NAF: Jini & EJB

Dr. Pekka Nikander
Chief Scientist, Ericsson Research NomadicLab
Adjunct Professor (docent), Helsinki University of Technology
Lecture outline

• Introduction

• Jini
  • Service model
  • Lookup Service
  • Leases
  • Security

• Java Spaces

• Enterprise Java Beans (EJB)
  • Service model
  • Deployment descriptions
  • Security
Introduction

• Two very different Java based distribution systems

• Jini
  • Goal: real zero administration
  • Suitable for ad hoc like environments
  • Aims for protocol independence via code shipping
  • Fairly early in life curve
  • Adoption spotty, mostly visionary companies

• Enterprise Java Beans
  • Middleware oriented, CORBA like
  • Suitable for static server environments
  • Builds directly on the top of Java RMI
  • Reasonably mature
  • Used fairly widely, even main street companies
Jini

- More an architecture than ready-to-deploy technology
- Current version 1.1 (same as year ago) or 1.2Beta
- Very mobile code centric
  - Each client must have a Java Virtual Machine
  - Servers are supposed to a JVM, but there are ways around
    - One can use a surrogate server running a JVM
- The idea: A fixed wireline protocol is replaced with a fixed interface + downloaded code to for the protocol
- Very much flexibility for the service developer
- Requires a common execution model for clients
  - This Java dependency is seen as a large drawback
Jini Service Model

1. Register Service

Server

Service

Lookup Service

Serialized Proxy
Jini Service Model

1. Register Service

Client JVM

Application

Server

Service

Lookup Service

Serialized Proxy

2. Look up Service
Jini Service Model

1. Register service

2. Look up service

3. Download Proxy

Client JVM

Application

Serialized Proxy

Server

Service

Lookup Service

Serialized Proxy
Jini Service Model

1. Register Service
2. Look up Service
3. Download Proxy
4. Instantiate Proxy
Jini Service Model

1. Register Service
2. Look up Service
3. Download Proxy
4. Instantiate Proxy
5. Use the Service
Jini Lookup Service

• The Jini directory
• (Usually) serves the local community only
  • Jini servers and clients search for a local lookup service using a Discovery Protocol
  • Jini servers *join* to a Jini *community* by running a *join protocol*, basically entering a proxy to the lookup service
  • Communities can be *federated* by registering a lookup server as a service within a remote lookup server
• The lookup stores *proxies for the services*
  • When the client wants to use a service, it downloads the proxy
Jini Leases

- Everything in Jini revolves around leases
- Leases are *the* means of housekeeping
- Whenever a resource is reserved for some remote party, a lease is handed over to the remote party
  - Each lease has an expiration time
  - The remote party has to *renew* the lease before it expires
  - If the lease expires, the resource is automatically released
- **E.g. when joining a community, the server gets a lease**
  - The lease is handed over by the lookup service
  - The lookup service stores the proxy until the lease expires
  - If the server crashes or leaves, the service automatically disappears from the lookup service after the expiration
Jini Security

• Sun is working on Jini security; none available yet
• Standard JDK 1.2 security is available, though
• Difficult to introduce one-size-fits-all security
  • Firstly, we still want to retain the protocol independency
  • Secondly, leases have no standard on-the-wire representation
• Pasi Eronen developed a framework as his Master’s
Jini Security

1. Register signed proxy
2. Look up service
3. Download Proxy

Client JVM
- Application
- Proxy
- Security Manager

Lookup Service
- Signed Proxy

Server
- Service
- Secure Dispatcher
Jini Security

1. Lookup Service
2. Signed Proxy
3. Application
3'. Authenticate the Proxy

Client JVM

Proxy

Server

Service

Secure Dispatcher

Security Manager
Jini Security

1. Lookup Service
2. Signed Proxy
3. Proxy
3’. Security Manager
4’ & 4”. Delegation Application > Proxy

Client JVM

Application

Server
Service
Secure Dispatcher
Jini Security

Client JVM

Application
4’ & 4”
Proxy

3’. Security Manager

Lookup Service
Signed Proxy

1.

Server

Secure Dispatcher

2.

3.

4’ & 4”

5’. Authenticate the Proxy
Jini Security

1. Lookup Service
   - Signed Proxy

2. 2.

3. 3.
   - 4' & 4''
   - Proxy

3'. 3'.
   - Security Manager

4. 4'.

5. 5'.
   - 5''. Check Chain

Client JVM
   - Application
     - 4' & 4''

Server
   - Service
   - Secure Dispatcher
Jini Security

1. Lookup Service
2. Signed Proxy
3. 4' & 4"
3'. Proxy
4' & 4''
5'. Security Man
5. Use the Service

Client JVM

Application

Server

Service

Secure Dispatcher
Java Spaces

- “Service farm” based on Jini
- A persistent “tuple space” for Jini
  - Much like Linda implemented in Jini with added persistency
- Clients register tuple “templates” for their interest
- Other clients enter tuples to provide resources
- JavaSpaces matches the tuples with the template
  - The template registered party receives the tuple
- Can be used as the main means of communication, or to bring together interested parties
Enterprise Java Beans (EBJ)

- Java “competitor” for CORBA and COM+/MTS
- Platform for Java based middleware servers
- Contains limited CORBA support (RMI over IIOP)
- Different implementations by different vendors
- Programming guide available from Nova Labs
Service model

The EJB Object interface intercepts all method calls and implements transactions, state management, persistence, and security services for the bean based on deployment descriptor settings.

The EJB Home interface is accessible through JNDI and implements all lifecycle services for the bean.
Deployment descriptions

- The “Beans” are described with XML

```xml
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>EmployeeService</ejb-name>
      <home>wombat.EmployeeServiceHome</home>
      <remote>wombat.EmployeeService</remote>
      <ejb-class>wombat.EmplServiceBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Bean</transaction-type>
      <env-entry>
        <env-entry-name>envvar1</env-entry-name>
        <env-entry-type>String</env-entry-type>
        <env-entry-value>some value</env-entry-value>
      </env-entry>
    </session>
  </enterprise-beans>
</ejb-jar>
```
<ejb-ref>
  <ejb-ref-name>ejb/EmplRecords</ejb-ref-name>
  <ejb-ref-type>Entity</ejb-ref-type>
  <home>wombat.EmployeeRecordHome</home>
  <remote>wombat.EmployeeRecord</remote>
  <ejb-link>EmployeeRecord</ejb-link>
</ejb-ref>

<resource-ref>
  <res-ref-name>jdbc/EmployeeAppDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
</session>
</enterprise-beans>
Security

- No single, all covering security standard
- Basic Java 1.1 and 1.2 security available
- EJB containers provide RBAC
  - The requestor has a Principal and Roles
  - Access control defined in the deployment description
- Underlying RMI can be secured with JSSE, i.e. SSL
  - Remember, this leaves authorization separate from the cryptographic authentication performed at the SSL level
  - Thus, the requestor must be trusted not to send bogus Principal or Role information
- Assumption: JAAS will eventually fix the situation?