Title: **DISTRIBUTED COMPUTING USING JINI -Synopsis**

Author: Elango Sundaram

Abstract: A standardized distributed computational model, which will have feature sets to lookup, discover and use network resources in a dynamic, secure, robust, and reliable fashion enabling network-to-network and participant-to-participant communication is much needed. The networks and computational entities could be inherently unreliable and heterogeneous; spanning various operating systems, network architectures, computational power, load, storage, service capabilities and other implementation and administration details. Systems have to minimize down time, administration, and maximize reliability and interchangeability. JINI is a technology from Sun Microsystems in an effort to enable the above.

1.Introduction:

JINI offers a federated, distributed computational model where a service of some kind can publish itself, look up for other services, request, authenticate, authorize and use other services in fulfilling a need by another service or a person. JINI is enabled and based on Java and to be used requires a JVM and the Java Byte Codes. JINI builds up on the Java RMI architecture for communication and transfer of message and data; so bound by Java Specification and Java RMI Specification at the application level. The members on a federation can decide on kind of rights to resource and identification. The systems are built with the services as the building blocks. A service may be of any kind based on inherent capabilities.

2. JINI Overview:

The JINI system could be said to consist of following parts a) components that provide support for federating services in a distributed environment. b) Programming model to develop reliable systems. c) Services that could be made part of the federation and could be made to offer services to other members of the federation. The overall goal of the system is to turn the network into a flexible, easily administrable tool on which resources and services could be found by the human and computer clients. The resources could be hardware or software or computation facilitators etc. The overall goals are the following a) Enabling users to access and share resources from the network b) Providing access from any part of the network so that the provider could change location c) simplification of the task of building, maintaining, altering of network services.

The JINI environment provides for a dynamic creation of service, which could decide to join the network. Once it registers then it becomes available for the others to access. The service could provide security restrictions, lease time variations etc. The JINI system federates computes and the computing devices to something that appears as a single system to the user.

The basic unit of a JINI system is a service. A person, program or another service could use a service. A service may be a computational resource, hardware, software etc. A JINI system consists of services that could work together to achieve a task. Examples of a service could be traffic signal that is JINI enabled, a weather monitoring unit an air quality monitoring unit etc. And an example of a service usage could be one traffic monitoring system requesting another for traffic density. Another usage could be a JINI mobile weather monitoring unit discovering nearby weather monitoring units and requesting data on temperature, humidity etc.

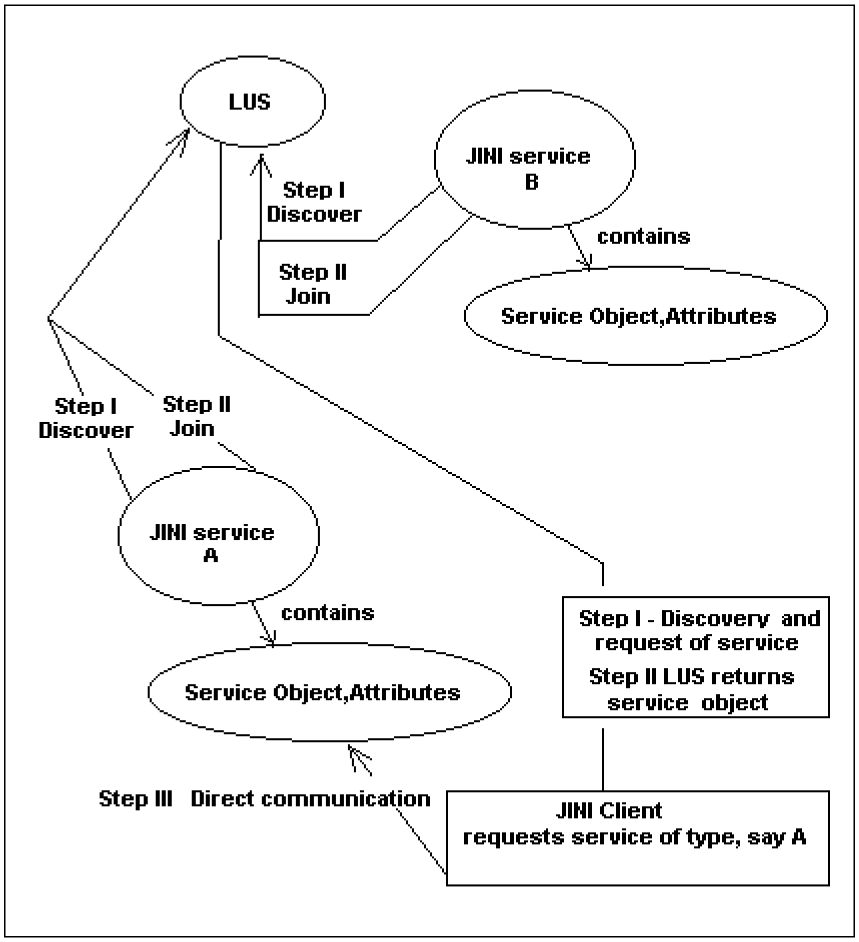
LUS
JINI service B
Step I Discover
Step II Join
contains
Service Object,Attributes
Step I Discover
Step II Join
JINI service A
contains
Service Object,Attributes
Step I - Discovery and request of service
Step II LUS returns service object
Step III Direct communication
JINI Client
requests service of type, say A

