integrated model.

A Jini system can be seen as a network extension of the infrastructure, programming model, and services that made Java technology successful in the single-machine case. These categories along with the corresponding components in the familiar Java application environment are shown in Figure 1.

	Infrastructure	**Programming Model**	**Services**
Base Java	Java VM RMI Java Security Model	Java APIs JavaBeans ...	JNDI Enterprise Beans JTS ...
Java + Jini	Extended RMI Discovery Distributed Security Lookup	Leasing Two Phase Commit Events	JavaSpace Two Phase Commit Manager ...

Figure 1: Jini Architecture Segmentation

2.2.1 *Infrastructure*

The Jini infrastructure defines the minimal Jini core. The infrastructure includes the following:

- an extended version of the Java Remote Method Invocation system (RMI)—this is the basic mechanism of communication between components in a Jini system
- a distributed security system, integrated into RMI, which extends the Java platform’s security model to the world of distributed systems
- the discovery protocol, a service protocol that allows services (both hardware and software) to discover, become part of, and advertise supplied services to the other members of the federation

- the lookup service, which serves as a repository of services. Entries in the lookup service are objects in the Java programming language; these objects can be downloaded as part of a lookup operation and act as local proxies to the service that placed the code into the lookup service

The discovery protocol defines the way a service of any kind becomes part of a Jini federation; extended RMI defines the base language within which the Jini components communicate; the distributed security model and its implementation define how entities are identified and how they get the rights to perform actions on their own behalf and on the behalf of others; and the lookup service reflects the current members of the federation and acts as the central marketplace for offering and finding services by members of the federation.

2.2.2 Programming Model

The infrastructure both enables the programming model and makes use of it. Entries in the lookup service are leased, allowing the lookup service to reflect accurately the set of available services currently. When services join or leave a lookup service, events are signaled, and objects that have registered interest in such events get notifications when new services become available or old services cease to be active. The programming model rests on the ability to move code, which is supported by the base infrastructure.

Both the infrastructure and the services that use that infrastructure are computational entities that exist in the physical environment of the Jini system. However, services also constitute a set of interfaces which define communication protocols that can be used by the services and the infrastructure to communicate between themselves.

These interfaces, taken together, make up the distributed extension of the standard Java programming language model that constitute the Jini programming model. Among the interfaces that make up the Jini programming model are the following:

- the leasing interface, which defines a way of allocating and freeing resources using a renewable, duration-based model
- the event and notification interface, which is an extension of the JavaBeans™ event model to the distributed environment that enables event-based communication between Jini services

- the two-phase commit (or transaction) interfaces, which enable entities to cooperate in such a way that either all of the changes made to the group occur atomically or none of them occur

The lease interface extends the Java programming language model by adding time to the notion of holding a reference to a resource, enabling references to be reclaimed safely in the face of network failures.

The event and notification interfaces extend the JavaBeans™ and standard Java event model to the distributed case, enabling events to be handled by third party objects while making various delivery and timeliness guarantees. The model also recognizes that the delivery of a distributed notification may be delayed.

The two-phase commit interfaces introduce a light-weight, object-oriented protocol enabling Jini objects to coordinate state changes. The protocol differs from most transaction interfaces in that it does not assume that the transactions occur in a transaction processing system. Such systems define mechanisms and programming requirements that guarantee the correct implementation of a particular transaction semantics. The Jini two-phase commit protocol takes a more traditional object-oriented view, leaving the correct implementation of the desired transaction semantics up to the implementor of the particular objects that are involved in the transaction. The goal of the two-phase commit protocol is to define the interactions that such objects must have to coordinate such groups of operations.

The interfaces that define the Jini programming model are used by the infrastructure components where appropriate and by the initial Jini services. For example, the lookup service makes use of the leasing and event interfaces: Leasing insures that services registered continue to be available, and events help administrators discover problems and devices needing configuration. JavaSpaces utilize leasing and events, and also support the two phase commit protocol. The two-phase commit manager can be used to coordinate the voting phase of a transaction for those objects that support the two-phase commit protocol.

It is not required that the implementation of a service use the Jini programming model, but such services need to use that model for their interaction with the Jini infrastructure. For example, every service interacts with the Jini lookup service by using the programming model; and whether a service offers resources on a leased basis or not, the service's registration with the lookup service will be leased and will need to be periodically renewed.

