CSE6306
Jini™ Architecture

-Rahul Khot

Abstract:
The paper discusses the important concepts in JINI technology. The paper first discusses the idea of a service the central idea behind the Jini system implementation. The paper also demonstrates how Jini provides the classes and interfaces required for implementing a distributed plug and play system. A conceptual problem implementation is described which demonstrates the ease of use of Jini in developing distributed applications. A comparison with a competing technology CORBA is also presented.

1.Introduction:

The JINI system is a distributed system, which is a collection of users and hardware together. The idea of service is the central idea of the system. The JINI system provides:[10]

- An infrastructure for federating services.
- Programming support for production of reliable distributed services.
- Services which offer support to other members of the JINI federation.

As quoted from Sun's White paper "Jini Architectural Overview", " 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. It makes use of the Java Programming model by extending Java Application Environment from a single virtual machine to a network of machines. Other contemporary technologies include HAVi™, UpnP™ to name a few.

2.Basic Components

2.1 Service:

JINI totally does away with the client server paradigm.[11] In order to facilitate sharing of resources between two participants each in different networks, it associates a client with the services it provides. Therefore a service may be a computation, storage, communication channel to another user, a software filter, a hardware device, or another user. An example of a service could be a high-resolution printer, a computer having immense computational power etc.[8] Thus the JINI system can be viewed as a confederation of services working together providing services to other services in the JINI federation.[9]

2.2Lookup Service:

A lookup service is a central bootstrapping mechanism for the system. A JINI lookup service maps functionality provided by a service to a set of objects that implement the service.[8] The objects in a lookup service may have references to other services in the network, thus have a hierarchical structure. It is the responsibility to JINI Technology enabled device (JTED) to make its service objects available to other services in the network.[4]

2.3 Proxy Object and its attributes:

The object that implements all the interfaces provided by the remote service is known as a proxy object.[1] It is important because it helps in information hiding or encapsulation in the sense that the client can call only those methods for which the interface has been defined on the server. Thus the remote object knows how the work is to be done and how to connect back to the service if it is

necessary for the work to be done.[8] The attributes of the proxy object being added are used to distinguish between attributes objects of the sane service.

2.4 Client:
A client can be defined as an entity, which requests services. It may be a user, a workstation, a device etc.

3. How services are added?

The services use a proxy, which is an object having the same attributes as that of the service and communication instructions to register with the lookup services in the network. Thus a proxy object can be said to implement of the interfaces of the remote service. When a service connects to a network it uses the Discovery Protocol to search for a lookup service and registers itself with the service. Registering means that the service has sent its service proxy to all the services in the network. There are two distinct discovery protocols:[8]

- Multicast Discovery Protocol
- Unicast Discovery Protocol

These in turn evolve into three distinct protocols:[8][5]

3.1 Unicast Discovery Protocol:
Services or clients already know the location of the lookup service and contact it using URL based constant mapping to discover the lookup service. It is based on a TCP connection and is a simple request response protocol. [2]The following pseudocode shows an attempt by a client to contact lookup service located at a node mylookup.com at port 9000.It waits for a TCP connection at port number 4160.[8]

```
LookupLocator myLoc= new LookupLocator(jinni://mylookup.com: 9000);
myloc.discover();
```

3.2 Multicast Request Protocol:
This is used when the clients first start up and is not aware of location of lookup services. The client uses multicast to look up for services nearby in terms of the multicast reach.[2] The multicast radius which is measured in terms of Time to Live can be use to measure how many hops will the multicast packets travel outside the local sub-network and can be varied when doing the lookup. It uses two connections, one of them is used for multicast using UDP to send a lookup message to the network and the other is a TCP connection for the lookup service to connect back. [2]The protocol opens a socket after sending the information stated below to the required IP address at port no 4160 starts listening for a TCP connection on any of the available ports with the following information: -[8]

- IP address of the initiator.
- Port at which the client is listening.
- Set of groups at which the client is interested in connecting to.
- Service ID of the lookup services that the client already knows about.

