JINI – Literature Review

Manish Kantamneni
manikant@indiana.edu

1. INTRODUCTION

This paper describes a new service oriented network-architecture called Jini. It defines a programming model which extends Java technology to enable the construction of secure, distributed systems. Jini technology can be used to build adaptive network systems. The technology ensures that these systems are scalable, evolvable and flexible.

The origin of Jini dates back to 1998. It was developed by Sun Microsystems based on the idea of federating groups of users and resources required by those users. The following sections in the paper explain its goals, architecture, programming models and virtualization of clusters.

2. GOALS

The following can be described as the end goals of the Jini architecture.

- Enabling end users to share services and resources over a network.
- Providing users easy access to resources anywhere on the network while allowing network location of the user to change.
- Simplifying the task of building, maintaining and altering a network of devices, softwares and users.

3. ARCHITECTURE

3.1 Overview

The Jini architecture (Figure 1) gains its simplicity from assuming that Java is the implementation language for its components. The ability to download and run a code is central to a number of features of the architecture. However the Java-centered nature depends on Java application environment than the programming language. Thus, any programming language can be supported by a Jini system provided that it has a compiler which produces compliant bytecodes for the Java programming language.

The architecture of a single Jini system is targeted to the workgroup. Members of the federation are assumed to agree on basic notions of trust, administration, identification and policy. This assumption enables Jini federating for larger organizations, but depends on trustworthiness of every system involved in the group. Looking into its three major components:

- Client - Anything that would like to make use of the Jini service
- Service Location – Anything that acts as a locator/trader/broker/ between service and client. It is employed for finding services in the distributed system.
- Service – Any entity that can be used by a client program or another service.

A Jini specification defines runtime infrastructure that resides on the network and enables adding, removing, locating and accessing services. Although the specification is independent of the protocol used for network communication, the current implementation is based on TCP/IP. Java’s socket support is used to send and receive object code that will be moved around among the three components. Typically, objects in one Java Virtual Machine (JVM) invoke objects on other JVMs using any protocol that supports Java Remote Method Invocation (Java/RMI) like semantics. The object code is serialized and marshaled over the wire.

3.2 Services

Jini Services use runtime infrastructure in Jini Specification to make themselves available when they join the network. They communicate using a service protocol, which is a set of interfaces written in Java. The infrastructure is also used by clients to locate and contact Jini services. It consists of support for discovery, join, and lookup. Let’s look at some of the important Jini services.

3.2.1 Lookup Services

Lookup services are the central bootstrapping mechanism for Jini-based systems. A lookup service maps interfaces indicating the functionality provided by a service to sets of objects that implement the service. A service is added to a lookup service by the discovery-join protocol pair. When a client wants to locate a Jini service to assist with some task, it consults a Jini lookup service. Objects in a lookup service may include other lookup services; this provides hierarchical lookup.

When a device is plugged into a network, it registers its services with a lookup service. The device and its services are now part of a federation.

<table>
<thead>
<tr>
<th>Jini Client</th>
<th>Service Locator</th>
<th>Jini Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jini Technology</td>
<td>Java Technology</td>
<td>Operating System</td>
</tr>
<tr>
<td>Network Transport</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 1. Simple Jini architecture while running.*
3.2.2 Runtime Services

The runtime infrastructure enables Jini services to register with lookup services through a process called discovery and join (Figure 2). Discovery is the process by which a Jini-enabled device locates lookup services on the network and obtains references to them. Join is the process by which a device registers the services it offers with lookup services.

The discovery process uses three types of sub-protocols, depending on the state a Jini component is in. They are:

- Multicast Request Protocol—used by a new Jini service to call anonymous lookup services on a LAN using multicast UDP.
- Multicast Announcement Protocol—used by lookup services to announce their presence on a LAN using multicast UDP.
- Unicast Request Protocol—used by Jini services contacting far lookup services using Unicast TCP.

The join process is used to register the Jini service proxy at the lookup service through the lookup service proxy obtained in discovery. Once registered, the lookup service contains an entry corresponding to its service Id and its attributes.

3.3 Miscellaneous

3.3.1 Security

Design of security model for Jini is built on twin notions of a principal and 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.

3.3.2 Leasing

Access to services 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 may conclude that the resource can be freed et al.

3.3.3 Transactions and Events

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. 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.

