Java Remote Method Invocation
RMI - Activation

Dave Price
Computer Science Department
University of Wales, Aberystwyth

Java RMI - Activation

- Provides a way to have RMI accessible objects without having to leave the RMI server program running all the time
- Uses an “RMI activation demon” to deal with activating objects on demand when they are required
- Provides “persistence” in the sense that it makes objects available over server crashes

Java RMI - Activation Demon

- Activation demon provided as a program called “rmid” with the JDK.
- “rmid” can also provide “registry” services if required

RMI Server

- Remote Objects now normally extend `java.rmi.activation.Activatable` rather then UnicastRemoteObject
- Remote Objects have to provide constructors with specific signatures that are invoked when the Remote Objects are activated or re-activated
- The parameter shown as in the example as `MarshalledObject data` can be used to initialise the Remote Object

An Example Activatable Object

```
import java.rmi.*;
import java.rmi.activation.*;

public class ActivateRemServ extends
Activatable implements RemInt {

    String nameSession;
```

```
public ActivateRemServ(ActivationID id,
MarshalledObject data)
  throws RemoteException {
  // Register the object with the
             activation system
  // then export it on an anonymous port
  super(id, 0);
  System.out.println(
    "ActivateRemServ: registered"
    + " with activation service and exported");
}
```

```
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";
        }
    }

}
```

Registering the Activatable Object

- We can register an object as Activatable without a copy being in existence
- We write another “setup” class
- We need to define an “activation group” and we register this with the activation service
- Then we associate our class with the activation group. It returns a “stub” which we then register with the RMI registry

Registering the Activatable Object

```
import java.rmi.*;
import java.rmi.activation.*;
import java.util.Properties;

public class SetupRemServ {

    // This class registers information about
the ActivateRemServ
    // class with rmid and the rmiregistry
    //
    public static void main(String[] args)
throws Exception {

        System.setSecurityManager(new
RMISecurityManager());
```

```
Properties props = new Properties();
        props.put("java.security.policy",

"/dcs/dap/dap_extra/cs25610/rmi/ACTIVATE/policy_
all.txt");

  ActivationGroupDesc.CommandEnvironment ace =
              null;
  ActivationGroupDesc exampleGroup = new
           ActivationGroupDesc(props, ace);

  // Once the ActivationGroupDesc has been
  // created, register it
  // with the activation system to obtain its ID
  //
  ActivationGroupID agi =
    ActivationGroup.getSystem().registerGroup(
            exampleGroup);
```

```
// Now explicitly create the group
        //
        ActivationGroup.createGroup(agi,
                exampleGroup, 0);
```

```
// Don't forget the trailing slash at the end
// of the URL
// or your classes won't be found
//
String location =
     "http://www.aber.ac.uk/~dap/Java/";

// Create the rest of the parameters that
// will be passed to
// the ActivationDesc constructor
//
MarshalledObject data = null;
```

```
// The second argument to the ActivationDesc
// constructor will be used
// to uniquely identify this class; it's
// location is relative to the
// URL-formatted String, location.
//
ActivationDesc desc = new ActivationDesc(
        "ActivateRemServ", location, data);

RemInt mri =
     (RemInt)Activatable.register(desc);
System.out.println(
    "Got the stub for the ActivateRemServ ");
```

```
// Bind the stub to a name in the registry
// running on 1099
        //
        Naming.rebind("ActivateRemServ", mri);
        System.out.println(
           "Exported ActivateRemServ");

        System.exit(0);
    }
}
```

An example Interface - RemInt.java - NOTE same as before!

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

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

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

Client - in essence same as before

```
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/ActivateRemServ");

 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();
 }
 }
}
```

Summary

- saves leaving lots of servers running
- provides resilience across crashes
- clients exactly the same
- servers a bit different
- needs an “activation demon” - rmid
- still needs a registry - rmiregistry or rmid can provide this