The end of the process is triggered by any of the following things namely, a time out (spec 5 seconds), no of loop iterations (spec 7 iterations) and or the number of lookup services located (enough to cover all the groups required). The following code attempts to find out all services within the uta network.

```
String school[]={“uta”};
LookupDiscovery dept=new LookupDiscovery(school);
Dept.discover( );
```

As some of the ISPs may not support multicast within their networks. A utility known as tunnel Master enables clients in multicast disabled networks to receive multicasts.[8]

3.3 Multicast Announcement Protocol:

This protocol is used by the lookup services to announce their availability to other lookup services. This acts as an “I am here” signal to the other lookup services. This mechanism helps the client whose multicast requests have failed to find a suitable lookup service. Also this helps in keeping the failed lookup services up to date when they restart. The lookup service sets up a Unicast discovery handler listening at a TCP port and regularly send multicast announcement packets to the multicast group. This contains its own service id, information about host, port of its Unicast discovery server and a list of group serviced by the lookup service.

4. Join Protocol:

After successful discovery of a lookup service a client registers with the service it is returned a lease for a fixed amount of time.[5][8] [6]Along with the lease a unique Service id is returned. This is done by means of another protocol known as the Join Protocol. The Join Protocol is not a low level protocol as the Discovery Protocol suite.[8] This gives set the requirements if a service is to join and leave the JINI federation. In order to achieve this a service has to maintain certain persistent information about its state. This is composed of: -[8]

- Service ID.
- Attributes used during lookup, which describe the service.
- Set of groups that the service wants to join.
- Set of lookup services that the services should contact during discovery.

The join protocol ensures that the JINI system is always in a consistent state. A helper class called as jini.lookup.JoinManager handles a majority of the implementation details for the protocol.
When the lease expires the entry for the service is automatically removed from the lookup services it registered with. This lease can also be renewed after the period of the lease is over.[8]

5. Remote Method Invocation:-

The infrastructure for this communication is not by itself a part of the JINI Architecture. The communication between services is done using Remote Method Invocation (RMI).[10]This is not a service in itself which has to be discovered ,but is SUN’s version of a Remote Procedure Call(RPC). But it goes ahead than RPC and supports code mobility.[12] The objects can be serialized a passed over the network. At the other end they are again deserialized to recover the passed object. A more interested reader is asked to refer to RMI specs at Sun Microsystems Inc.

6. Security:

JINI is consistent with Java Development Toolkit (jdk) permission based security model.[The security model for JINI technology is based on the concept of: -

- Principal:-Entity which wants to access a service
- Access Control List: Content of this list verifies if an entity/service has access to the service it wants to access.

Security can be distinguished a security on the client side and security on the lookup side. On the lookup side it is necessary to have RMI Security Manager as without it would not allow service

proxy objects to be added to the lookup service. On the client side there is no sort of authorization and authentication mechanism.[8] JINI expects the service to perform its own authentication and access control at the application level wherever necessary.[8]

7. Leasing:

JINI mostly uses lease based access mechanism for most of the services in the JINI federation. A lease can be defined as a guaranteed access to a resource for a fixed period of time.[5] If the service wants to renew the lease it must indicate interest in the resource so that the lease can be renewed.
The services that are to be leased are as varied as JINI applications. They include: -[9]

- Physical services such as disk storage, physical memory, seats on an aero plane.
- Semi-tangible services like computational time or an hour long classroom booking.
- Completely intangible such as temporary bandwidth boost between two components and the probability of being served in the next 10 minutes.

A JINI system is supposed to be long lived through various component crashes and other reasons for failure, the leasing mechanism helps in this regard. The following section presents the played by distributed leases in the JINI model. The main focus of this section is as follows:-

- Granting and receiving leases.
- Renewing, canceling and expiring leases.
- The importance of leasing in JINI's resource management
- The implementation assistance that JINI provides for both lease receiver and lease grantor.