Fig 1 Overview of JINI service and JINI Client

The services in a JINI system are found and resolved by a central bootstrapping mechanism called the *lookup service*. A look up service can provide and map the information requested by the client with the ones available in the server. In other words, the look up service could give proxies that were provided by the services during registration to the client. Also its possible to provide for a description of a requested service during the request. A service becomes eligible to be found by doing two things a) discovery and b)join. The service will first do discovery and find the desired lookup server. After the lookup service is found, it could register itself through the join process.

The discovery and join process could be represented as follows (refer Fig 1). The service A and Service B are the services that wish to offer their services to the federation. In order to do so they first find out the LUS (Look up server) they wish to register with. Once they find the look up server, they register with it by informing it of their presence and giving it service attributes and proxy object. The service attributes refers to a set of parameters that could be used to find out a specific service; the service attribute could be as simple as a name or could be an object with a collection of parameters. The client who wishes to use a service finds the look up service first. Once a look up server is found, the look up server could be asked if it contains a service with some desired parameters. Assuming that the client requests service A, the look up server would return the proxy for the server for service A. After this the client could establish direct communication with the service A.

The JINI architecture uses Java Remote Method Invocation (RMI) for the purpose of communication. This means that the code downloading features of the *RMI* could be used. The client that requested the service A could get the Stub of the Server A RMI. After obtaining this *stub*, the client could make calls to the Server A methods, as if it were local method calls. All the features of RMI are available through JINI. The objects that need to be transferred between JINI networks need to be Serializable as in the RMI environment. Also a Java policy file could be set up at the client and at the server to set up permissions for the downloaded code. The typical security limitations could be the disk areas that the downloaded code could access, the kinds of permissions for the disk areas (like read, write, read-write) and the network resources it can access. A method dispatched by RMI may or may not execute in a separate thread. So the objects used in RMI need to be thread safe. To ensure garbage collection, RMI runtime keeps track of all references. The references are kept track by using a reference tracking counter. When all the references are discarded, an "unreferenced" message is sent to the server. When an object becomes completely unreferenced, then it becomes eligible for garbage collection. The JINI system additionally has the leasing mechanism, which reduces the possibility of any stale references in the system.

Each entity in a JINI network is a *service* also called *djinn*. A service is capable identifying and being identified by other services. An entity, when it starts up sends a proxy object to all the desired look up services by the process of discovery. The proxy is a Java object that could act as a publisher of the capabilities of the corresponding service. The services are based on the concept of *lease*. A service may be used based on the time period of the lease and the leased resource may be exclusive or non-exclusive (possibility of parallel or concurrent access). On the *time out*, the lease may or may not be renewed; on the latter case the resource is freed up. Wrapping a series of operations can create transactions. The event notification and handling can also supported in JINI architecture.

