CprE 450/550x
Distributed Systems and Middleware

Inter-process Communication

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu
February 11, 2004

Potential Topics of Term Papers

The topic should be related to distributed systems in general, such as

- Group Communications,
- Peer-to-Peer systems,
- Overlay Networks,
- Grid Computing,
- Object Middleware,
- and others.

3

Potential Topics of Term Papers

- Resource Discovery and Management
- Security and Policy Management
- Resource Scheduling and Load Balancing
- Synchronization (e.g., clock synchronization, Election Algorithm, mutual exclusion, etc.)
- Consistency and replication
- Reliability and Survivability
- Performance Evaluation
- Anonymity. Censor-resistent
- Workload characterization
- Multi-cast Fingerprinting
- Anonymous authentication in dynamic group communications
- Data Replication strategies for Grid Computing systems
- Reputation-based resource scheduling for Grid Computing systems
- Middleware-based application design and development (e.g., real-time CORBA, fault-tolerant CORBA, etc.)

- **You are welcome to propose your own topic!!!**

4

Your Term Papers

- Every student is required to finish a term paper.

- Deadlines:
  - **Feburary 26 (Thursday, 5:00pm), topic selection due (UG).**
  - **March 11 (Thursday, 5:00pm), 3-pages proposal (problem definition) due (UG).**
  - **March 25 (Thursday, 5.00pm), 5-pages solutions and drafted evaluation plan due (G).**
  - **April 6 (Tuesday, 5:00pm), 5-pages summary of the papers you read and important issues you think (U)**
  - **April. 15 (Thursday 5:00pm), experimental results and improved solution due (G).**
  - **April. 27 (Tuesday 5:00m), 10-page literature survey (U) and 15-pages full term paper (G) due, including problem definition, solutions, experimental data, conclusion and future works.**

- Paper format: Latex or WORD, IEEE transactions, please refer:
  http://www.ieee.org/organizations/pubs/transactions/stylesheets.htm

5

Readings for Today's Lecture

- References
  - **Chapter 2 of "Distributed Systems: Principles and Paradigms"**

6

Object-Oriented Distributed Technology

- Objects
- Objects in Distributed Systems
- Requirements of Multi-User Applications

7

Object-Oriented Languages

- Object Identity
  - **"object identifiers" (OIDs)**
  - **OIDs as first class values**
- Actions
  - **Inititiated by sending message to object requesting method invocation**
  - **State in object may change**
  - **cascaded invocations of methods**
- Dynamic Binding
  - **The method executed is chosen according to the class of the recipient of the message.**
- Garbage Collection
  - **Dynamically allocated instances may be explicitely *deleted* or space is freed implicitely by garbage collector.**

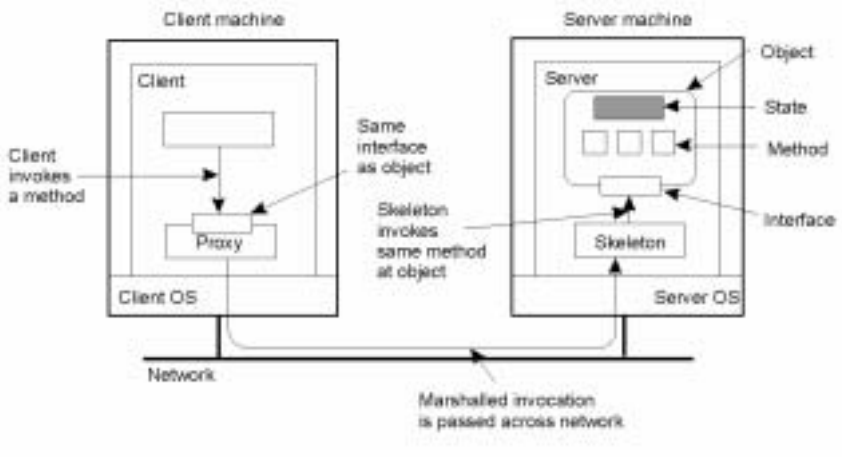
8

Objects in Distributed Systems

- Object Identity in a Distributed System
  - **Remote object identifiers (ROIDs)**
  - **Ex. Java: ROID = endpoint (Java vm) + identifier (ObjID)**
  - **ROIDs as first-class values**
  - **Service for comparing remote object identifiers**
    - **e.g. Java: *RemoteObject::equals()***
- Actions in a Distributed Object System
  - **Remote Method Invocation**
- The Role of Proxies for Transparent RMI
  - **Local proxy for each remote object that can be invoked by local object.**
  - **Local proxy behaves like local object, but, instead of executing message, forwards it to the remote object. (client stubs)**
  - **Remote object has skeleton object with server stub procedures**

