Lecture 06: Distributed Object

Distributed Systems
Behzad Bordbar
School of Computer Science, University of Birmingham, UK

Recap
Interprocess communication
- Synchronous and Asynchronous communication
- use of Socket for comm.
- various types of failure
- “no global time”
- Synchronous and Asynchronous interaction model
- Java API for UDP

Overview
Distributed applications programming
- distributed objects model
- RMI, invocation semantics
- RPC
- events and notifications
Products
- Java RMI, CORBA, DCOM
- Sun RPC

Objects
Objects = Data (attributes) + Operations (methods)
- encapsulating Data and Methods
- State of Objects: value of its attributes
Interact via interfaces:
- define types of arguments and exceptions of methods

The object (local) model
Programs:
- a collection of objects
Interfaces
- the only means to access data, make them remote?
Actions
- via method invocation
- Interaction, chains of invocations
- may lead to exceptions, specified in interfaces
Garbage collection
- reduced effort, error-free (Java, not C++)

In contrast: distributed object model
Objects distributed (client-server models)
Extend with
- Remote object reference
- Remote interfaces
- Remote Method Invocation (RMI)
Remote object reference

Object references
- Used to access objects which live in processes
- Can be passed as arguments, stored in variables...
Remote object references
- Object identifiers in a distributed system
- Must be unique in space and time
- Error returned if accessing a deleted object
- Can allow relocation (as in CORBA)

Constructing unique remote object reference
- IP address, port, interface name
- Time of creation, local object number (new for each object)
Use the same as for local object references
If used as addresses
- Cannot support relocation (alternative in CORBA)

Remote interfaces

Specify externally accessed
- Variables and procedures
- No direct references to variables (no global memory)
- Local interface separate

Parameters
- Input, output or both,
- Instead of call by value, call by reference
No pointers
No constructors

Remote object and its interfaces

CORBA: Interface Definition Language (IDL)
Java RMI: as other interfaces, keyword Remote

Handling remote objects

Exceptions
- Raised in remote invocation
- Clients need to handle exceptions
- Timeouts in case server crashed or too busy

Garbage collection
- Distributed garbage collection may be necessary
- Combined local and distributed collector
- Cf Java reference counting

RMI issues

Local invocations
- Executed exactly once
Remote invocations
- Via Request-Reply (see DoOperation)
- May suffer from communication failures!
  - Retransmission of request/reply
  - Message duplication, duplication filtering
- No unique semantics…
Invocation semantics summary

<table>
<thead>
<tr>
<th>Fault tolerance measures</th>
<th>Invocation semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retransmit request message</td>
<td>Duplicate filtering</td>
</tr>
<tr>
<td>No</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Re-executing a method sometimes dangerous...

Communication modules

Reside in client and server
Carry out Request-Reply jointly
- Use unique message ids (new integer for each message)
- Implement given RMI semantics

Server’s communication module
- Selects dispatcher within RMI software
- Converts remote object reference to local

Remote reference module

Creates remote object references and proxies
Translates remote to local references (object table):
- Correspondence between remote and local object references (proxies)
Directs requests to proxy (if exists)
Called by RMI software
- When marshalling/unmarshalling

RMI software architecture

Proxy (for transparency)
- Behaves like local object to client
- Forwards requests to remote object

Dispatcher
- Receives request
- Selects method (methodID) and passes on request to skeleton

Skeleton
- Implements methods in remote interface
  - Unmarshals data, invokes remote object
  - Waits for result, marshals it and returns reply

Binding and activation

The binder
- Mapping from textual names to remote object references
- Used by clients as a look-up service (cf Java RMI registry)

Activation
- Objects active (within running process) and passive (implementation of methods + marshalled state)
- Activation = create new instance of class + initialise from stored state

Activator
- Records location of passive and active objects
- Starts server processes and activates objects within them
Object location issues

- Persistent object stores
  - Stored on disk, state in marshalled form
  - Readily available
  - Cf Persistent Java
- Object migration
  - Need to use remote object reference and address
- Location service
  - Assists in locating objects
  - Maps remote object references to probable locations

Remote Procedure Call (RPC)

- RPC
  - Historically first, now little used
  - Over Request-Reply protocol
  - Usually at-least-once or at-most-once semantics
  - Can be seen as a restricted form of RMI
  - Cf Sun RPC
- RPC software architecture
  - Similar to RMI (communication, dispatcher and stub in place of proxy/skeleton)

RPC client and server

Implemented over Request-Reply protocol.

Summary

- Distributed object model
  - Capabilities for handling remote objects (remote references, etc)
  - RMI: maybe, at-least-once, at-most-once semantics
  - RMI implementation, software architecture
- Other distributed programming paradigms
  - RPC, restricted form of RMI, less often used
- Further reading: chapter 5

Exercises:

5.2 Discuss the invocation semantics that can be achieved when the request-reply protocol is implemented over a TCP/IP connection, which guarantees that data is delivered in the order sent, without loss or duplication. Take into account all of the conditions causing a connection to be broken.

5.4 The Election service must ensure that a vote is recorded whenever any user thinks they have cast a vote. Discuss the effect of maybe call semantics on the Election service. Would at-least-once call semantics be acceptable for the Election service or would you recommend at-most-once call semantics?

Exercises:

5.5 A request-reply protocol is implemented over a communication service with omission failures to provide at-least-once RMI invocation semantics. In the first case the implementor assumes an asynchronous distributed system. In the second case the implementor assumes that the maximum time for the communication and the execution of a remote method is T. In what way does the latter assumption simplify the implementation?

5.6 Outline an implementation for the Election service that ensures that its records remain consistent when it is accessed concurrently by multiple clients.

5.7 The Election service must ensure that all votes are safely stored even when the server process crashes. Explain how this can be achieved with reference to the implementation 1 outline in your answer to Exercise 5.6.