3. Discovery and Join in JINI:

The services that wish to participate in a JINI network also called djinns must have to obtain references to one or more look up services. A service has to use discovery mechanism to discover the reference to the look up service. Some of the terms used in the discovery and join part are as follows:

- A *host* refers to a single hardware connected to network, which may contain one or more Java Virtual Machines. A host that wishes to participate in a djinn need to have a JVM with all needed classes and a proper network protocol stack. If assuming IP is used for communication, then the host should have an IP address, support for unicast TCP and multicast UDP. The host also needs to provide for some kind of a mechanism like an HTTP server to enable automatic downloading of Java code.
- A *discovering entity* is the one which is planning to obtain references to one or more look up services
- A *joining entity* is the one that has a reference to a look up service and in the process of using other services or wanting to make others use its services
- An *entity* may be a discovering or a joining entity or a member of djinn.
- A *group* is a logical name for a collection of djinns. The group is an arbitrary string that's used as a name. Each look up service has a set of zero or many groups bound to it. The clients that would like to get a service could look for a look up service that contains the needed group.

Three kinds of discovery protocols used in JINI are the a) Multicast Request Protocol: This could be used to find look up services on LAN. The entities could use this to discovery neighboring look up services. b) Multicast Announcement Protocol: This is used to announce presence of a look up server on the network. c) Unicast Discovery Protocol: This is used for finding specific lookup in a WAN or for long use static situations.

4. Multicast Request Protocol:

Each entity that wishes to discover a djinn uses a multicast request service and a look up server uses multi cast response service. On the requesting side (say a JINI client) are the a) multicast request client that performs multicasts to discover nearby lookup services and b) multicast response server that listens for responses from those lookup services. Zero to many of these pairs could be existing in a single JVM. On the look up server side are the a) multicast request server that listens for incoming multicast requests and b) multicast response client that responds to callers. This client passes each requester a proxy that they can use to communicate with lookup service. The multicast request service is based on the multicast datagram facility of the network. In TCP/IP environment multicast UDP is used. (This is the reason a host has to be properly network configured with multicast datagram facility to enable discovery). The multicast response service is based on normal TCP/IP. The source IP address and port are obtained from the request and a unicast discovery is done. The multicast packet consists of the protocol version, port to connect to, count of heard look-ups (for which no response needs to be done), count of groups and the array of groups. The Multicast Request Protocol is briefly depicted in the following picture.