9

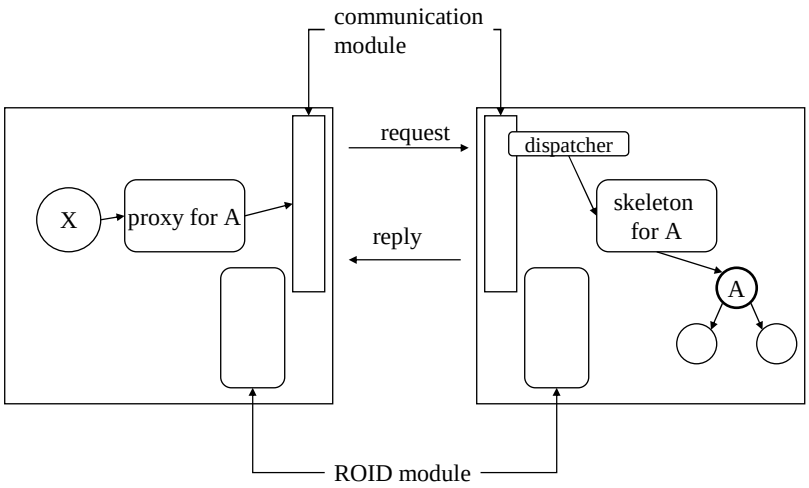
Distributed Objects



- Common organization of a remote object with client-side proxy.

10

Proxies and Skeletons



11

Proxies and Skeletons (cont)

- Proxies:
  - **Need proxies to invoke remote objects.**
  - **Proxies are created when needed whenever ROID arrives in Reply message.**
  - **ROID module manages proxies and ROIDs.**
- Dispatchers and Skeletons:
  - **Not necessary for systems with reflection capabilities.**
  - **e.g. class *Method* in Java 1.2 reflection package: method *invoke* can be called on instance of *Method*. Dispatcher now generic and skeleton unnecessary.**

12

Binding a Client to an Object

```
Distr_object* obj_ref;                  //Declare a systemwide object reference
obj_ref = ...;                          // Initialize the reference to a distributed object
obj_ref-> do_something();               // Implicitly bind and invoke a method
```

(a)

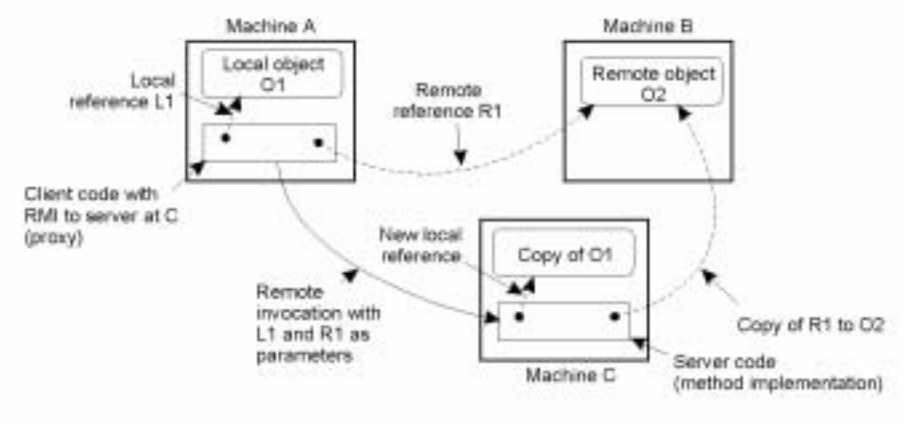
```
Distr_object objPref;                   //Declare a systemwide object reference
Local_object* obj_ptr;                  //Declare a pointer to local objects
obj_ref = ...;                          //Initialize the reference to a distributed object
obj_ptr = bind(obj_ref);                //Explicitly bind and obtain a pointer to the local proxy
obj_ptr -> do_something();              //Invoke a method on the local proxy
```

(b)

(a) Example with implicit binding using only global references
(b) Example with explicit binding using global and local references

13

Parameter Passing



- The situation when passing an object by reference or by value.

