A Jini system (djinn) is a distributed system based on the idea of federating groups of users and the resources required by those users.

A Jini system consists of the following parts:

- Infrastructure components
  - Provides an infrastructure for federating services in a distributed system
- A programming model
Supports and encourages the production of reliable distributed services

- Services
  Offer functionality to any other member of the federation

The end goals Jini 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
- Simplifying the task of building, maintaining, and altering a network of devices, software, and users

---

**Jini System Requirements**

**Network**

- Reasonable speed
- Bandwidth requirements vary among devices
- Reasonable latency

**Jini is Java-based**

- Any language that produces Java byte codes can be used

**Devices**

- Some memory and processing power to run Java or
- Connected to a device (proxy) that contains the memory and processing power

---

**Jini System Overview**

**Terms**

**Services**

- A service is an entity that can be used by a person, program or another service
- A service can be:
  - a computation,
  - storage,
  - communication channel to another user
  - a software filter
  - a hardware device
  - another user
Lookup Service

Is used to find services in a djinn (a Jini system)

Leasing

A lease provides access to a service for a fixed time period
If a lease is not renewed at the end of the lease period, then the user and provider of the service can free all resources connected the lease

Transactions

Provides protocol for a two-phase commit process for managing state changes between objects in a Jini system

Events

Distributed events
Distributed version of listeners or Observer-Observable

Component Overview

<table>
<thead>
<tr>
<th>Infrastructure</th>
<th>Programming Model</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Java</td>
<td>Java VM</td>
<td>JNDI</td>
</tr>
<tr>
<td></td>
<td>RMI</td>
<td>Enterprise Beans</td>
</tr>
<tr>
<td></td>
<td>Java Security</td>
<td>JTS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JMS</td>
</tr>
<tr>
<td>Java + Jini</td>
<td>Discovery/Join</td>
<td>Transaction Manager</td>
</tr>
<tr>
<td></td>
<td>Distributed Security</td>
<td>JavaSpaces</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lookup service</td>
</tr>
<tr>
<td></td>
<td>Lookup</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Discovery

Service provider looks for a lookup service to register itself

Join

Service provider is registered in the lookup service
Service provider registers:
  A remote reference to itself
  Descriptive attributes about the service

Lookup

Client requests a service by Java type and/or by attributes
Client receives remote reference to the service
### Jini 1.0 Service Application Names

<table>
<thead>
<tr>
<th>Service</th>
<th>Application Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup</td>
<td>reggie</td>
</tr>
<tr>
<td>Transaction</td>
<td>mahalo</td>
</tr>
<tr>
<td>JavaSpaces</td>
<td>outrigger</td>
</tr>
</tbody>
</table>

---

## Basic Device Architecture Examples

(Note: Images are taken from Jini Distributed Leasing Specification.)

### Devices with Resident Java Virtual Machines

![Diagram of Devices with Resident Java Virtual Machines]

- A Jini device does not need to support the entire JVM. One should be able to configure Personal Java or Embedded Java to support Jini. Sun’s documentation is not clear on this point.

### Devices Using Specialized Virtual Machines

- A Jini device does not need to support the entire JVM. One should be able to configure Personal Java or Embedded Java to support Jini. Sun’s documentation is not clear on this point.

### Clustering Devices (physical option)

![Diagram of Clustering Devices (physical option)]

### Clustering Devices (network option)

![Diagram of Clustering Devices (network option)]
Jini Software Services over the Internet Inter-Operability Protocol

IIOP (Internet Inter-Operability Protocol) is a Corba standard for communication between Corba orbs

Sun has RMI-IIOP which implements a subset of RMI over IIOP

The Sun documentation states:

appears that the Jini Lookup service interfaces are in conformance with the RMI over IIOP subset

For more information about RMI-IIOP see:

For more information about the issues of doing RMI over IIOP see:

For more information about Corba see http://www.omg.org/corba/beginners.html

---

Distributed Leasing

Distributed systems must deal with:

- Failure of part of the system
- Accumulation of outdated and unwanted information

Jini’s Distributed leasing:

- provides a simple way of indicating time-based resource allocation or reservation
- provides a uniform way of renewing and canceling leases

A lease is

"a time period during which the grantor of the lease insures (to the best of the grantor’s abilities)"
that the holder of the lease will have access to some resource.

Lease duration

Can be solely determined by lease grantor
Negotiated between grantor and holder of the lease

Lease holder can:

Request a renewal of the lease
Cancel a lease

When a lease is cancelled the grantor of the lease can clean up any resources allocated to support the lease and "obliges the grantor of the lease to not take any action involving the leaseholder that was part of the agreement that was the subject of the lease".

A lease can expire

"Both the grantor and the holder are obliged to act as though the leased agreement is no longer in force."

Lease time is not measured in absolute time, but in duration. Clients should use the time they requested the lease as the beginning of the lease duration.

Basic Lease Operations

Interface net.jini.core.lease.Lease

Static Fields

//Length of lease
static long FOREVER = Long.MAX_VALUE;
static long ANY = -1;
//Formatting of serialized Lease object
static int DURATION = 1;
static int ABSOLUTE = 2;

Methods

long getExpiration();
returns the expiration time (in milliseconds from the beginning of the epoch)

void cancel() throws UnknownLeaseException, RemoteException;

void renew(long duration) throws LeaseDeniedException,
UnknownLeaseException, RemoteException;

Duration is the new duration requested of the lease, it is not added to the remaining time of the lease
void setSerialFormat(int format);

   Specifies how to store the lease time when serialized. Options are either duration (how long the
lease will last from the time it was serialized) or absolute (in the current clock time). Use duration
when sending a lease to a remote machine. Duration is the default.

int getSerialFormat();

LeaseMap createLeaseMap(long duration);
boolean canBatch(Lease lease);

   Used in handling leases in batches.

---

**Lease Exceptions**

LeaseDeniedException

   An exception generated when a lease request or renewal is denied.

LeaseException

   Generic superclass for specific lease exceptions.

LeaseMapException

   An exception generated when a LeaseMap renewAll or cancelAll call generates exceptions on one
or more leases in the map.

UnknownLeaseException

   An exception used to indicate that a lease is not known to the grantor of the lease