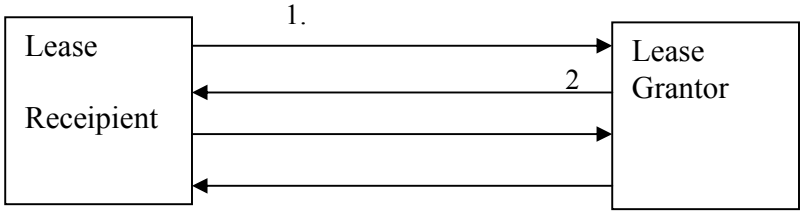
There is an exception to the notion of leasing all resources. The service object that is sent from the client to the lookup service is not associated. This is done for the following two reasons. [8]

- The look up service returns a copy of the service item to the client.
- The lookup service obligations within that that particular loop are complete on return of the proxy copy and the client uses this copy to communicate with the service thereafter.

We have to consider the lease from the point of view of the receiver and grantor. The role of the receiver is considerably simpler to implement and the receiver has to negotiate the duration of the lease and renew the lease before the duration expires.

7.1 Negotiating lease duration:

A lease is granted for a fixed amount of time. An absolute expiry time is set on the basis of the clock of the lease receipient. The negotiation process in JINI is as follows.The client requests a lease on a resource for a particular period of time. The granter considers the request and either grants it for the requested amount of time or for the time period it chooses. Thus the time period of the lease is decided entirely by the granter. The lease negotiation process is kept extremely simple in JINI, as a client may be a recipient of a number of leases. A complex negotiating protocol would increase the network traffic and make the client coding difficult.



1. Request from the lease recipient to acquire a lease for15 minutes.
2. Reply from the grantor saying lease granted for 10 minutes
3. Before the lease expires request form lease recipient to the grantor for a lease of 5 minutes
4. Lease renewed for 5 more minutes.

Fig 1 Lease Negotiation.[8]

The JINI specifications are not in a rigid API for leasing, but the software developer has to implement the method best suited fore the application.

7.2 Lease Renewal:

As seen from the above figure demonstrating lease renewal, it is the responsibility of the lease recipient to renew the lease as long as it needs the service. The initiation for lease renewal should be done before the lease expires so that network traffic and latency do not interfere. The renew lease time is not added to the old lease but a new lease is granted. If the lease grantor refuses to renew the lease then the earlier lease is still valid until it expires. This allows the lease recipient to make more attempts to renew its lease.

The grantor never grants a lease for an indefinite amount of time in order to avoid problems of distributed resource leaks and depletion. The grantor therefore grants leases for only a short period of time. This is in fact a design trade off., done to keep the state of the service as current as possible. This is done at the expense of increased network renewal processing and traffic. Granting leases for a long time will decrease the network renewal traffic but the service will wind up maintaining stale states for a longer period of time.

The LeaseRenewalManager is a JINI library class that helps in handling lease renewal.[5][3]] The LeaseRenewalManager ploys no role in granting the lease but just helps in managing the lease. The LeaseRenewalManager implements code to: [8]

- Manage the set of leases that are granted to the user by remote lease grantors
- Handle the renewal of all leases on behalf of the grantor.
- Perform the renewal of leases efficiently and in batches.

When an application creates an instance of LeaseRenewalManager it can delegate the task of lease renewal to it.[3] Since the instance is in the JVM it has same life as that of the application. This means when the application crashes the lease will not be renewed. This feature ensures that resources within the system are not blocked by applications, which have crashed and no longer need the lease.

The following diagram describes the functional components within the LeaseRenewalManager Class. [3][8]The lease holder first obtains one or more leases with one or more lease grantors, and then it creates an instance of the LeaseRenewalManager utility supplying it with a lease to handle with the desired expiration time. Then it continues to add leases to the managed set maintained by the LeaseRenewalManager specifying expiration time for each, and an optional listener for lease renewal failure. The LeaseRenewalManager goes to work and keeps performing the renewal with the actual lease grantor until the desired expiration time has been reached.[3]

In order to use LeaseManagerUtility to its fullest potential it is necessary to request an expiration time that is considerably latter than the lease time granted by the grantor. To assist in this the LeaseRenewalManager gives the programmer to specify the expiration time for the lease. Special values can also be specified in terms of Lease.ANY or Lease.FOREVER.[8][3]