The binding of the programming model to the services and the infrastructure is what makes a Jini federation a system as opposed to a collection of services and protocols. The combination of infrastructure, service, and programming model, all designed to work together and constructed using each other, simplifies the overall system and unifies it in a way that makes it easier to understand.

2.2.3 Services

The Jini infrastructure and programming model are built to enable services to be offered and found in the network federation. These services make use of the infrastructure to make calls to each other, to discover each other, and to announce their presence to other services and users.

Services appear programmatically as objects written in the Java programming language, perhaps made up of other objects. A service has an interface which defines the operations that can be requested of that service. Some of these interfaces are intended to be used by programs, while others are intended to be run by the receiver so that the service can interact with a user. The type of the service determines the interfaces that make up that service and also define the set of methods that can be used to access the service. A single service may be implemented by using other services.

The initial Jini services include the following:

- JavaSpaces, which can be used for simple communication and for storage of related groups of Java objects
- a two-phase commit manager, which enables groups of objects to participate in the two phase commit protocol defined by the programming model

2.3 Service Architecture

Services form the interactive basis for a Jini system, both at the programming and user interface levels. The details of the service architecture are best understood once the discovery and lookup protocols are presented.

2.3.1 *Discovery and Lookup Protocols*

The heart of the Jini system is a pair protocols called *discovery* and *lookup*, which occur at different times. Discovery occurs when a service joins a Jini lookup service, for example, when a device is plugged in; lookup occurs when a client or user needs to locate and invoke a service described by its attributes and possibly, other attributes. The following diagram outlines the discovery process.

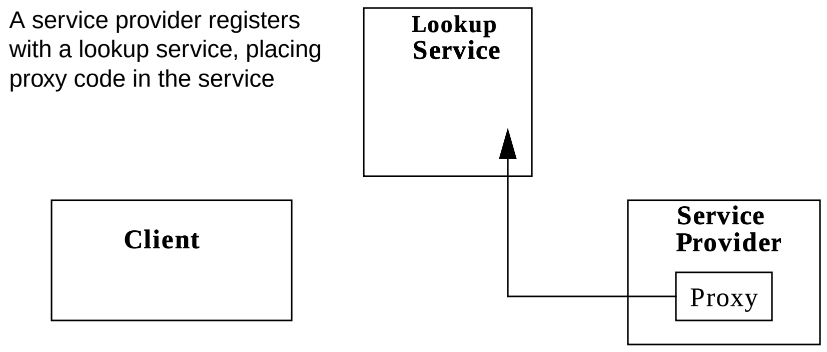
A service provider registers with a lookup service, placing proxy code in the service
Lookup Service
Client
Service Provider
Proxy

Figure 2: Discovery

Discovery is the process of adding a service to a Jini system. A service provider is the originator of the service—a device or software, for example. First, the service provider locates a lookup service by broadcasting a presence announcement. Then, a proxy for the service is loaded into the lookup service. This proxy contains the interface for the service along with any other descriptive attributes. Services must be able to find a lookup service; however, this requirement may be delegated to a third party. The service is now ready to be looked up and used, as shown in the following diagram.

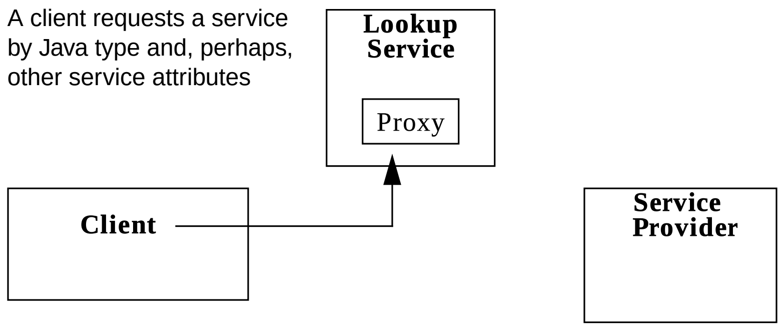
A client requests a service by Java type and, perhaps, other service attributes
Lookup Service
Proxy
Client
Service Provider

Figure 3: First Stage of Lookup

A client locates an appropriate service by its type—that is, by its Java interface—along with descriptive attributes which are used in a user interface for the lookup service.

