Java Remote Method Invocation
RMI

Dave Price

Computer Science Department

University of Wales, Aberystwyth

Java RMI

- Provides a mechanism whereby objects active within one Java Virtual Machine (JVM) may invoke methods on other objects active within another JVM
- The JVMs may be running on the same underlying host computer or on different host computers connected by an Internet connection.

Java RMI

- Somewhat like the Remote Procedure Calls sometimes provided by other procedural languages.
- Java RMI only works between two Java programs, can't be used between Java and a program running in another language.
- (from Java 2 version 1.2 allows interaction with CORBA too).

Server and Client

- It is normal to refer to the program which has methods which can be invoked remotely as the SERVER
- It is also normal to refer to the program which invokes those remote methods as the CLIENT
- During these slides if I use the word “local” I'll mean things in the client and “remote” will mean things in the server.

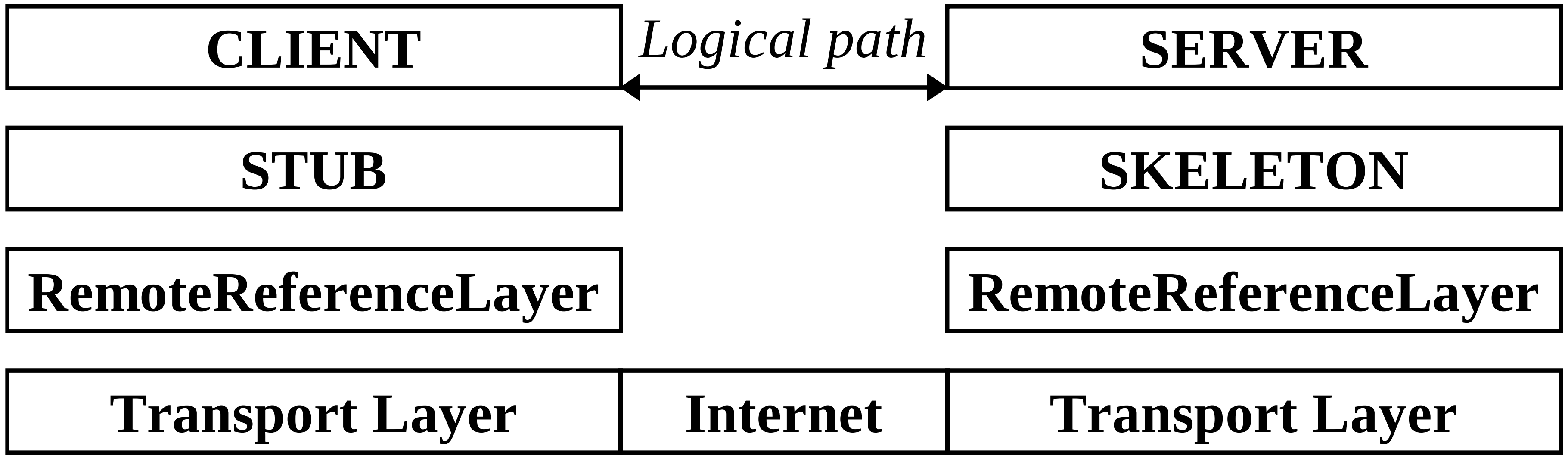
Parameters passed in Remote Method Invocations

- All data passed must either
- be of primitive type
- or be references to local objects which implement the serializable interface
- or be references to remote objects

Java Packages

- Most of the things that are specific to RMI are contained in either …
- java.rmi most things need by clients
- java.rmi.server extra things servers need
- java.rmi.registry see “registry” later
- some more java.rmi.* things we will not worry about

RMI Architecture



Finding Each Other - RMIRegistry

- The client and server find each other via the use of an RMIRegistry
- The server registers with the registry by “binding” to the name of a service to which clients may connect
- The clients ask the registry for a remote reference to an object providing a named service

Server Issues

A Remote object

- is defined as being any object that implements the Java interface java.rmi.Remote

or

- any object that implements an interface which itself extends the interface java.rmi.Remote

and

- remote class normally defined to extend java.rmi.UnicastRemoteObject

Server Methods

- The remote object can have both methods which can only be invoked by the server itself as well as methods which can be invoked via RMI from a client
- methods which can be invoked by RMI need to be declared as “throws RemoteException”
- java.rmi.RemoteException is the superclass of most exceptions that can be thrown when RMI is being used.

