Programming models 2

Brian Nielsen
bnielsen@cs.aau.dk
The registry provides a bootstrap naming service using URLs.

- **rmi://slowww.server.edu/object1**

### Problem:
Client must know what name the server machine has.

---

**Diagram:**
- **Registry = NameServer**
- **RMI registry**
  - obj1
  - obj2
  - obj3
- **client VM**
  - client app
  - proxy
  - RMI layer
  - transport
- **server VM**
  - server app
  - skeleton
  - RMI layer
  - transport

1: `Naming.bind(URL, obj1)`
2: `proxy1 = Naming.lookup(URL)`
3: `proxy2 = proxy1.method()`
Motivation

• Example scenarios
  – "Find all nearby color duplex printers"
  – "Start brewing coffee five minutes before my alarm clock goes off"
  – "Let my cell phone use the car speakers"

• Coordination framework
  – Simple, seamless, and scalable interoperability
  – Network "plug and play" with minimum admin
  – Networked software and hardware provide services
  – Any device can find and use existing services
One Way Scenarios Might Be Done Today

- Desktop PC
- Coffee Maker
- Cell Phone
- Printer Service
- Alarm Clock Service
- Stereo Speaker Service
What Jini Proposes

- Desktop PC
- Coffee Maker
- Cell Phone
- Printer Service
- Alarm Clock Service
- Stereo Speaker Service

Network Lookup service
Spontaneous networking

• Jini enables clients to automatically discover services at runtime

• Associative search
  – Not by name lookup (e.g. http://some.url:port)
  – Instead: find a service that does this or that

• Loose coupling
  – Services and clients can join and leave the system (Jini federation) at any time without causing system failure
# Jini and Java

| Jini Services            | • JavaSpaces™
|                         | • Transaction Managers
|                         | • Printing, Storage, Databases...
| Jini Infrastructure      | • Discovery
|                         | • Lookup Service
| Jini Programming Model   | • Leasing
|                         | • Distributed Events
|                         | • Transactions
| Java 2 Platform          | • Java RMI
|                         | • Java VM
Service Interface

- Everything is a service on the network.
- Interfaces define **what** a service offers, not **how** it implements it.
- Java interfaces give strong typing.
- The *ServiceItem* Java object represents a service registration:
  - A unique service ID
  - The service proxy object
  - A set of optional Java attribute objects
    - (e.g. location, status, vendor, etc)
- Services are found in the lookup service by *template matching* described by a *ServiceTemplate* object.
Discovery, join, lookup protocols

• **Join protocol**
  – Services register with lookup services
  – Registration is a Java object

• **Discovery protocol**
  – Clients and services **find lookup services** by multicasting requests at predefined IP multicast address or direct addressing (*PULL*)
  – Lookup service may advertise its existence by periodic multicasting its presence (*PUSH*)

• **Lookup protocol**
  – Clients send a **search request** to lookup services
  – Specify service type and properties as a Java object
  – Lookup service finds and returns matching services
  – Returned is a Java object (ServiceItem) that carries the service proxy
Service Registration

1. Find Lookup Service
2. send proxy to client ser
3. Upload service proxy

Network

Lookup service

Alarm Clock Service

Lookup service’s Proxy Object

Alarm Clock service’s Proxy Object
Service Leasing

1. Find Lookup Service
2. Return proxy to lookup
3. Query for service
4. Service returned

Find interface AlarmClock(radio=yes, vendor=*)
Using Services

Can use any protocol to communicate to service (or proxy can contain service itself!)
The role of the proxy

- The proxy is a Java object downloaded from the service
- Hides communication between proxy and service
  - Costumizable, dedicated protocol (transparent)
    - Protocol independent (TCP/IP, HTTP, SOAP, etc.)
    - Replication
  - Updatable
  - protocol
  - Interact directly with physical devices
Leasing

• What happens if a service dies?
  – It must be de-registered at the lookup service
  – May crash, or “forget” to do so

• A lease is a time period during which the grantor of the lease promises that the holder of the lease will have access to some resource
  – Eg. The lookup service stores a service registration

• A client requests a lease from the provider (lease grantor) and must periodically renew it.

