Jini: What it is, how we use it, and where it's going

Philadelphia Area Java Users' Group
December 12, 2001

Michael Ogg
CTO, Valaran Corporation

http://www.valaran.com
ogg@valaran.com
• v1.0 Announced by Sun Microsystems Jan. 1999
• Marketed as “ubiquitous computing”
• Main concepts very simple:
  – All objects (service) defined by java interface
  – Lookup based on object type (not name)
  – Registration means depositing proxy code
  – Decoupling of object model from wire protocol
  – Nothing is forever: registration receives lease
  – Objects register to receive remote events
  – Attributes can be attached to objects
• Java is “natural” – but services need not be java
Consequences of Design

• Jini enables spontaneous, self-healing systems
  – Discovery, reflection, lease expiration
• Intrinsically scalable
  – Point-point invocation, remote events
• Dynamic, flexible, changeable
  – Mobile, replaceable proxies
• Heterogeneous in language, protocol
  – Wire protocol decoupled from use (java interface)
• Resilient, no single point of failure
  – Use multiple lookup services
• Objects can carry their own descriptions, UI, etc.
  – Attributes
Jini in a Nutshell

Lookup Service

Proxy

Client

Proxy

Service

Local Method

Arbitrary Protocol
Jini in a Nutshell

- Proxy
- Lookup Service
- Agent
- Service
- Local Method
- Arbitrary Protocol
• Service finds Lookup Service – LS finds Service
  – Service multicasts LS-discovery datagram to LS
  – LS multicasts LS-announce datagram to Services
  – All LS instances eventually get all Service registrations
  – All Services eventually get all LS proxies

• Datagram contains host:port for unicast discovery
  – Only bootstrapping is knowledge of discovery protocol

• Lease renewal ensures liveness
  – Expiration removes stale references
  – All LS instances eventually consistent
  – Lease time negotiated between LS and Service. Can change dynamically
• Proxy is mobile java code. Implements
  – java.rmi.Remote for “trivial” RMI/JRMP object
  – java.io.Serializable for any other protocol
  – Proxy byte code from ClassLoader or http

• Remote Events: peer-peer events across JVMs
  – Listener registers with source, and receives lease
  – Event notification can be delegated to 3rd party
  – Events can be multicast to several listeners

• Attributes allow objects to carry extra information
  – Implements net.jini.core.entry.Entry
  – Can be descriptive, UI, include html docs, etc.
• Additional properties:
  – Transaction support (lease expires at end of transaction)
  – Javaspaces related service for global object repository
  – Event mailbox, Lease renewal manager in v1.1

• Jini characterized by its light weight
  – Contains enough for robust distributed computing
  – And no more

• For the full story: http://www.sun.com/jini
Architectural Concepts

- Based on **distributed service** architecture
- All components (services) are modeled as a Jini service with a Java interface
- The services are comprised of:
  1. Core (framework) native services
  2. Legacy services with adaptors (proxies)
- The Business Process is the (distributed) application
- The logic controlling that Business Process can be in zero, one, or several Jini services
For everything that is not a Jini Service:

• Create a Jini Service:

1. Model the behavior with a Java interface
   Will it be a Remote or Serializable proxy?

1. Implement the proxy
   Where is the proxy instantiated?
   Where does the proxy execute?

1. Deploy the proxy
For example: a device that is controlled by a string sent to a “well-known” UDP port

```java
public interface MyDevice {
    public void send(String msg)
        throws java.io.IOException;
}
```
public class MyDeviceProxy

    implements MyDevice, java.io.Serializable {

    ...

    public void send(String msg) throws java.io.IOException {
        byte[ ] buff = msg.getBytes();
        DatagramPacket datagram =
            new DatagramPacket(buff, buff.length, host, port);
        getSocket().send(datagram);
    }

}
Fine Grain NGOSS™ Architecture Framework

Valaran Framework Services
- Service Container
- Service Controller
- Resource
- Security
- Persistence
- Transaction
- Log
<table>
<thead>
<tr>
<th>Jini</th>
<th>CORBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup Service</td>
<td>Name Service</td>
</tr>
<tr>
<td>Stored by Class, Attribute</td>
<td>Objects stored by name (string)</td>
</tr>
<tr>
<td>Resolves to Stub or Proxy</td>
<td>Resolves to IOR</td>
</tr>
<tr>
<td>Lookup Service found by multicast</td>
<td>NS found by &quot;well known&quot; IOR</td>
</tr>
<tr>
<td>Object in LS until lease expires</td>
<td>Object in NS until removed</td>
</tr>
<tr>
<td>Any transport</td>
<td>IIOP &quot;natural&quot; transport</td>
</tr>
<tr>
<td>Decentralized event listeners</td>
<td>Centralized Event Service</td>
</tr>
<tr>
<td>Dynamic object architecture</td>
<td>Static object architecture</td>
</tr>
</tbody>
</table>
• CORBA (or other proprietary systems) forces specific protocols
• Jini is protocol-agnostic
  – Entire system may be Jini
  – Or Jini can interconnect protocol islands
• In particular, Jini can interconnect unmodified legacy services
  – Legacy service: client-side proxy
  – Legacy client: server-side proxy
  – Legacy service & client: client & server-side proxies
• Jini does not distinguish hardware and software
Where is Jini Heading?

• Next release, JSK 1.2 (aka Alewife) December 2001
• JSK 1.3 (Davis) planned early 2003
• Davis available for review now
• Jini Community is a thriving organization
• Jini Decision Process, JDP, established
• Follow it at: 
  
  **http://www.jini.org**