CS 580 Client-Server Programming
Spring Semester, 2005
Doc 20 RPC

References

http://www.xmlrpc.com/ Main XML_RPC web site

http://davenet.userland.com/1998/07/14/xmlRpcForNewbies XML_RPC For Newbies

http://xml.apache.org/xmlrpc/ Home page for Java XML-RPC implementation

RPC
Remote Procedure Call

A client can "directly" call a function or procedure on the server

Issues

- Cross platform
  Primitive data types may be different on client & server

- Marshalling/unmarshalling of parameters and results
  Procedure parameters must be sent from client to server
  How can one handle pointers as parameters?
  Result of procedure call must be sent back to client

- Different contexts of client and server

- Registering and finding servers

Sample Uses

Unix NFS (Network File System)
Unix license managers

RPC implementations

SUN RPC
Distributed Computing Environment (DCE)

XML-RPC

RPC using
- HTTP as transport layer and
- XML to encode request/response
- Language and platform independent

Started by Userland (http://frontier.userland.com/) in 1998

Languages/Systems with XML-RPC implementations

- Java, Perl, Python, Tcl, C, C++, Smalltalk
- ASP, PHP, AppleScript, COM
- Zope, WebCrossing

Led to the development of SOAP

Java Example

```
import java.util.*;
import org.apache.xmlrpc.*;

public class XmlRpcExample
   {
  public static void main (String args[])
      {
      try
         {
         XmlRpcClient xmlrpc =  new XmlRpcClientLite
               ("http://xmlrpc.usefulinc.com/demo/server.php");
        Vector parameters = new Vector ();
         parameters.addElement (new Integer(5) );
         parameters.addElement (new Integer(3) );
         Integer sum =
            (Integer) xmlrpc.execute ("examples.addtwo",
                                      parameters);
         System.out.println( sum.intValue() );
         }
      catch (java.net.MalformedURLException badAddress)
         {
         badAddress.printStackTrace( System.out);
         }
      catch (java.io.IOException connectionProblem)
         {
         connectionProblem.printStackTrace( System.out);
         }
      catch (Exception serverProblem)
         {
         serverProblem.printStackTrace( System.out);
         }
      }
   }
```

Issues

Client program has to know

- Server machine name or IP (xmlrpc.usefulinc.com)
- Path to server program (/demo/server.php)
- Name of remote method (examples.addtwo)
- Number, Type and Order of arguments

Supported Data Types

XML-RPC data type	Java
<i4> or <int>	java.lang.Integer
<boolean>	java.lang.Boolean
<string>	java.lang.String
<double>	java.lang.Double
<dateTime.iso8601>	java.util.Date
<struct>	java.util.Hashtable
<array>	java.util.Vector
<base64>	byte[ ]

How do you know about methods on the Server?

The server:

- url
- method name
- method arguments

Need to be documented some place

How does this work?

Client marshals (serialize) the rpc request

Converts the requests in to a format that can be sent on the network

Client

- Sends the marshaled version to the server
- Waits for server response

Server

- Unmarshals the request,
- Runs the requested method
- Marshals the result
- Send the marshaled result back to the client

Client unmarshals the result

Complete Request sent to Server

```
POST /demo/server.php HTTP/1.1
Host: xmlrpc.usefulinc.com
Content-length: 190
Content-type: text/xml;charset=iso-8859-1
User-Agent: Smalltalk XMLRPC version 0.5 (VisualWorksÆ NonCommercial,
Release 7 of June 14, 2002)
Connection: keep-alive

<?xml version="1.0"?>
<methodCall>
   <methodName>examples.addtwo</methodName>
   <params>
      <param>
         <value><int>5</int></value>
      </param>
      <param>
         <value><int>3</int></value>
      </param>
   </params>
</methodCall>
```

XmlRpc ServersJava Example

The following starts an addtwo server on port 8080
Server URL is serverMachinename:8080
Method name is: examples.addtwo

How come the server is still running after the last println?

```
import org.apache.xmlrpc.*;

public class JavaServer
   {
   public Integer addtwo(int x, int y)
      {
      return new Integer( x + y);
      }

   public static void main( String[] args)
      {
      try
         {
         System.out.println("Starting server on port 8080");
         WebServer addTwoServer = new WebServer(8080);
         addTwoServer.addHandler("examples", new JavaServer());
         System.out.println("server running");
         }
      catch (Exception webServerStartError)
         {
         System.err.println( "JavaServer " +
                  webServerStartError.toString());
         }
      }
   }
```

Notice

We have not explicitly handled sockets in any example

RMI

Java's Remote Method Invocation

Allows easy communication between remote Java VMs

Hello World Example

Modified from "Getting Started Using RMI"

The Remote Interface

```
public interface Hello extends java.rmi.Remote
  {
  String sayHello() throws java.rmi.RemoteException;
  }
```