4. PROGRAMMING MODEL

Jini Infrastructure both enables the programming model and makes use of it. 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 constitutes 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 interfaces, which are an extension of the event model used by JavaBeans components to the distributed environment, enable event-based communication between Jini technology-enabled services.
- The 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 Jini transaction protocol differs from most transaction interfaces in that it does not assume that transactions occur in a transaction processing system. Such systems define mechanisms and programming requirements that guarantee the correct implementation of particular transaction semantics. The Jini transaction protocol takes a more traditional object-oriented view, leaving the correct implementation of the desired transaction semantics up to the implementer of the particular objects that are involved in the transaction. The goal of the transaction protocol is
to define the interactions that such objects must have to coordinate such groups of operations.

The binding of the programming model to the services and the infrastructure is what makes such a federation a Jini system not just a collection of services and protocols. The combination of infrastructure, service, and programming model, all designed to work together and constructed by using each other, simplifies the overall system and unifies it in a way that makes it easier to understand.

5. DYNAMIC CLUSTERING

Dynamic clustering is a technique to find entries in a log similar to a current situation. Essentially, it is a K-nearest neighbor algorithm. Let look into is a high-availability, scalable clustering technique using Jini technology. As a tool for dynamic networking, Jini technology found a niche as middleware in support of highly available and scalable enterprise systems that operate on clusters built on commodity hardware. We will start with service and application oriented cluster virtualizations.

5.1.1 Service-Oriented Cluster virtualization

Several techniques help transform the collection of independent computers in a cluster to a more or less unified computing device. A favorite clustering technique aims to virtualize cluster resources. Virtualization abstracts out hardware and operating system resources from a single node to the level of a cluster. Common cluster services, such as a file system or even I/O device access, are often defined by means of a network communication protocol. For instance, the Network File System (NFS) defines a network mounting protocol in terms of message exchanges between the NFS daemon process and its clients.

![Figure 3 - Service Virtualization](image)

5.1.2 Application-Oriented Cluster virtualization

Service interfaces on a cluster can define not only lower-level services, such as a file system, but also more specialized services in support of an application domain. A fundamental problem of system integration is the need to access various data sources, accept input from those data sources, translate that input into another data format, and then present the results to employees or systems that can act on those results.

We try to break up system integration processes into discrete tasks, such as those responsible for data input, rules execution, data transformation, or providing output. Services representing those tasks can then be distributed on cluster nodes and executed in parallel. Such distribution yields improved reliability. Distributing such services on a cluster, however, introduces the problem of locating instances of a service running on certain cluster nodes and accessing those services from other nodes. If we attempt to make them scalable by managing services at individual nodes, we increase cost of administration.

And that’s where we make use of the Jini technology. As a dynamic networking tool, Jini technology aids cluster management by its ability to expose implementations of Java interfaces as services on a cluster network, and by enabling clients to locate a service based on the service’s type.

5.1.3 Type-based service lookup with Jini

Jini technology makes type-based service definition and lookup explicit by requiring that Java interface type define a network service: A Jini service implements one or more Java interfaces, and the Jini service lookup infrastructure enables network clients to find services based on what Java interfaces a service implements. Jini technology's dynamic service lookup capability reduces cluster administration chores. Instead of looking in configuration files to locate services on the network, clients can use Jini lookup services to locate services dynamically.

Jini lookup services serve as central rendezvous points for Jini service lookup, and are themselves Jini services. Clients use the Java interface type, in addition to optional service attribute values, to find Jini services registered in lookup services on a cluster. Implementations of those interfaces, in turn, must register a proxy object with lookup services.

A Jini service's proxy (Figure 4) object plays a crucial role in how a client invokes that service. Such proxies are serialized Java objects that implement the service's Java interface. Thus, when a client locates a matching service registration in a Jini lookup service, the client retrieves that service's proxy object, and then invokes service methods on the proxy. The proxy may implement the functionality completely in the proxy, or interact with a remote service implementation via any communication protocol possible between proxy and remote server. As a result, the client does not have to care about what communication protocol to use with a service, because such protocols are hidden away into the service's proxy.

![Figure 4 – Role of Jini service proxy](image)
One way Jini deals with unreliable network is by leasing references to remote network resources to clients. A lease places a time limit on the client’s use of network resources, such as Jini service proxy or a lookup service. The lease-based access to resources defined in the Jini architecture fits well with the dynamic nature of a cluster. Since cluster nodes can fail at any time, a failed node’s registrations will be purged by lookup services. Also introducing a new cluster node involves merely installing and starting new service implementations on the new node. Those services will register their proxies with the cluster’s lookup services, making the new node available to clients without further administrative work.

6. CONCLUSION

The network architecture and programming models of Jini are explored. Application of Jini technology to virtualization of clustering shows its efficiency. A number of services and features like leasing, lookup and proxy make Jini quite suitable and reliable for dynamic clustering for large organizations.

7. REFERENCES