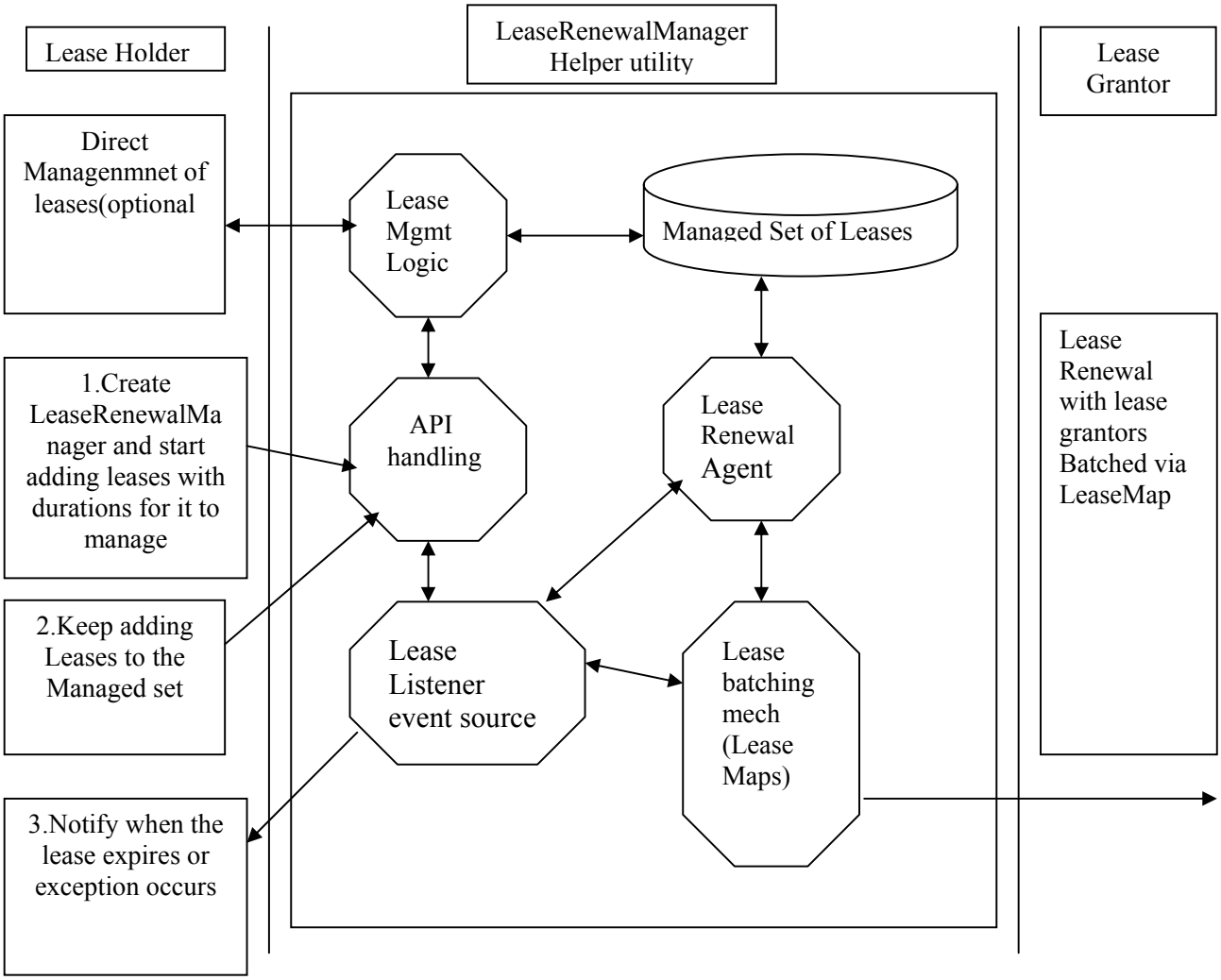


Fig 2 LeaseRenewalManager Helper Utility. [8]

Despite the above conveniences offered by the LeaseRenewalManager, the programmer has to do the following things: -[8]

- Requesting the lease
- Assist in lease persistence, in order to survive failure or unintentional restarts.
- Change the serialization format of the lease whenever necessary.