• Expired or non-renewed lease results in removing lease and freeing up resources
Microsoft Message Queue (MSMQ)

Messaging-Oriented Middleware, .NET/COM style
MSMQ:
Microsoft Message Queueing

• Specification for messaging on Microsoft OS
  – currently at 4.0 (Vista, 3.0 XP)

• Captures essence of messaging
  – Synchronous or asynchronous operation
  – Loosely- or strongly-coupled operation
  – Transacted send/receive
  – Prioritized delivery
  – Deadline-based delivery
  – Delivery filtering

• Several language bindings, including C#
  – System.Messaging namespace
  – System.Messaging.dll assembly
MSMQ: MS Message Queues

Sender 1
Message

Sender 2
Message

Sender 3
Message

<machinename>\private$<queue_name>

Message Queue
Message 1
Message 2
Message 3
Message 4
Message 5

Receiver 1
Message 1

Receiver 2
Message 2

Send()

peek(): returns copy
receive(): removes msg

Header
Body
Label
priority
lifetime
deadletterQ
recoverable
Byte array
with formatted
contents
User defined
string indicating
intention of message
“Order”
The MessageQueue object is just wrapper around native queue
- Queue must already exist on machine.

```csharp
using System.Messaging;
string queueName = @".\private$\myqueue";
if (MessageQueue.Exists(queueName))
{
    MessageQueue myQ = new MessageQueue(queueName);

    Message msg = new Message("Hello World!");
    myQ.Formatter = new XmlMessageFormatter(new Type[]{typeof(string)});
    myQ.Send(msg, "greetingmsg");

    Message msg2 = myQ.Receive();
    Console.WriteLine((string)(msg.Body));

    myQ.close()
}
```
Queues

• Creating the Queue administratively:
  – Message Queuing, under Services and Applications
  – right-click folder to create New queue

• Creating the Queue programatically:
  – `MessageQueue.Create(name)`
Queues

- Domain Mode (public queues via Active Directory)
- Work Group mode (private Queues)

- Independent Client vs. Dependent client (Active Dir)

- System Queues
  - Journal queues
    - stores copies of messages sent to/through/from this machine
    - allows for better quality-of-service guarantees for messages
    - read-only (can't be directly sent to); much like database logs
  - Dead-letter queues
    - final resting place of undeliverable messages
    - one each for transactional and non-transactional Q’s
    - read-only (can only be read and deleted, not sent to)
  - Connector queues
    - used for store-and-forward messaging in route
Event Based Systems
Event driven programming

- Programming principle based on the idea that objects should react on events (state change, either concretely or abstractly) happening in other objects.
- Objects *publishes* information about events that other objects might be interested in knowing about.
- Objects can *subscribe* to receive information about when an event occur.
Event Programming

Keypad ctrl

Event: OpenPressed

Tray ctrl

Display Ctrl

Motor Ctrl

Open Tray

Clear Display

Stop Motor
Architecture for distributed event notification

1. OOI directly notifies subscriber
2. OOI notifies subscriber via observer
3. Observer queries (polls) OOI and notifies subscriber
Remote Events

- Remote events more expensive to send than local events
- Remote events propagate slowly
- Remote events may not arrive in order sent
- Remote events may not arrive at all (or at all subscribers)
- What does sender do if receiver has crashed -- keep trying? How long?
Jini Remote Events
Jini Remote Events

1: register(…)

2: eventRegistration

3: notify(remoteEvent)

4: renew(…)

Remote event listener
Jini Events

• **Event generators:** An object, that allows other objects to subscribe to its events and generates notification (via RMI).

• **Remote event listeners:** An object that can receive notifications.

• **Remote events:** An object that is passed by value to remote event listeners (a notification).

• **Third-party agents:** Objects that may be interposed between an object of interest and a subscriber (observers).

• Subscriptions are subject to leasing
RemoteEventListener interface

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

RemoteEvent Class

public class RemoteEvent extends java.util.EventObject {
    public RemoteEvent(
        Object source,
        long event ID,
        long SeqNum,
        MarshalledObject handback) {
        
    }

    public Object getSource( ) {
        public long getID( ) {
        public long getSequenceJumber( ) {
        public MarshalledObject getRegitrationObject {
    }

    source+eventID+SeqNum uniquely identifies event occurrence:
    ⇒ Notify is IDEMPOTENT
**Example EventGenerator**

```java
public interface EventGenerator extends Remote {
    public EventRegistration register(
        long evID,
        MarshalledObject handback,
        RemoteEventListener toInform,
        long leaseLength)
        throws UnknownEventException, RemoteException;
}
```
The EventRegistration Class

public class EventRegistration implements java.io.Serializable
{
    public EventRegistration( long eventID,
                            Object eventSource,
                            Lease eventLease,
                            long seqNum) {