The final stage is to invoke the service, as shown in the following diagram.

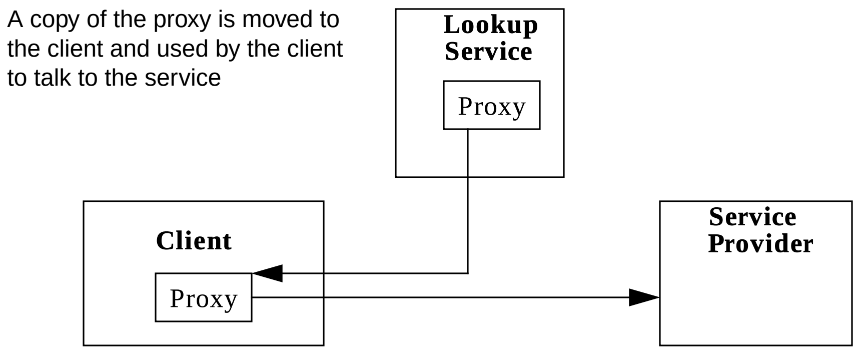
A copy of the proxy is moved to the client and used by the client to talk to the service
Lookup Service
Proxy
Client
Proxy
Service Provider

Figure 4: Invoking a service

The proxy code is loaded into the client. The proxy code may implement a private protocol between itself and the original service provider. Different implementations of the same service interface can use completely different interaction protocols.

The ability to move 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. This code movement also ensures that the proxy held by the client and the service for which it is a proxy are always synchronized, because the proxy is supplied by the service itself. The client only knows that it is dealing with an implementation of an interface written in the Java programming language, so the code that implements the interface can do whatever is needed to provide the service. Because this code came originally from the service itself, the code can take advantage of implementation details of the service known only to the code.

The client interacts with a service via a set of well-defined interfaces written in the Java programming language. These interfaces define the set of methods that can be used to interact with the service. At a rough level, these interfaces can be broken down into two distinct types:

- programmatic interfaces, which allow other software to interact with the service by making direct method calls on the service
- user interfaces, which present an interaction mechanism to end-users of the system

Programmatic interfaces are identified by the type system of the Java programming language, and services can be found in a lookup service by asking for those that support a particular interface. Finding a service this way ensures that the program looking for the service will know how to use that service, because that use is defined by the set of methods that are defined by the type.

Programmatic interfaces may be implemented either as RMI references to the remote object that implements the service, as local objects that provide all of the service locally, or as some combination. Such combinations, called *smart proxies*, implement some of the functions of a service locally and the remainder through remote calls to a centralized implementation of the service.

A user interface can also be stored in the lookup service. Such an interface will be identified by the Java programming language type `Applet` which allows the interface to be displayed by browsers or other user-interface tools. A user interface stored in the lookup service by a Jini service is an implementation that allows the service to be directly manipulated by a user of the system.

In effect, a user interface for a service is a specialized form of the service proxy that enables a program, such as a browser, to step out of the way and let the human user interact directly with a service.

An alternative form of lookup is the *peer lookup service*. A peer lookup service can be used when a client cannot find a lookup service. In such situations, the client can send out the same identification packet used by a lookup service to request service providers to register. Service providers will then attempt to register with the client as though it were a lookup service. The client can select those services it needs from the registration requests it receives in response and drop or refuse the rest. Such an approach works best in small, isolated networks.

2.3.2 Service Implementation

Objects that implement a service may be designed to run in a single address space with other, helper objects, especially when there are certain location or security-based requirements. Such objects make up an *object group*, An object group is guaranteed to always reside in a single address space / virtual machine when those objects are running. Objects that are not in the same object group are isolated from each other, typically by running them in a different virtual machine or address space.

A service may be implemented directly or indirectly by specialized hardware. Such devices can be contacted by the code associated with the interface for the service.

From the service client’s point of view, there is no distinction between services that are implement by objects on a different machine, services that are downloaded into the local address space, and services that are implemented in hardware. All of these services will appear to be available on the network, will appear to be objects written in the Java programming language, and one kind of implementation can be replaced by another kind of implementation without change or knowledge by the client.

2.4 For More Information

Further details on the Jini architecture are available in the detailed specifications.