CS 5523 Operating Systems: Remote Objects and RMI

Instructor: Dr. Tongping Liu

Thank Dr. Dakai Zhu and Dr. Palden Lama for providing their slides.

Outline
- Distributed/Remote Objects
- Remote object reference (ROR)
- Remote Method Invocation (RMI)
- Case study and example: Java RMI
- Other issues for remote objects
  - Factory method; Transient vs. Permanent objects;
  - Callback objects; Distributed Garbage collection;

Object-Oriented Programming
- Fundamental idea: encapsulation
  - both data and methods
- Interfaces – define the signature of a set of methods
- Object reference – access object’s data/method
- Method invocation – may affect target objects; may need chain invocation
- Data/variable – could be directly access
- Exceptions – thrown when an error occurs

Distributed Object

Client process:
- disObj1.m1(...);
- disObj1.m2(...);
- disObj2.m3(...);
- disObj2.m4(...);

Machine a

Machine b

Machine c

Obj: m3; m4;

Distributed Object

Distributed Object
Remote Objects vs. Message Passing

Host A

Client:
Sum = math.add(n1, n2);

Host B

Server:
int state;
object math;
int add(int n1, int n2)
{
… …
}

Remote Object Model

- Remote objects -- can receive remote invocations; having state information.
- Remote object reference – identify remote objects in distributed environments
- Remote interface – specifies methods to be invoked remotely
- Process contains objects (local/remote)
  ➢ Local objects: accept only local invocations
  ➢ Remote object: accept both local/remote invocations
  ➢ Remote invocation – different processes (same or different hosts)
- Exceptions – application level

An Example: Remote Object

- Object may implement both remote & local interface
  ➢ Other processes: invoke only methods in remote interface
  ➢ Same process: local object reference and invoke methods in local interface

Objects and Method Invocations

- Objects receiving remote invocations (service objects) are remote objects, e.g., B and F
- Object references are required for invocation, e.g., C must have E’s reference or B must have D’s reference

What are the local object references?
Outline

- Distributed/Remote Objects
- Remote object reference (ROR)
- Remote Method Invocation (RMI)
- Case study and example: Java RMI
- Other issues for remote objects
  - Factory method; Transient vs. Permanent objects;
  - Callback objects; Distributed Garbage collection;

Remote Object Reference (ROR)

- Uniquely identify remote objects in distributed systems
- Needed to invoke remote method of a remote object
- Remote object references may be passed as input arguments or returned as output arguments.

Compared with ordinary object reference, what additional information is needed for remote object reference?

<table>
<thead>
<tr>
<th>Internet address</th>
<th>port number</th>
<th>time</th>
<th>object number</th>
<th>interface of remote object</th>
</tr>
</thead>
</table>

Parameters for Remote Methods

- Primitive types
  - pass by value
- Ordinary objects
  - passed by copy (e.g. using Java serialization; the object must implement the java.io.Serializable interface).
- Remote objects
  - use the remote object reference (ROR)

Why not pass a copy of the remote object?

1. The copy of a remote object can not perform its function
2. The object proxy and implementation is auto downloaded

Interfaces

- Interface for local objects
  - Specify methods and/or data that can be accessed
  - Do not specify an implementation
- Interface for remote objects
  - Specifies methods for remote invocation
  - Input and output parameters are also specified and parameters may be objects
Outline

- Distributed/Remote Objects
- Remote object reference (ROR)
- Remote Method Invocation (RMI)
- Case study and example: Java RMI
- Other issues for remote objects
  - Factory method; Transient vs. Permanent objects;
  - Callback objects; Distributed Garbage collection;

RPC Review

Middleware generates stubs on both sides

Object encapsulates data and operations
Object offers only its interface (group of methods) to clients.

How do clients know where the remote objects are?

- RMI register: the string name of the object, the remote object itself
- The registry returns to the caller a reference (called stub) to the remote object.
  - Invoke methods on the object (through the stub).

RMI Overview

Object server is responsible for a collection of remote objects
Skeleton (stub) handles (un)marshaling and object invocation
Proxy (stub) implements the same interface

Client Side: HelloClient

//HelloClient.java
import java.rmi.*;
import java.rmi.server.*;
public class HelloClient{
    public static void main(String[] argv) {
        try {
            HelloInterface hello = (HelloInterface) Naming.lookup("/localhost/Hello");
            System.out.println(hello.say());
            System.out.println(hello.add("Here is added information!!!"));
        } catch (Exception e) {
            System.out.println("HelloClient exception: " + e);
        }
    }
}
RPC vs. RMI

- **Similarity:** Marshaling and parameter passing
- **Difference:**
  - **RPC:** C based, structure based semantics. RMI: java and object-oriented
  - **RPC:** call remote functions, passed everything. RMI: remote/system-wide object reference and invoke methods. We can also pass and return objects that can be distributed among many JVM instances, much more powerful.
  - **RMI** can support dynamic invocations: a client can call a remote procedure even if the signature of the remote procedure or the name of the service are unknown until runtime.