14

Arguments and Results in RMI

- Semantics of passing arguments for RMI in object-oriented languages needs to be defined.
- Argument and Result passing in Java RMI:
  - **When type of parameter is defined as remote interface, argument or result is passed as ROID (*by reference*).**
  - **Other non-remote objects may be passed *by value* if they are *serializable*.**
- Which objects can be accessed by RMI?
  - **Any object can be accessed by RMI**
  - **Distinguish between remote objects and local objects. (e.g. Java)**
  - **Use interface definition language (IDL)**
- Problem: migration/replication

15

Dynamic Binding

- Dynamic method binding should also apply to RMI.
- Smalltalk: Allow any message to be sent to any object, and raise exception if method is not supported.

  Distributed Smalltalk: general-purpose proxies.
- Java RMI:

  dynamic binding as a natural extension of local case

  Example:

```
Shape aShape = (Shape) stack.pop();
float f = aShape.perimeter();
```

16

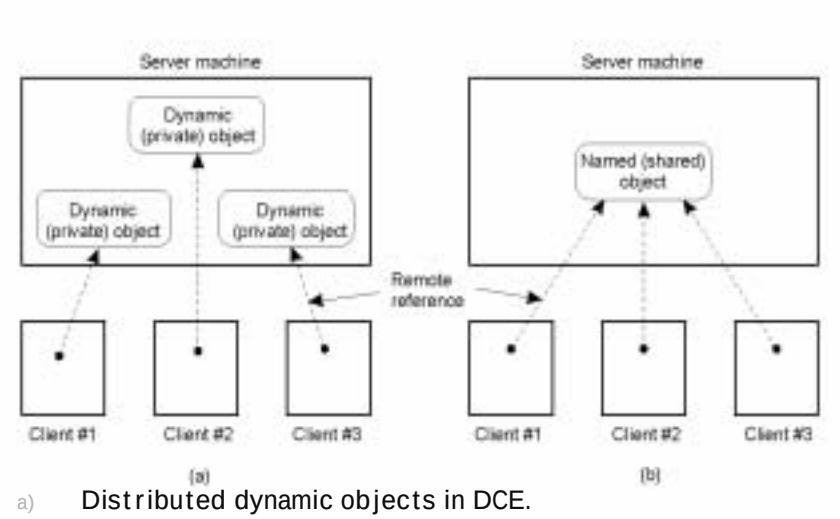
Garbage Collection

- Some languages (Java, Smalltalk) support garbage collection.
- Explicit memory management difficult/impossible in distributed environment.
- Distributed garbage collection typically realized in ROID modules. Each ROID module:

  keeps track how many sites hold remote ROIDs for each local object
  (maintains `holders` table)

  informs other ROID modules about generation/deletion of ROIDs for their local objects (through the use of `addRef()` and `removeRef()`)
- Local garbage collector collects objects with no local or remote references.
- Reference counting (`addROID()`/`removeROID()`) over unreliable networks

The DCE Distributed-Object Model



a) Distributed dynamic objects in DCE.
b) Distributed named objects

Readings for Today's Lecture

- References
  - **Chapter 2 of "Distributed Systems: Principles and Paradigms"**
  - **Chapter 11 of "Java Network Programming and Distributed Systems"**

19

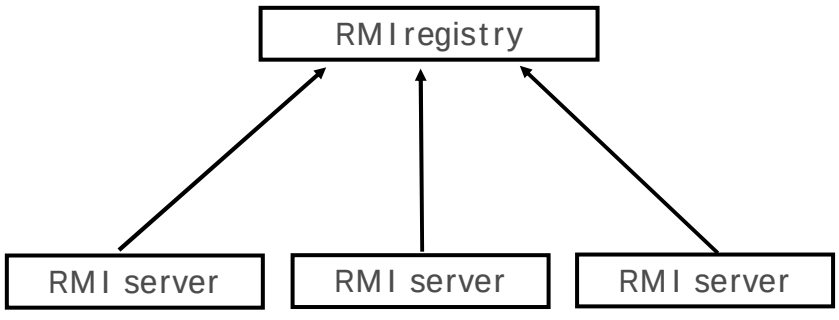
Jave RMI

- RMI: A Java technology that allows one JVM to communicate with another JVM and have it execute an object method.
- RPC and RMI
  - **RPC supports multiple languages, whereas RMI only support Java**
  - **RMI deals with objects, but RPC does not support the notion of objects**
  - **RPC offers procedures (not associated with a particular object)**