7.3 Granting Leases:

The role of the grantor is considerably difficult to implement. The management functions that the grantor has to take care of are as follows: -[8]

- Managing, allocating and leases resources on one hand, while granting and canceling associated leases on the other.
- Managing the leases themselves.
- Designing a lease allocation and renewal policy that is suitable for the application under consideration.

- Implementing the proxy object on the client that supports the lease interface.
- Supporting the proxy object and enhancing efficiency through LeaseMap based lease interface.

JINI does not specify how the resources should be granted nor does it provide any default implementation. Instead it provides interfaces and abstract classes to support the landlord model.

7.4 JINI Library support

It is not necessary for the grantor to use JINI library support but the object must support the "Lease " Interface. This lease interface contains both the local and remote methods, an implementation of the proxy object if needed. The JINI library provides a prebuilt proxy class:[7][5] com.sun.jini.lease.landlord.LandLordLease.

At the very root of the lease implementation library hierarchy is com.sun.jini.lease.AbstractLease, an abstract class that the JINI's Landlord proxy inherits from .The AbstractLease class handles functionality such as absolute verses relative time, and leaves the rest for the subclasses to handle. The following diagram illustrates relationship between AbstractLease, LandLordLease and the Landlord implementation of the programmer.

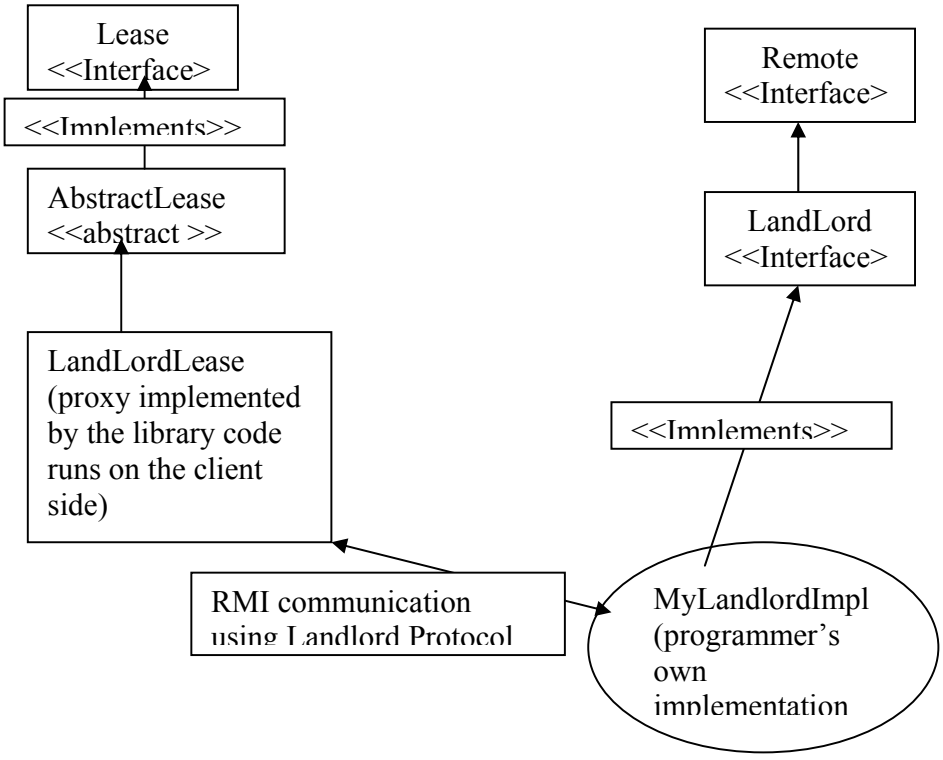


Fig 3 Relationship between AbstractLease , LandLordLease and programmers Landlord Implementation.[8]

Grantors supporting the LandLordLease proxy only needs to implement renew( ), cancel( ),renewAll() and cancelAll () methods. The LandLordLease proxy and its super class take care of the other methods of the Lease interface. The Lease Policy Manager manages renewal of leases and implements a specific policy. Each implementation can contain a different renewal strategy i.e. either it may renew for a fixed period of time or adjust it according to current level of resource allocation. The LandLord delegates renew() and renewAll( ) operations to it. It will have direct access to the lease resources to determine if a lease should be renewed. The leases themselves are managed by the Lease Manager. The Lease Manager manages the association of the lease with the leased resources.