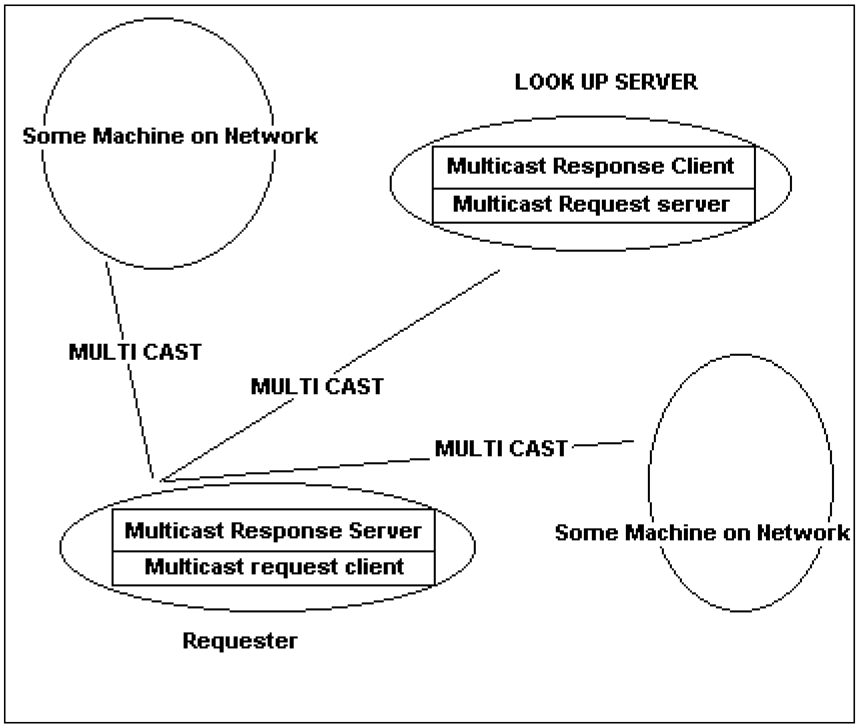

Fig 2 Multicast Request Protocol

5. Multicast Announcement Protocol:

The multicast Announcement protocol is used by the look up services to announce their presence to the network. The announcement will be done to the entities in the multicast span. The Multicast Announcement Protocol consists of two parts; the multicast announcement service and the multicast announcement client. The multicast announcement service resides in every entity that wishes to be notified and the multicast announcement clients are present the look up server that makes the announcement. The multicast announcement client is alive as long as the lookup server. The multicast announcement uses of multicast announcement packets which consist of the protocol version, host address for unicast discovery, port no, service ID or the originator, group count and the groups in originator. The multicast announcement protocol works like the following:

At the lookup server

- Create datagram socket object
- Make server side for the unicast discovery object
- Multicast announcement packets at intervals

At the listening entity

- Set up a list of known(previously heard entities)
- Bind datagram socket and listen for the multicast announcement
- On receipt of announcement check if the Service ID is already present; if not then perform unicast discovery and obtain reference server. Add the service ID to the set of known look up servers.

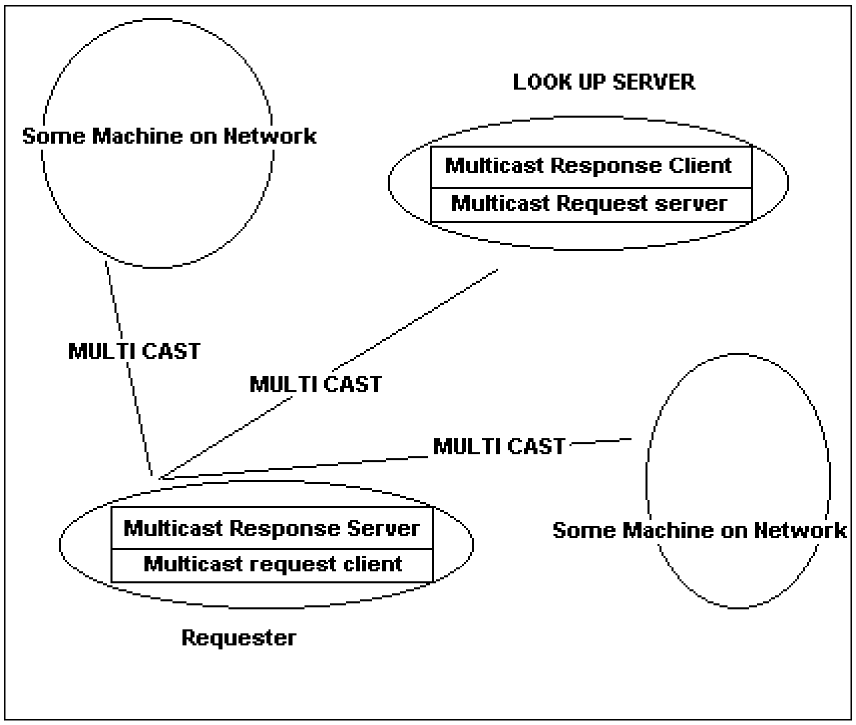

Fig 3 Multicast Announcement Protocol

7. Unicast Discovery:

The Unicast discovery protocol has a request and response and is used where the address of host is known and in situations where hosts are in a wide area network. The initiator sends a unicast request. The unicast server could respond by sending unicast response.

