Grid Computing

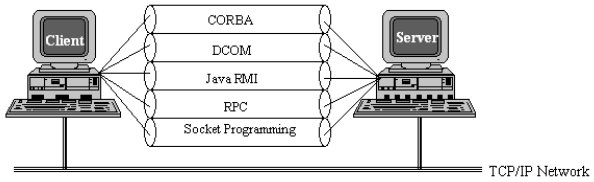
Paradigms for Distributed Computing 2 RMI

Paul A. Farrell
Fall 2006

The Grid: Core Technologies
Maozhen Li, Mark Baker
John Wiley & Sons; 2005, ISBN 0-470-09417-6

Including material by Amy Apon at U. Arkansas

Traditional paradigms for distributed computing

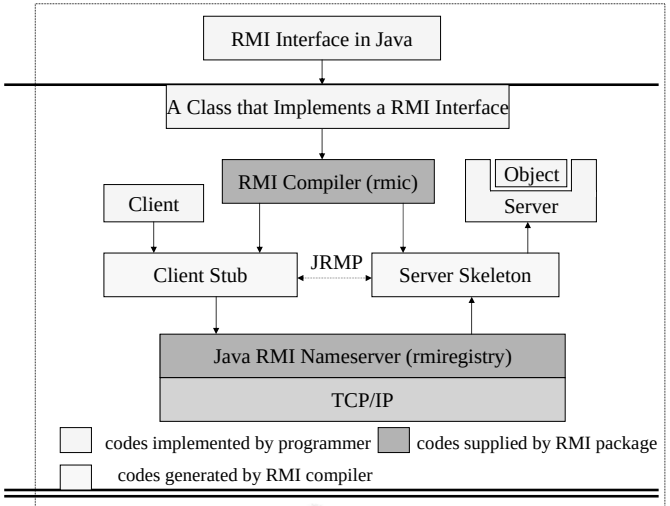


Java Remote Method Invocation (RMI)

- The Java RMI is an object-oriented mechanism from Sun Microsystems for building distributed client/server applications
- Uses a client-side stub and a server-side skeleton (which is not needed in Java 1.2 or later) that are automatically generated from a class that extends *java.rmi.UnicastRemoteObject* and implements a RMI *Remote* interface
- RMI allows us to distribute our objects on various machines, and invoke methods on the objects located on remote sites.

RMI Application

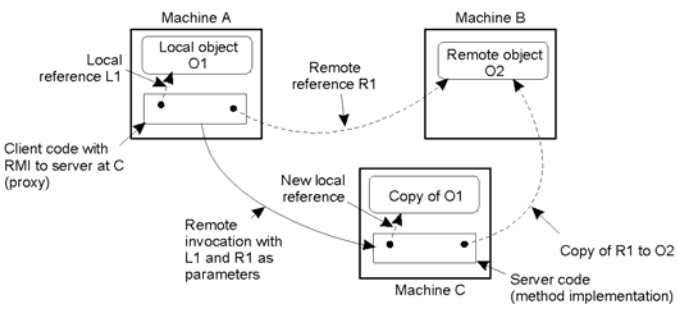
- Three entities
- A client that invokes a method on a remote object.
- A server that runs the remote object, which is an ordinary object in the address space of the server process.
- The object registry (*rmiregistry*), which is a name server that relates objects with names. Remote objects need to be registered with the registry. Once an object has been registered, the registry can be used to obtain access to a remote object using the name of that object.



Steps to implement and run using RMI

- Write a RMI interface.
- Write a RMI object to implement the interface.
- Use RMI compiler (*rmic*) to compile the RMI object to generate a client-side stub and a server-side skeleton.
- Write a RMI server to register the RMI object.
- Write a RMI client.
- Use Java compiler (*javac*) to compile all the Java source codes.
- Start the RMI name server (*rmiregistry*).
- Start the RMI server.
- Start the RMI client.

