

Remote Method Invocation

Netprog Java 2001 - RMI

1

Network Programming Paradigms

- Sockets programming: design a protocol first, then implement clients and servers that support the protocol.
- RMI: Develop an application, then move some objects to remote machines.
 - Not concerned with the details of the actual communication between processes – everything is just method calls.

Netprog Java 2001 - RMI

2

Call Semantics

- Method Call Semantics – what does it mean to make a call to a method?
 - How many times is the method run?
 - How do we know the method ran at all?
- RMI does a great job of providing *natural* call semantics for remote objects/methods.
 - Simply a few additional Exceptions that you need to handle.

Netprog Java 2001 - RMI

3

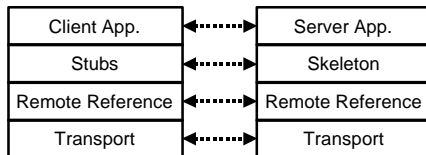
Finding Remote Objects

- It would be awkward if we needed to include a hostname, port and protocol with every remote method invocation.
- RMI provides a *Naming Service* through the RMI Registry that simplifies how programs specify the location of remote objects.
 - This naming service is a JDK utility called **rmiregistry** that runs at a well known address (by default).

Netprog Java 2001 - RMI

4

RMI Adds a few layers



Netprog Java 2001 - RMI

5

Remote Object References

- The client acquires a reference to a remote object.
 - This part is different from creating a local object.
- The client calls methods on the remote object
 - No (syntactic) difference!
 - Just need to worry about a few new exceptions.

Netprog Java 2001 - RMI

6

Overview of RMI Programming

- Define an *interface* that declares the methods that will be available remotely.
- The *server* program must include a *class* the implements this *interface*.
- The *server* program must create a remote object and register it with the naming service.
- The *client* program creates a remote object by asking the naming service for an object reference.

Netprog Java 2001 - RMI

7

Server Details – extending Remote

- Create an interface the extends the `Remote` interface.
 - This new interface includes all the public methods that will be available as remote methods.

```
public interface MyRemote extends Remote {  
    public int foo(int x) throws RemoteException;  
    public String blah(int y) throws RemoteException;  
    . . .  
}
```

Netprog Java 2001 - RMI

8

Server Details – Implementation Class

- Create a class that implements the interface.
 - The class should also extend `UnicastRemoteObject`*
- This class needs a constructor that throws `RemoteException` !
- This class is now used by `rmic` to create the stub and skeleton code.

*It doesn't have to extend `UnicastRemoteObject`, there is another way...

Netprog Java 2001 - RMI

9

Remote Object Implementation Class

```
public class MyRemoteImpl extends
    UnicastRemoteObject implements MyRemote {

    public MyRemoteImpl() throws RemoteException {
    }

    public int foo(int x) {
        return(x+1);
    }
    public String blah(int y) {
        return("Your number is " + y);
    }
}
```

Netprog Java 2001 - RMI

10

Generating stubs and skeleton

- Compile the remote interface and implementation:

```
> javac MyRemote.java MyRemoteImpl.java
```

- Use `rmic` to generate `MyRemoteImpl_stub.class`,
`MyRemoteImpl_skel.class`

```
> rmic MyRemoteImpl
```

Netprog Java 2001 - RMI

11

Server Detail – main()

- The server `main()` needs to:
 - create a remote object.
 - register the object with the Naming service.

```
public static void main(String args[]) {
    try {
        MyRemoteImpl r = new MyRemoteImpl();
        Naming.bind("joe",r);
    } catch (RemoteException e) {
        . . .
    }
}
```

Netprog Java 2001 - RMI

12

Client Details

- The client needs to ask the naming service for a reference to a remote object.
 - The client needs to know the hostname or IP address of the machine running the server.
 - The client needs to know the name of the remote object.
- The naming service uses URLs to identify remote objects.

Netprog Java 2001 - RMI

13

Using The Naming service

- **Naming.lookup()** method takes a string parameter that holds a URL indicating the remote object to lookup.

```
rmi://hostname/objectname
```

- **Naming.lookup()** returns an **Object!**
- **Naming.lookup()** can throw
 - **RemoteException**
 - **MalformedURLException**

Netprog Java 2001 - RMI

14

Getting a Remote Object

```
try {
    Object o =
        naming.lookup("rmi://monica.cs.rpi.edu/joe");

    MyRemote r = (MyRemote) o;

    . . . Use r like any other Java object! . . .
} catch (RemoteException re) {
    . . .
} catch (MalformedURLException up) {
    throw up;
}
```

Netprog Java 2001 - RMI

15

Starting the Server

- First you need to run the Naming service server:
`start rmiregistry`
- Now run the server:
`java ServerMain`

Netprog Java 2001 - RMI

16

Sample Code

- There is sample RMI code on the course homepage:
 - SimpleRMI: remote integer arithmetic
 - AlternativeSimpleRMI: Implementation class doesn't extend UnicastRemoteObject
 - RemoteSort: remote sort server – uses Java List objects

Netprog Java 2001 - RMI

17
