Creating a Distributed System with RMI

B. Ramamurthy

Remote Method Invocation

- Remote Method Invocation (RMI) is Java’s implementation of object-to-object communication among Java objects to realize a distributed computing model.
- RMI allows us to distribute our objects on various machines, and invoke methods on the objects located on remote sites.
- Source code for the demo is a modified version of code in Chapter 20 of Deitel & Deitel’s text Java: How to Program.

Remote Method Invocation

Steps in RMI-based Application

1. Design the interface for the service.
2. Implement the methods specified in the interface.
3. Generate the stub and the skeleton.
4. Register the service by name and location.
5. Use the service in an application.

Compile and Register Commands

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.
Parameter Marshalling

- Transfer of parameters (or marshalling) is done by the RMI.
- Complex objects are streamed using Serialization.
- RMI model of networking for distributed system involves only Java.
- No need to learn IDL or any other language.

Case Study: Temperature Service

- Lets create a distributed system using RMI model for networking (remote access).
- Basically this program will download the weather (temperature) information from the site: http://iwin.nws.noaa.gov/iwin/us/traveler.html

Temperature Client/Server Distributed Application

Defining Remote Interface

```java
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”
}
```

Implementing the Remote Interface

```java
import java.rmi.*;
import java.rmi.server.*;
import java.net.*;
// others as needed
TemperatureServerImpl extends UnicastRemoteObject
    implements TemperatureServer {
```
TemperatureServerImpl

◆ This class's constructor calls a private method which in turn:
  1. Connects to the url specified
  2. Streams into a buffer the page referenced.
  3. Parses the buffer to get the required data.
  4. Creates an array of weather information.

TemperatureServerImpl (contd.)

◆ It implements the service method getWeatherInfo which simply returns the weather data gathered.
◆ The main method instantiates an object for the service, and registers it with rmiregistry.

Streaming URLs

◆ Using the openStream of java.net.URL class you can stream in the file specified by a universal resource locator(url).
◆ It can be streamed into a buffer where it can be analyzed for information.
◆ Any number of urls can be streamed in.
◆ Unicast Communication: When you are interested in a particular remote site you will direct your net connection to that particular site using unicast.

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

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.

WeatherInfo class

◆ It is very traditional class for keeping the information about the temperature at a single location.
◆ It has data fields: cityName, temperature, and description and get methods for these.
◆ An array of objects of this class is used in the server implementation.
Temperature Client

import java.rmi.*;
// import other packages
constructor calls a private method
getRemoteTemp which takes care of
lookup of remote object and access.
In this application it also displays the
information.

Temperature Client (contd.)

◆The main method in this client can get
the IP address of the remote host as a
command line argument.
◆Command line argument is an array of
String of items in the command line
after the name of the application.

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.
◆WeatherItem class used in the Graphical
display of the weather information.

Preparing the Application

4. Run the server which will register with the
RMI registry.
Java TemperatureServerImpl &
5. Run the client.
Java TemperatureClient &
or
java TemperatureClient {IPAddress}
java TemperatureClient 192.168.0.150

Preparation of Application

1. Compile all the class using javac.
2. Generate the stub and the skeleton:
   rmic -v1.2 TemperatureServerImpl
3. Then start the registry (this will be
   running as a daemon)
   rmiregistry &

Summary

◆We discussed the various models of
distributed systems.
◆Java RMI was used to illustrate the
distributed system concepts.
◆Temperature examples shown
illustrates some of the distributed
system model discussed and all the
important RMI features.