    }
    public long getID( ) {

    }
    public Object getSource( ) {

    }
    public Lease getLease( ) {

    }
    public long getSequenceNumber( ) {

    }
}
3rd party objects (Observers)

- Store-and-forward agents
- Notification filters
- Notification mailboxes
Store and forward agent

- Offloading the (remote) communication of the event generator.
- Delivery policies (e.g., retry policies)
- Reliable Multicast, hardware multicast
Notification Multiplexing

Event generator

Machine A

Notification filter

Obj X

Obj Y

Obj Z

Machine B

Notification filter

Obj X

Obj Y

Obj Z

Optimize network usage
Notification Demultiplexing

Generate Composite Events
- N consecutive events
- N different events
- An event at each source
Notification mailbox

1) Mailbox stores events until registrant comes “online”.
2) When registrant comes “online” the stored events are delivered.
Distributed Shared Memory

• Give processes the *illusion* that they are running on a shared memory machine, even though they run on different machines and that their memories are physically distributed.
Virtual Memory 1

Load $aaff (load valid address)

Address space p1

Address space p2

Address space p3

MMU uses pagetable to map to physical address

Load $0011

Virtual Memory (pages)

Physical Memory (frames)

Disk
Virtual Memory 2

Address space p1
Load $ddff (unmapped or invalid page)

Address space p2

Address space p3

MMU uses pagetable to map to physical address
Gives page fault interrupt, served by OS

Virtual Memory (pages)

Physical Memory (frames)

Disk
Distributed Shared Memory 2

Load $aaff (load address not present)

Address space p1
Address space p2
Address space p3

Virtual Memory (pages)

MMU uses pagetable to map to physical address
Gives page fault interrupt, served by OS

Physical Memory (frames)

Host 1
Host 2
Host 3

Memory Consistency when writes occur???!!

=> Coherency Protocols
Linda (and JavaSpaces)

David Gelernter, yale University.
Linda

- Program Concurrency by using uncoupled processes with shared data space
- Add concurrency into a sequential language by adding:
  - Simple operators
  - Runtime kernel (language-independent)
  - Preprocessor (or compiler)
Basic Idea

• Have a shared memory space (“tuple space”)
  – Processes can add, read, and take away values from this space
• Bag of processes, each looks for work it can do by matching values in the tuple space
• Implicit load balancing, synchronization, messaging, etc.
<table>
<thead>
<tr>
<th></th>
<th>Conventional Memory</th>
<th>Linda/JavaSpaces</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit</strong></td>
<td>Bit</td>
<td>Logical Tuple</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(23, “test”, false)</td>
</tr>
<tr>
<td><strong>Access Using</strong></td>
<td>Address (variable)</td>
<td>Selection of values</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
<td>read, write</td>
<td>in, out, read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JavaSpaces: read, write, take</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Immutable</td>
</tr>
</tbody>
</table>
Tuple Space Operations

- **out** \((t)\) – add tuple \(t\) to tuple space
- **in** \((s) \rightarrow t\) – returns and removes tuple \(t\) matching template \(s\)
- **read** \((s) \rightarrow t\) – same as **in**, except doesn’t remove \(t\).

- Originally also **Eval**\((f(42),g(45))\);
- Operations are atomic (even if space is distributed)
Meaning of in

in ("f", int n)
in ("f", 23)
in (?string s, ? int n)
in ("cookie")

Tuple Space

("f", 23)
("t", 25)
("t", true)
("t", false)
("f", 17)
Counting Semaphore

- Create (int n, String resource)
  
  for (i = 0; i < n; i++) out (resource);

- Down (String resource)
  
  in (resource)

- Up (String resource)
  
  out (resource)
Distributed Ebay

• OfferItem (String item, int min bid, int timeout):
  out (item, minbid, “owner”);
  sleep (timeout);
  in (item, ? bid, ? bidder);
  if (bidder ≠ “owner”) SOLD!

• Bid (String bidder, String item, int bid):
  in (item, ? highbid, ? highbidder);
  if (bid > highbid) out (item, bid, bidder)
  else out (item, highbid, highbidder)
Further Reading

• Jini Specification
  http://www.sun.com/jini/specs

• Jini Developers
  http://jini.org

• Universal Plug and Play
  http://www.upnp.org
END