The LandLord delegates cancel( ) and cancelAll( ) methods to it. The Lease Policy Manager and the Lease Manager share the task of managing the leased resources and communicate themselves to ensure consistent state and handling. Thus the LandLord effectively delegates its operations to the managers, which results in faster and more efficient management.

8. Events

The JINI Architecture supports distributed events. It allows for an object to register interest in event of another object and receive notification on occurrence of such an event. However when such a distributed event handling mechanism is implemented the unreliable nature of the network and infrastructure of a multi-machined network introduces a lot of complexity in the event handling mechanism. JINI's remote event API abstracts most of the difficulties JINI extends java's single machine event handling model to the distributed model. [10] The consumer as in single machine model creates a java object that implements a listener interface. Since the event notification may be done remotely ,the listener interface is made a remote interface via RMI .A specific remote (RMI) interface is defined as remote event listener.

```
public interface RemoteEventListener extends Remote,java.util.EventListener
{
  void notify(RemoteEvent theEvent) throws UnknownEventException,RemoteException;
}
```

8.1 Remote Events:

When remote notifications are sent over the network the following limitations come into picture.[8]

- Event delivery order may not be assured.
- Success of event delivery itself cannot be assured.
- Latency or delay in getting an event delivered may be larger than the actual time required for processing the event.

JINI resolves the problem of out of order delivery by associating unique event Ids and sequence numbers with each event. The event ID uniquely identifies the type of the event and /or the registration instance from the producer of the event. [8]Since a single consumer can be registered for different type of events with a single producer, this will enable the consumer to determine what a specific notation is about .The sequence will help in deciding the order of event occurrence. The sequence number has to incremented if the event is for the same producer consumer pair, but the increment is left as choice of the programmer. This is also essential in tracking missing events as certain applications may be sensitive to missing events.[8]

Due to the above-mentioned problems JINI batches events together should there be a likelihood of many events traveling between producer consumer pairs. This substantially helps in reducing the network traffic and also decreases the average latency time. [8]

8.2Third party event handling:

JINI also provides a facility for clients who do not want to handle the events themselves, or cannot handle or want the added value third party event handling brings. A third party as the name implies does event handling.[8][3] Use of the third party event handler also helps the client in having certain value adding operation like custom processing ,store and forward buffering .The figure illustrates the use of third party buffering.