The Server Implementation

```
// Required for Remote Implementation
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

// Used in method getUnixHostName
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class HelloServer
      extends UnicastRemoteObject
      implements Hello
   {

   public HelloServer() throws RemoteException
      {
      }

   // The actual remote sayHello
   public String sayHello() throws RemoteException
      {
      return  "Hello World from " + getUnixHostName();
      }
```

// Works only on UNIX machines

```
protected String getUnixHostName()
  {
  try
    {
    Process  hostName;
    BufferedReader  answer;

    hostName = Runtime.getRuntime().exec( "hostname" );
    answer = new BufferedReader(
                 new InputStreamReader(
                   hostName.getInputStream()) );

    hostName.waitFor();
    return answer.readLine().trim();
    }
  catch (Exception noName)
    {
    return "Nameless";
    }
  }
```

// Main that registers with Server with Registry

```
public static void main(String args[])
  {
  // Create and install a security manager
  System.setSecurityManager(new RMISecurityManager());

  try
    {
    HelloServer serverObject = new HelloServer ();

    Naming.rebind("//roswell.sdsu.edu/HelloServer",
                  serverObject );

    System.out.println("HelloServer bound in registry");

    }
  catch (Exception error)
    {
    System.out.println("HelloServer err: ");
    error.printStackTrace();
    }
  }
}
```

The Client Code

```
import java.rmi.*;
import java.net.MalformedURLException;

public class HelloClient
  {
  public static void main(String args[])
    {
    try {
      Hello remote = (Hello) Naming.lookup(
                          "//roswell.sdsu.edu/HelloServer");

      String message = remote.sayHello();
      System.out.println( message );
      }
    catch ( Exception error)
      {
      error.printStackTrace();
      }
    }
  }
```

Note the multiple catches are to illustrate which exceptions are thrown

Running The Example
Server Side

Step 1. Compile the source code

Server side needs interface Hello and class HelloServer

javac Hello.java HelloServer.java

Step 2. Generate Stubs and Skeletons (to be explained later)

The rmi compiler generates the stubs and skeletons

rmic HelloServer

This produces the files:

HelloServer_Skel.class
HelloServer_Stub.class

The Stub is used by the client and server
The Skel is used by the server

The normal command is:

rmic fullClassname

Step 3. Insure that the RMI Registry is running

For the default port number

```
rmiregistry &
```

For a specific port number

```
rmiregistry portNumber &
```

On a UNIX machine the rmiregistry will run in the background and will continue to run after you log out

This means you manually kill the rmiregistry

Step 4. Register the server object with the rmiregistry by running HelloServer.main()

```
java HelloServer &
```

Client Side

The client can be run on the same machine or a different machine than the server

Step 1. Compile the source code

Client side needs interface Hello and class HelloClient

javac Hello.java HelloClient.java

Step 2. Make the HelloServer_Stub.class is available

Either copy the file from the server machine

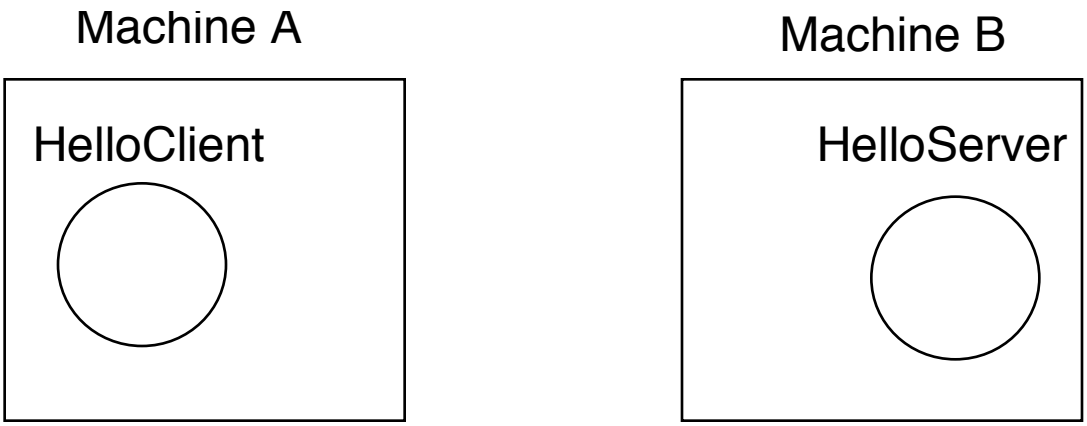
or

Compile HelloServer.java on client machine and rum rmic

Step 3. Run the client code

java HelloClient

Proxies
How do HelloClient and HelloSever communicate?



Client talks to a Stub that relays the request to the server over a network

Server responds via a skeleton that relays the response to the Client