RMI Dynamic Invocation

- **Dynamic invocation:** the method invocation is composed at run-time
  - `invoke(object, method, input_para, output_para)`
  - Useful for applications where object interfaces are discovered at run-time, e.g. object browser, batch processing systems for object invocations, “agents”

RMI Dynamic Invocation

```java
// Create method object, method name and parameter types
Method meth = dummyClass.getMethod("toString", partypes);
// parameter types for constructor
Class<?>[] constrpartypes = new Class[]{String.class, String.class};

// Create constructor object, parameter types
Constructor<?> constr = dummyClass.getConstructor(constrpartypes);

// Create instance
Object dummyto = constr.newInstance(new Object[]{"Java Programmer", "India"});

// Arguments to be passed into method
Object[] arglist = new Object[]{"I am"};

// Invoke method!
String output = (String) meth.invoke(dummyto, arglist);
```

Architecture to Support Remote Objects
Remote Reference Module (RRM)

- **Server side:**
  - Create remote object reference (ROR)
  - Maintain remote object reference table
  - Entry for remote objects
  - Map between remote reference and local reference

- **Client side:**
  - Create proxy object when first get ROR
  - Maintain remote object reference table
  - Entry for local proxy (client side)
  - Map between remote reference and local reference

Communication Module

- Carry out request-reply protocol
- Provide certain invocation semantics
  - Retry request
  - Duplication message filtering
  - Reply message history cache
- Interact with remote reference module (server side)
  - Get remote object’s local reference
  - Pass message and local reference to appropriate dispatcher on server side

Proxy and Skeleton

- Proxy - makes RMI transparent to client. Class implements remote interface. Marshals requests and unmarshals results. Forwards request.
- Skeleton - implements methods in remote interface. Unmarshals requests and marshals results. Invokes method in remote object.

Dispatcher

- Dispatcher - gets request from communication module and invokes method in skeleton (using methodID in message).
Middleware for Remote Objects

- Layer between application and communication/remote reference modules
- Automatically create proxy, skeleton and dispatcher from remote interface definition
- Client side: one proxy for each remote object
  - (kind-of) Implement the methods in remote objects, provide the transparency to the client
- Server side: one dispatcher and one skeleton
  - Dispatcher accepts message and select appropriate method in the skeleton: methodID
  - Skeleton: Marshall/unmarshall messages and invokes corresponding method in the remote object

Server/Client Programs and Binder

- Server program
  - Dispatcher, skeleton
  - Servant class: implement methods for remote objects
- Client program
  - Proxy
  - Use binder to get remote object reference
- Binder: kind of naming service
  - Mapping between text name and remote object reference
  - System wide register/look up service

Steps in RMI

How to Use Remote Objects: Server Side

- Step 0: start binder
- Step 1: server start communication and remote reference module
- Step 2: server create remote objects, add them to remote object table, and obtain remote object reference from remote reference module
- Step 3: server publish the remote objects to naming service → bind the remote object reference with a name; wait for invocation requests
Steps in RMI

1. client start communication and remote reference module
2. client contact naming service for desired remote object reference
3. client remote reference module create proxy
4. client call methods in proxy → mashall parameters
5. client locate remote object through remote reference module
6. Send method invocation request (contain remote object reference) through communication module

How to Use Remote Objects: Client Side

1. Step 4: client start communication and remote reference module
2. Step 5: client contact naming service for desired remote object reference
3. Step 6: client remote reference module create proxy
4. Step 7: client call methods in proxy → mashall parameters
5. Step 8: client locate remote object through remote reference module
6. Step 9: Send method invocation request (contain remote object reference) through communication module

How to Use Remote Objects: Server Side

1. Step 10: server get invocation requests (contains remote object reference) through communication module
2. Step 11: server consult with remote reference module and get local reference for the remote object
3. Step 12: server hand the request to the dispatcher/skeleton of the remote object’s class → which method (method id)
4. Step 13: server call method in skeleton → unmarshal parameters in request and invoke the real method of the remote object
5. Step 14: perform operation in remote object and return results to skeleton
6. Step 15: server marshall results in skeleton and send out message (remote object reference) through communication module
CS5523: Operating Systems @ UTSA

How to Use Remote Objects: Client Side

- Step 16: get result message (contain remote object reference) in communication module
- Step 17: obtain proxy reference from remote object reference and hand the result message to the proxy
- Step 18: unmarshall the results in proxy and return results to the caller

Complete a Remote Method Invocation

Why do people say RMI is simpler than sockets?