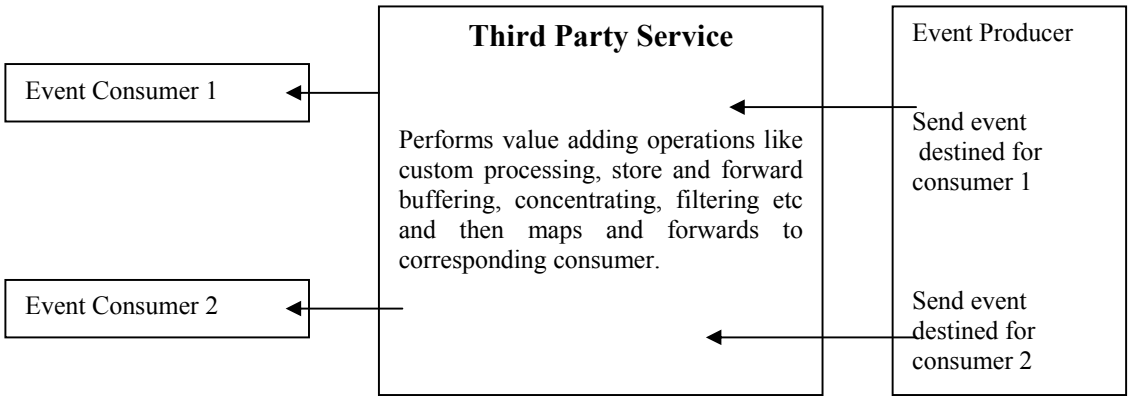


Fig 4 Third Party Event Handling

9. Distributed Transactions:

JINI provides support for a transaction mechanism and does guarantee properties ACID properties. [8]JINI also provides a mechanism for a series of operations either within a single service or spanning multiple services to be wrapped together in a transaction. The synchronization between the various services is done using Two Phase Commit Protocol.[11] This synchronization using the Two Phase Commit Protocol is done by the distributed Transaction Manager. JINI lease transactions and so a participant that crashes permanently either before or after reporting a successful preparation does not suspend the transaction indefinitely. The eventual expiry of the participant's lease lets the system know that it should clean up the system.[8]

10. Conceptual Implementation of a problem using JINI.

Consider the following Scenario

The central business area of a city has several restaurants and parking places distributed geographically. It's very hard to get a parking slot and a table during lunchtime. It's even harder to get a parking slot close to a restaurant with available tables.

Imagine a 'service' on the distributed computing environment that keeps track of available parking slots and tables in restaurants in a dynamic fashion. A user can contact this 'service' through his palm device (or cell phone). Basically the user tells the service where she/he is, what kind of cuisine she/he likes and a time limit. The 'service' processes the request and finds a parking slot – restaurant match so that the distances (driving and walking) and waiting time are minimized.

You can assume availability of GPS enabled devices, PCs, cameras, sensors etc.

Assumptions:

- It is assumed that all the devices wishing to be part of the JINI system have a JVM running on them. This though not a always essential for implementation, it is assumed , so that the underlying implementation is simplified.
- Each of the restaurants and its parking spaces and sensors within the same geographical area are considered to be a part of the same network.
- Lookup services are distributed on the basis of the network and clients within the same geographical area are serviced by the same lookup services or a group of them.
- Also it is assumed that all the networks participating share a mutual trust relationship so that we do not need to worry about security considerations.
- All such networks are TCP/IP networks.
- The cell phone has a JVM present on its chip.
- Each of the restaurants also keeps a track of the average serving time for the restaurant.

Let us first consider the various components that will be a part of the JINI system.

- Cell phone or Palmtop: A device carried by the mobile user so that he can send information about his location and specify the choices. This device is assumed to be GPS enabled.
- One computer at least in each of the restaurants so that it can keep track of the empty seats in the restaurant.
- One sensor minimum to monitor parking lots of each restaurant.
- Various computers having JVM provide lookup services to the each group of restaurants located within the same geographical area.

System at start up:
Consider the subsystem of the network which is composed of a group of restaurants, a number of sensors, a computer in each restaurant and the parking lot number for each restaurant. When the system is at start up each of the components of the subsystem needs to register with the lookup service in order to indicate its availability to the other services in the network. Since a particular lookup service has been assigned to a sub network unicast discovery protocol can be used by the components of the subsystem to connect to the lookup service. Thus after completion of this step all the available services would have the address of the required lookup service.

Join protocol:
After successful discovery of the lookup service the service must join the JINI federation. This is the job of the join protocol. While joining the JINI federation the restaurant service specifies the following parameters.

- Location in terms of longitude and latitude.
- Serving time.
- Restaurant.
- Seat number.
- Parking lot number.

Request of lease by the user:
A user using his cell phone or palm top, which has a JVM, sends out a multicast request protocol message. This message includes the eating preferences of the customer and time he has and his current location. The lookup service in proximity of the matches the attributes and if it has a slot available leases the service to the user.
When the user finishes his lunch the lease terminates and if needed the LeaseRenewalManager can extend the lease as needed.