An example Interface - RemInt.java

```
public interface RemInt extends java.rmi.Remote
  {

    public boolean setSession(String s1)
          throws java.rmi.RemoteException;

    public String getText()
          throws java.rmi.RemoteException;
}
```

My Server - RemServ.java

- has two methods callable via RMI

- first allows the client to set a string used later by the server
- second asks the server for some text

- Note: it uses the rebind method of the Naming class to register with the RMIRegistry

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

public class RemServ extends
     UnicastRemoteObject implements RemInt {
    String nameSession;

  public static void main(String args[])
    throws RemoteException {
                 new RemServ();
  }
```

```
public RemServ() throws RemoteException {

 // Create and install a security manager
 System.setSecurityManager(new
              RMISecurityManager());
 try {
  Naming.rebind(
   "rmi://moin.dcs.aber.ac.uk:5000/RemServer",
            this);
 } catch (Exception e) {

   System.out.println("RemServ err: " +
                      e.getMessage());
   e.printStackTrace();
 }
 System.out.println(
        "RemServ: I'm registered");
}
```

```
 public boolean setSession(String nameSess ){
    nameSession = nameSess;
    return true;
 }

 public String getText() {

  try {
     return nameSession + " Hello";
  } catch (Exception e) {
     System.out.println("goPublic err: " +
                        e.getMessage());
     e.printStackTrace();
     return "Broken";
  }
 }
}
```

My Client - Client.java

- creates an object of type Client
- and then from it’s constructor (not very good style here…..)
- contacts the registry to get a remote object
- converts it to the right type
- uses setSession to pass a message to server
- asks server for some text
- displays it to the user

```
import java.rmi.*;
import java.rmi.RMISecurityManager;

public class Client{
  public static void main(String args[]) {
         new Client();
  }
```

```
public Client() {
 String replyMessage;

 // Create and install a security manager
 System.setSecurityManager(new
             RMISecurityManager());

 try {

  Object objEng =
  Naming.lookup(
  "rmi://moin.dcs.aber.ac.uk:5000/RemServer");

  RemInt remobjEng = (RemInt)objEng;

  remobjEng.setSession("It's me friend");
```

```
   replyMessage = remobjEng.getText();

   System.out.println("He say " +
                       replyMessage);

  } catch (Exception e) {
    System.out.println("Client problem " +
    e.getMessage());
    e.printStackTrace();
  }
 }
}
```

Some Proof that it Works !!

```
Script started on Wed 21 Apr 1999
 09:32:34 PM BST
moin%
moin% rmiregistry 5000&
[1] 16337
moin% java RemServ&
[2] 16348
moin%
```

And the Client

```
moin% java Client
He say It's me friend Hello
moin%
script done on Wed 21 Apr 1999
  09:33:09 PM BST
```

Security Issues...

- Clearly there are some, we have a client invoking methods on a server
- would normally set the security manager to be a RMISecurityManager for applications
- I’ll do more about RMI and SecurityManagers later in the module
- FWL will say a bit more about SecurityManagers

More about the Naming Registry

- supports some extra methods…
- bind … like rebind but fails if name in use
- unbind … obvious
- list …. lets you ask a registry for what names it has registered servers

Stubs and Skeletons

- I talked about these earlier
- “rmic - The Java RMI Stub Compiler” will create these automatically for you
- rmic RemServ
- stubs can be automatically downloaded across the network when needed so that they are available to the client

Conclusions

- Really quite simple to use
- Nice mechanism for building distributed applications
- Java <-> Java only not mixed language (except remember CORBA remark)
- Look at CS25610 web site for some pointers to various Sun tutorials with more complicated examples and the RMI APIs