RMI = Sockets + Object Serialization + Some Utilities
User don’t need to worry about communication

Outline

- Distributed/Remote Objects
- Remote object reference (ROR)
- Remote Method Invocation (RMI)
- Case study and example: Java RMI
- Other issues for remote objects
  - Factory method; Transient vs. Permanent objects;
  - Callback objects; Distributed Garbage collection;

Case Study: Java RMI

- Define a remote interface
  - HelloInterface.java

Server side:
- Implement the interface
  - Hello.java
- Develop the server
  - HelloServer.java

Client side:
- Develop a client
  - HelloClient.java
- Run the RMI registry, the server, and the client

Server Class Implement Remote Interface

import java.rmi.*; //Hello.java
import java.rmi.server.*;

public class Hello extends UnicastRemoteObject implements HelloInterface {
    private String message;

    public Hello (String msg) throws RemoteException {
        message = msg;
    }

    public String add(String s) throws RemoteException {
        return message + s;
    }

    public String say() throws RemoteException {
        return message;
    }
}
Server: Create Servant Object and Bind

//HelloServer.java
import java.rmi.*;
public class HelloServer{
    public static void main(String args[]){
        try {
            Naming.rebind("Hello", new Hello("Hello, world!");
            System.out.println("Hello Server is ready.");
        } catch (Exception e) {
            System.out.println("Hello Server failed: " + e);
        }
    }
}

The binder: RMI Registry

For server
- Void rebind (String name, Remote obj)
  - Register an object by name
  - Override previous registration
- Void bind (String name, Remote obj)
  - Register an object by name
  - If existent throw exception
- Void unbind (String name, remote obj)

For Client
- Remote lookup (String name)
  - ROR is returned if found
- String[] list()
  - Show all names bound in this registry

Implementations of Client

- First, get remote object reference
  - It can retrieve a remote object reference from RMIregistry
    on the machine where the remote object resides
  - It received a remote object reference from a previous call

- Invoke methods specified in remote interface

- Compile the client code
  - Needs the interface file in order to compile
Client Side: HelloClient

```java
//HelloClient.java
import java.rmi.*;
import java.rmi.server.*;
public class HelloClient{
    public static void main (String[] argv) {
        try {
            HelloInterface hello = (HelloInterface) Naming.lookup("//localhost/Hello");
            System.out.println(hello.say());
            System.out.println(hello.add("Here is added information!!!");
        } catch (Exception e) {
            System.out.println("HelloClient exception: "+ e);
        }
    }
}
```

Java RMI: Compile and Run the Client

- Compile the interface class
  ```bash```
javac HelloInterface.java
```
- Compile client
  ```bash```
javac HelloClient.java
```
- Start Hello client
  ```bash```
java HelloClient [X]
```

> java HelloClient
Hello, world!
> java HelloClient
Hello, world!
> java HelloClient
Hello, world!
> java HelloClient
Hello, world!
> java HelloClient
Hello, world!
> java HelloClient
Hello, world!
```

Other Issues for Remote Objects

- When/who create remote objects?
  - Remote Object Factory method
- Can server contact client?
  - Call-back

Summary of Java RMI Steps

- Design the remote interfaces being implemented
- Server create and to register the remote objects.
  - Implement remote methods in the remote interfaces
  - Create and install a security manager
    - Regulates whether the server itself might download classes when accessing other remote servers
  - Create one or more instances of a remote object
  - Register at least one of the remote objects with the RMI remote object registry (or another naming service)
- Client program to use the methods of remote object
  - Looks up server in remote object registry and gets ROR
  - Use normal method call syntax for calling remote method
<table>
<thead>
<tr>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed/Remote Objects</td>
</tr>
<tr>
<td>Remote object reference (ROR)</td>
</tr>
<tr>
<td>Remote Method Invocation (RMI)</td>
</tr>
<tr>
<td>Case study and example: Java RMI</td>
</tr>
<tr>
<td>Other issues for remote objects</td>
</tr>
<tr>
<td>➢ Remote Object Factory</td>
</tr>
<tr>
<td>➢ RMI callbacks</td>
</tr>
<tr>
<td>➢ CORBA case study</td>
</tr>
<tr>
<td>➢ Transient vs. Permanent objects;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed/Remote Objects</td>
</tr>
<tr>
<td>Remote object reference (ROR)</td>
</tr>
<tr>
<td>Remote Method Invocation (RMI)</td>
</tr>
<tr>
<td>Case study and example: Java RMI</td>
</tr>
<tr>
<td>Other issues for remote objects</td>
</tr>
<tr>
<td>➢ Remote Object Factory</td>
</tr>
<tr>
<td>➢ RMI callbacks</td>
</tr>
<tr>
<td>➢ CORBA case study</td>
</tr>
<tr>
<td>➢ Transient vs. Permanent objects;</td>
</tr>
</tbody>
</table>