Parameter Passing



RMI advantages

- Uses an object-oriented approach, compared to the procedural one that RPC uses.
- Client can pass an object as a parameter to a remote object.
- Unlike RPC which needs an IDL interface, a Java RMI interface is written in Java.
- RMI has good support for marshalling, which is a process of passing parameters from client to a remote object, i.e., a *Serializable* Java object can be passed as a parameter.

More Details

- Once the object (or service) is registered, a client can look up that service.
- A client (application) receives a reference that allows the client to use the service (call the method).
- Syntax of calling is identical to a call to a method of another object in the same program.

Case Study : Calculator

- Let's create a distributed system using RMI model for networking (remote access)
- Server will implement calculator functions of square and power for arbitrary length integers (BigInt)

Defining Remote Interface

```
import java.rmi.*;
// the interface extends Remote interface
// any class implementing Remote can be accessed remotely security
   permitting
public interface TemperatureServer extends Remote
{  // specify methods that can be called remotely
   // each method "throws RemoteException"
}
```

- Any time you depend on outside entities there is a potential for problems in communication, networking, server crash etc.
- Any exception due to these should be handled by the services.
- This feature imparts robustness to the application.
- Java mandates this feature for any RMI service.

Implementing the Remote Interface

```
import java.math.BigInteger;
import java.rmi.*;

//
// PowerService Interface
//
// Interface for a RMI service that calculates powers
//
public interface PowerService extends java.rmi.Remote
{
      // Calculate the square of a number
      public BigInteger square ( int number )
            throws RemoteException;

      // Calculate the power of a number
      public BigInteger power  ( int num1, int num2)
            throws RemoteException;
}
```

PowerServiceServer.java

- This class's constructor calls a private method which in turn:
  - Implements the square and power methods
- The main method
  - Assigns a security manager, in the event that dynamic classes are loaded
  - instantiates an object for the service, and
  - registers it with rmiregistry using bind or rebind

Name Binding

- rebind method binds a server's object name to the object's name as it is in the registry.
- Clients use the name in the registry.
- There is also a bind() method.
- But rebind is better since it binds the most recently registered object.

Server Object Name

- Syntax for the server object name is:
  //host:port/remoteObjectName
- Default port number for rmiregistry is 1099
- For local host the object name:
  //localhost/TempServer
- For a remote host
  //127.0.0.1/TempServer

PowerService Client

```
import java.rmi.*;
import java.rmi.Naming;
import java.io.*;
```

- Assigns a security manager
- Calls registry for PowerService

  ```
  PowerService service = (PowerService) Naming.lookup
  ("rmi://" + args[0] + "/PowerService");
  ```
- Displays prompts, takes input and invokes the appropriate method in service

Client Details

- The name of the server object along with the IP of the remote location is used in Naming class's lookup method to get an object reference.
- This object reference is then used for remote method calls.
- Observe that there is no difference between the local and remote call.

Preparing the Application

1. Compile all the classes using **javac *.java**
2. Generate the stub and the skeleton:
   rmic –d . PowerServiceServer
3. Copy the classes to the public_html directory
   cp *.classes ~/username/public_html
4. Then start the registry (this will be running as a daemon)
   rmiregistry &

Preparing the Application

- 5. Run the server which will register with the RMI registry.
- java -Djava.rmi.server.codebase=http://www.cs.kent.edu/~farrell/sys2002/RMI/try3/ -Djava.security.policy=java.policy -Djava.rmi.server.hostname=arakis.cs.kent.edu PowerServiceServer
- 6. Run the client.

setenv CLASSPATH /users/cs/faculty/farrell/public_html/sys2002/RMI/try3

java -Djava.rmi.server.codebase=http://www.cs.kent.edu/~farrell/sys2002/RMI/try3/ -Djava.security.policy=java.policy PowerServiceClient arakis.cs.kent.edu

Summary

- We discussed the various models of distributed systems.
- Java RMI was used to illustrate the distributed system concepts.
- A more general description of the process is at
  - http://java.sun.com/docs/books/tutorial/rmi/index.html