Jini™ Technology Overview

Bob Scheifler

Senior Staff Engineer

Sun Microsystems, Inc

Talk outline

- very brief Jini overview
- Jini lookup service in some depth
  - service types and type matching
  - attributes and attribute matching
  - events
  - discovery

Goals of Jini

- network service architecture
  - federate users and resources required by them
- spontaneous networking
  - simplify building, maintaining, altering and using a network of devices, software and users
  - network plug-and-work, unplug-and-heal
- erase the hardware/software distinction
- simplicity

Unabashedly Java-centric

- homogeneous network
  - assume everything has a Java Virtual Machine
- single type system
  - both in the JVM and on the network
- both code and data are portable
  - Java object serialization
  - Java RMI (Remote Method Invocation)
- simplicity and power

Jini structure

- federation infrastructure
  - lookup service
  - discovery protocol
- distributed programming model
  - leasing
  - events
  - transactions
- services

Federation

- services and clients find a lookup service using the discovery protocol
- services advertise by registering with the lookup service
  - focus is workgroup-sized federations
- clients find services by Java type
- peer-to-peer communication after that

Leasing

- used to control resource lifetimes
  - garbage collection
- client requests a resource for some period of time
- service grants access for some period
  - may be less than requested
- client can renew or cancel the lease

Leasing

- resource is freed if not renewed within granted time
- both parties know the outcome up front
  - client crash
  - service crash
  - network failure

Distributed events

- extends AWT/JavaBeans event model
- client registers interest in events
  - interest characterization is service-specific
  - client provides RMI-based listener callback
  - registration with service is leased
- event objects can include behavior
  - code downloading on demand
- supports third-party delivery services

Transactions

- ACID properties for objects
- two-phase commit protocol
  - multi-service distributed transactions
  - supports nested transactions
  - transaction manager service (coordinator)
- participant services control semantics
  - e.g., two-phase locking, causal ordering, orphan detection, deadlock detection

Jini service structure

- service is defined by one or more Java interfaces
- service has a front-end *proxy* object
  - RMI stub or custom-coded class
- proxy implements Java interfaces
- clients only know about Java interfaces

Jini service structure

- proxy class can be specialized
  - to particular implementation of service
  - network protocol between proxy and service can be private and varied
- proxy class is downloaded to clients
  - classes are annotated with codebase URL
  - clients oblivious to implementation details

Example: Transaction Manager

- service interface
  - net.jini.core.transaction.server.TransactionManager
- RMI-based service implementation
  - com.sun.jini.mahalo.TxnManagerImpl
- proxy class (RMI stub)
  - com.sun.jini.mahalo.TxnManagerImpl_Stub
- client only knows about TransactionManager interface

Example: JavaSpace

- service interface
  - net.jini.space.JavaSpace
- RMI-based service implementation
  - com.sun.jini.outrigger.FrontEndSpace
- proxy class (custom-coded)
  - com.sun.jini.outrigger.SpaceProxy
- client only knows about JavaSpace interface

Jini lookup service

- network service
  - for finding other network services
  - “workgroup”-scale, not the entire Internet
- *not* a naming or directory service
  - X.500, LDAP, DNS, NDS, RMI registry
- focus is on types, not names
  - yellow pages, not white pages
- both programmatic and end-user access

Lookup service is closer to:

- IETF Service Location Protocol
- OMG Trading Object Service

Feature Comparison

	Jini	SLP	Trader
discovery	✓	✓	
leases	✓	✓	
events	✓		
single type system	✓		
service type hierarchy	✓	2 levels	✓
service value	object	URL	stub
attribute value	object	string/int/bool	object
query language	template	✓	✓
get service & attributes	✓		✓
lookup by multiple type	✓		
lookup by attribute only	✓		
co-dependent attributes	✓		
multi-valued attributes	✓	✓	
cross-server links			✓
dynamic attribute values			✓

Lookup service 101

- each service registers its proxy object
- client does lookup by Java type
  - *instanceof* semantics
  - API uses java.lang.Class instances, not strings
  - client gets proxy object back
  - needed classes are automatically downloaded

Service type matching

- can match on any Java type
  - including superinterface or superclass
  - can get subinterface or subclass back
- example: NestableTransactionManager extends TransactionManager
- java.lang.Object matches everything

Service type matching

- can match using multiple Java types
  - AND semantics
- “I want a combination TV/VCR”
  - org.ce.TV AND org.ce.VCR
- yellow pages
  - automatic placement in all categories
    - all superinterfaces and superclasses

Attributes

- each service also registers with attributes
- attributes allow further discrimination between services of the same type
  - by end-user or program
- can do lookup using service types and/or attributes

Example generic attributes

- name
- location
- manufacturer
- model

Example printer attributes

- monochrome or color
- resolution
- speed
- paper size

Attribute characteristics

- some defined by service itself
- some defined by administrator
- values aren’t always strings
- multiple values for some attributes
- attributes can have co-dependent values

Example printer attributes

- monochrome, 600 dpi, 10 ppm
- monochrome, 300 dpi, 15 ppm
- color, 300 dpi, 5 ppm

- letter, legal, envelope

Attribute representation

- each attribute is a public field of a class
  - attribute name = field name
  - attribute type = field type
- each class defines an *attribute set*
  - co-dependent attributes in the same class
  - separate classes for orthogonal attributes
- each service has *set of attribute sets*
  - instances of multiple classes
  - multiple instances of same class