20

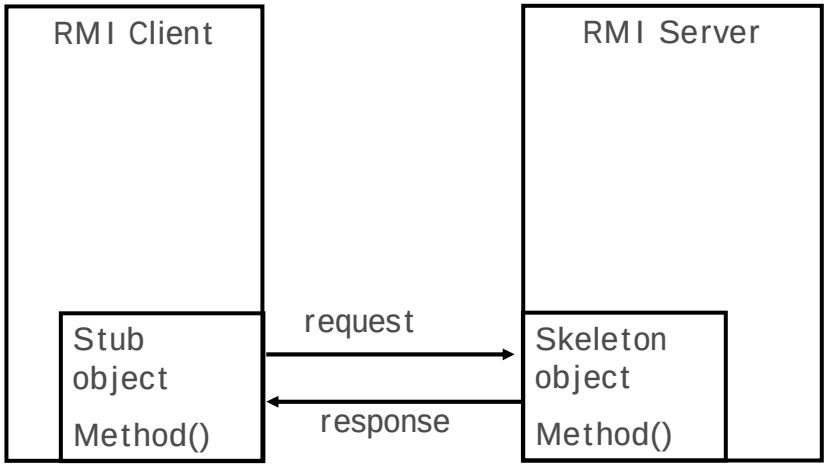
How RMI works

- The format used by RMI for representing a remote object reference: rmi://hostname:port/servicename



21

How RMI works



22

Define a RMI Service Interface

```
Public interface RMILightBulb extends java.rmi.Remote
{
   Public void on() throws java.rmi.remoteexecution;
   Public void off() throws java.rmi.remoteexecution;
   Public boolean ison() throws java.rmi.remoteexecution;
}
```

23

Implement a RMI Service Interface

```
Public class RMILightBulbImpl
    extends java.rmi.server.UnicastRemoteObject
    implements RMILightBulb
{
    Public RMILightBulbImpl () throws java.rmi.remoteexecution
    {setBulb(false);}

    Private boolean lighton;

    Public void on() throws java.rmi.remoteexecution
    {       setBulb(true); }

    Public void off() throws java.rmi.remoteexecution
    {       setBulb(false); }

    Public boolean ison() throws java.rmi.remoteexecution
    {return getBulb();}

    Public void setBulb(boolean value)
    {lighton = value;}

    Public void getBulb()
    {return lighton;}

}
```

24

Create Stub and Skeleton Classes

Rmic RMILightBulbImpl

Two files would be produced:

- **RMILightBulbImpl_Stub.class**
- **RMILightBulbImpl_Skeleton.class**

Create a RMI Server

```
import java.rmi.*;
Import java.rmi.server.*;

Public class LightBulbServer
{
    Public static void main(String args[])
    {
        Try{
                RMILightBulbImpl bulbService=new RMILightBulbImpl();
                RemoteRef location = bulbService.getRef();

                String registry = args[0];

                String registration = “rmi://”+registry+”/RMILightBulb”;

                Naming.rebind(registration, bulbService);
        }
    }
}
```


Create a RMI Client

```
import java.rmi.*;

Public class LightBulbClient
{
    Public static void main(String args[])
    {
        Try{
                String registry = args[0];

                String registration = “rmi://”+registry+”/RMILightBulb”;

                Remote remoteService = Naming.lookup(registration);

                bulbService.on();
                system.out.println(bulbService.isOn());

                bulbService.off();
                system.out.println(bulbService.isOn());


        }
    }
}
```

27

Running the RMI system

- Copy all necessary files to a directory on the local file system of all clients and the server.
- Change to the directory where the files are located, and run rmiregistry.
- In a separate console window, run the server with a hostname where rmiregistry is running.

  Java LightBulbServer hostname
- In a separate console window (another machine), run the client with a hostname where rmiregistry is running.

  Java LightBulbServer hostname

28

Any Questions?

See you next time.