11.Discussion:
11.1Pros:
The ability to plug a device into the network have it available to all the users is an extremely good feature.[11][13] This is accomplished by two simple protocols Discovery and Join. Once a device joins a JINI federation it can ask for services from any of the services present in the federation. Since service is a fundamental concept in the JINI system, the distinction between hardware and software is erased.[8] This results in freedom from implementation details of the service. It may be implemented in hardware or software or both. Since all the devices in a JINI system have a JVM in some way or the other the technology is highly platform independent.[15] The fundamental idea of leasing a service for a particular amount of time and having the lease receiver take care of renewing the lease helps in maintaining the state of the lookup service as current as possible.[8] This automatically incorporates self-healing capabilities into the network. The leasing mechanism is also

a great convenience for the administrator, as it makes sure that none of the services are being occupied by a client who has failed.[14]
The JINI system allows devices with no computational power be a part of the JINI federation like network sensors, printers etc. [4]The JINI system is extremely flexible and can adjust to changing topology. [14]JINI also supports object serialization and therefore it is possible to pass code from one location to another.[8]

11.2 Cons:
Although, JINI promises a lot of the above-discussed things, one of the major disadvantages is its java dependence. The JINI system aims for java to be a standard for application programming. But this is not possible as even though java offers a convenient programming environment to the programmer, java would be extremely slow for real time applications. Also java places a lot of overhead on the systems therefore making the computing platforms faster would be infeasible. JINI assumes that the following hardware trends will continue:[9]

- Processors will get smaller, faster and cheaper enabling the simplest device to have high performance processors.
- Memory will get larger, faster and cheaper.
- Network bandwidth will continue to expand.
- Storage devices will continue to increase in Capacity per unit cost

Although some of the assumptions as reasonable, a network latency of zero is too good to be true. The lease negotiation and renewal mechanism are built in such away that the send a lot of messages to each other resulting in large consumption of network bandwidth. In JINI there is no implementation of security on the service side. It is programmers task of having elaborate security mechanism built into the application. One of the strangest requirements is that JINI requires minimum three megabytes of code. This greatly hampers the ability to connect small devices into the network.[8]

12. Comparison with CORBA:[8]

CORBA is a middleware technology. JINI can be used with any middleware technology or layer.
CORBA is language independent .It uses OMG IDL and IDL as compilers for most of the popular programming languages whereas JINI relies on java. It uses java “interface” to describe the interface to remote services. The fundamental difference between JINI and CORBA is that JINI supports object serialization. This gives it the ability to move code from one client to another across. Secondly due to the concept of s lease JINI supports network plug and play, which CORBA does not support. CORBA focuses on the concept of the object whereas JINI relies on the concept of a service.

13. Conclusion:

Though JINI technology promises a lot, one of the major drawbacks is dependence on the Java therefore either the application program must be written in java or generate byte code. Also for some of the JINI enabled test networks no experimental results have been released, therefore leaving a large room for doubt about the feasibility of having “Jini™ Enabled Networks”.

References:

[1] Sun Microsystems Inc., Jini ™ Architecture Specification.

[2] Sun Microsystems Inc., Jini ™ Technology Core Platform Specification.

[3] Sun Microsystems Inc., A Collection of Jini™ Technology Helper Utilities and Services.

[4] Sun Microsystems Inc., Jini ™ Device Architecture Specification.

[5] Sun Microsystems Inc., Jini™ Lease Utilities Specification.

[6] Sun Microsystems Inc, Jini ™ Join Utilities Specification.

[7] Sun Microsystems Inc., Jini™ Lease Renewal Service Specification.

[8] Sing Li, *Professional Jini*, Wrox Press: Birmingham UK, 2000.

[9] Sun White paper, "Why jini technology now".

[10] Sun White paper, " Jini Architectural Overview".

[11] Sun White paper, "Jini Network Technology".

[12] Sun White paper , " Virtual Organizations, Pervasive Computing and an Infrastructure Networking at the edge".

[13] Sun White paper, "Jini Technology and Emerging Network Technologies "

[14] Arnold K., "The Jini Architecture: Dyanmic Services in a Flexible Network", Design Automation Conference, 1999. Proceedings. 36th, 1999, Page(s): 157 –162.

[15] Waldo Jim, "The JINI Architecture for Network Centric Computing" ,Communications of the ACM,1999 July Volume 42 no 7.