Example attribute classes

```
public class PrintAttrs implements net.jini.core.entry.Entry {
    ...
}
public class PrintEngine extends PrintAttrs {
    public Integer ppm;
    public Integer dpi;
    public Boolean color;
    ...
}
```

Example attribute classes

```
public class PaperSize extends PrintAttrs {
    public PaperType size;
    ...
}
public class PaperType implements java.io.Serializable {
    public static final PaperType Letter = ...;
    public static final PaperType Legal = ...;
    public static final PaperType A4 = ...;
    public static final PaperType Envelope = ...;
    ...
}
```

Example attribute classes

```
public class Location implements net.jini.core.entry.Entry {
    public String floor;
    public String room;
    public String building;
    ...
}
public class Name implements net.jini.core.entry.Entry {
    public String name;
    ...
}
```

Example printer attributes

PrintEngine{color: false, dpi: 600, ppm: 10}

PrintEngine{color: false, dpi: 300, ppm: 15}

PrintEngine{color: true, dpi: 300, ppm: 5}

PaperSize{size: PaperType.Letter}

PaperSize{size: PaperType.Legal}

PaperSize{size: PaperType.Envelope}

Location{floor: “1”, room: “1220”, building: “UCHL03”}

Name{name: “narf”}

Attribute set matching

- match on attribute set class
  - including superclass
- match on attribute values
  - exact value match only
- template is just an attribute set instance
  - null field is wildcard = match any
- taken from Linda via JavaSpaces

Matching examples

template	PrintEngine{color: true, dpi: 300, ppm: 5}
matches	itself
template	PrintEngine{color: false, dpi: null, ppm: null}
matches	PrintEngine{color: false, dpi: 600, ppm: 10}
and	PrintEngine{color: false, dpi: 300, ppm: 15}
template	PrintAttrs{}
matches	PrintEngine{color: true, dpi: 300, ppm: 5}
and	PaperSize{size: Legal}

Service template matching

- service template contains:
  - zero or more service types
  - zero or more attribute set templates
- <proxy, set of attr sets> matches if:
  - proxy is *instanceof* each service type, and
  - for each attribute set template:
    - there is a matching attribute set

Example queries

- any service named “narf”
- any Printer named “narf” in UCHL03
- all services on floor 2
- any JavaSpace made by Sun

Complex queries

- “between 300 and 600 dpi”
- “on floor 1, 3, or 7”
- first do template-based match
  - using wildcards as necessary
- then do finer-grained match locally

Attribute modification

- can only be done by service itself
  - allows service to control policy
  - service is responsible for persistent storage
- add, modify, delete, replace all
- attributes should change infrequently
  - primary search keys
  - query service directly for very dynamic state

Basic summary

- service perspective
  - register with desired attributes
  - keep renewing lease
  - add, modify, delete attributes as desired
- client perspective
  - lookup any 1 of, get just proxy object
  - lookup up to N of, get proxies and attributes

Leasing

- ensures results of client lookup are reasonably up to date
- lookup service imposes maximum lease duration
- tradeoff between bandwidth (and load on lookup service) and how up to date
- maximum lease will vary with context

Asynchronous events

- uses same service template matching
  - interest registration is also leased
- no match → match
  - new service added or attributes now match
- match → no match
  - service deleted or attributes no longer match
- match → match
  - attributes changed but still match

Example uses of events

- admin notified of new services
- admin notified when service crashes
- user notified when service is back on line
- admin notified when printer jams

User perspective

- support for incremental browsing
  - by service type
  - by attribute set class
  - by attribute value
- access to service-specific GUIs
  - applet URL as an attribute

Discovery and selection

- how does a client or service find lookup services
- how does a service know which ones to register with
- how does a client know which ones to do lookups in

Lookup groups

- each lookup service is configured to be a member of one or more groups
  - group is just an arbitrary string name
  - one standard “public” group, as default
- each service is configured to register with one or more groups
- each client is configured to use lookup services from one or more groups

Initial discovery

- performed by both clients and services
- send multicast request packet
  - groups you are looking for
  - host and TCP port to use to contact you
- wait for lookup services to connect back
  - only those in ≥ 1 requested groups respond
- download lookup service's proxy object
  - register with or do lookup in

Latecomer discovery

- don’t send multicast requests forever
  - stop after a short while
- listen for multicast announcements
  - sent periodically by each lookup service
  - groups lookup service is a member of
  - host and TCP port to contact lookup service
- connect to if member of desired group
  - download lookup service proxy

Administrative perspective

- service dies
  - lease expires, registration is dropped
- network partition
  - services automatically reregister at merge
- availability: replicate lookup service
  - services automatically register with replicas
- failover: start a new lookup service
  - services automatically register with it

Hierarchy: one option

- DNS-style hierarchical group names
- each lookup service is also a Jini service
- one set of groups to be a member of
- another set of groups to register with
  - up and/or down links in hierarchy

Non-Java services

- integrate using *device architecture*
- build a bridge or proxy service
  - bridge has JVM
  - bridge uses some protocol to find/discover non-Java service
  - bridge discovers lookup service and registers on behalf of non-Java service

Example: SLP bridge

- use SLP to discover DA
- find supported service in DA
- convert service URL to Java object
- convert SLP attributes
- register with Jini lookup service

For more information

- http://java.sun.com/products/jini