8. Join protocol:
The join protocol defines the sequence of steps that should be performed during the registration process. When the service starts up, it waits for a random time so as to prevent a packet flood during instances like power failure and system starts up. For each member of the group that the service needs to join, the service performs an unicast discovery and joins them. If the set of groups to be joined is empty, then a service does a multicast discovery and joins with each member that announces itself to be a member of the group that the service should join.

9. Distributed Leasing:
The term *lease* refers to the use of resources of a grantor by the holder. JINI uses the distributed leasing method to simplify the concept of resource allocation and standardized way of acquiring and canceling resources. The grantor and the holder could negotiate the terms of the lease in a standardized way. The distributed systems could tend to accumulate and gather up the unwanted information and dead resource data. The goals for the leasing system are a) to simplify time based resource allocation b) standardized acquiring and cancellation of resource c) Standard interfaces for programming. The characteristics of the JINI leasing system are:

- During the lease duration the grantor tries best to make the resource available for holder. The time period is negotiable and could be implemented based on application constraints
- Holder can cancel the lease during lease period.
- Holder could request lease renewal once again negotiable time period
- Lease could expire and no communication is needed on that. If a lease were not renewed after lease duration, then it would expire.

The lease is granted as absolute time duration rather that on clock times. This system is designed to avoid synchronization issues in distributed systems.

10. Distributed Events:
Distributed Event Notification refers to object in one machine showing interest and registering with another machine to get notified on a certain event happening in that machine. Distributed notification involves certain characteristics different form a single machine in the way of message delays, loss, transmission and request schedule and may other parameters. The Event notification is based on distributed lease model so that the notification is not for ever but time bound. The JINI distributed event notification aims at providing the following features:

a) Standard way (interface) to send notification
b) Specification of kind of information on the notification
c) Various degree of assurance for notification
d) Support different scheduling policies

The notion of *event listener* refers to the object that gets to receive the notification of the occurrence of an event. The *event generator* is the entity that could generate an event. The *remote event* is the entity that would be passed around as a reference to the event. The JINI event mechanism allows for third party event system enhancers like store-and-forward agent, notification filters etc.

11. Entries:
In the JINI system entries are used to do exact match up of objects in the distributed environment. Three types of operations are supported by the services that support entries. Thy are a) the store

operation that stores up entries b) the match operation that matches entries and c) the fetch operation that fetches the entries.

12. Distributed Transactions:

Distributed Transaction refers to grouped execution of operations, which can roll back and ensure *ACID* (atomicity, consistency, isolation and durability) properties in a distributed environment. The atomicity of a distributed transaction refers to complete success of all operations required in transaction in all different machines involved or a complete roll back in states of all the machines involved in the transaction. The consistency refers to the fact that the transaction should leave all participant machines in a consistent state. The isolation of a distributed transaction refers to the fact that the execution of distributed transaction should not be affected by intermediate states of other transactions. The durability of transaction refers to all participant machines are able persist their results. The distributed transactions are more difficult to implement that a single machine transaction as they have to take care of network latencies, availability of CPU cycles on different machines etc.

Two-phase commit protocol is used to ensure Distributed Transaction in JINI. In this, a manager will check the votes for participants. The votes may be prepared, not changed (read-only), or aborted. On a prepared or read-only vote from all participants, the manager will ask all participants to commit the transaction. In the event of even one participant failure, the whole transaction will be aborted.

A JINI transaction should make sure that the execution of sibling transactions concurrently, is same as executing them in order. Two phase locking is used to ensure this serializability. The resources are obtained using *locks*. The transaction that wishes to use a shared resource has to first acquire lock for the resource. The *lock* is released on completion of the resource. The locks could be read locks, write locks etc. The deadlocks in transaction are *not* guaranteed to be prevented or detected; but managers and participants are allowed to abort in order to break dead locks. There is also a possibility of *orphan* transactions. The orphans are transactions, whose ancestors are aborted. The JINI system cannot detect orphan transactions.

Author: Elango Sundaram
www.geocities.com/esundara