CS 580 Client-Server Programming
Fall Semester, 2000
Doc 25 RPC, XML-RPC, SOAP, CORBA
Contents

Classical Client-Server 2
RPC 4
XML-RPC 5
  Example 7
SOAP 10
Corba 11
  Interface Definition Language (IDL) 13
    Corba IDL Types 14
    Attributes and Operations 17
    Parameters types 19
    Inheritance 20
    Name Scopes 21
  A Simple Example 22
  CORBA Services 30

References

Object Management Group (OMG) http://www.omg.org

Client/Server Programming with Java and CORBA, Orfali and Harkey, John Wiley & Sons, Inc., 1997

Java Programming with CORBA, Vogel and Duddy, John Wiley & Sons, Inc., 1997

XML_RPC Web Site http://www.xmlrpc.com/

SOAP http://soap.weblogs.com/ and http://www.w3.org/TR/SOAP/

Apache SOAP http://xml.apache.org/soap/

Microsoft SOAP http://msdn.microsoft.com/soap/

Classical Client-Server

Client and server are separate programs

Protocol specifies the communication between client & server

Client -server communication

- Normally uses ASCII text
- Uses TCP/UDP

Advantages

Language independent

Platform independent

Client & Server can be developed independently

Disadvantages

Requires low level work:

Parsing

Network communications

Other Approaches

Protocol parsing & network communication can be automated

- RPC
- CORBA
- RMI
- XML RPC
- SOAP

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
  XDR (eXternal Data Representation) used to send data

- Marshalling/unmarshalling of parameters and results
  Procedure call parameters on the client must be sent 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) RPC

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,
- ASP, PHP, AppleScript, COM
- Zope, WebCrossing

XMP-RPC Protocol

Sample Request

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
      <param>
         <value><i4>41</i4></value>
      </param>
   </params>
</methodCall>
```

Uses http to send request
Body contains XML describing the request

See http://www.xmlrpc.com/spec for details

Example

Using the Java implementation from Hannes Wallnöfer

See http://classic.helma.at/hannes/xmlrpc/

Server will just add two numbers

Adder

```
public class Adder {
   public int add(int a, int b) {
      return a + b;
   }
}
```

Server

```
import helma.xmlrpc.WebServer;

public class XmlRpcServer  {
   public static void main(String args[]) throws Exception {
      WebServer server = new WebServer( 5432 );
      server.addHandler( "MyAdd", new Adder());
      System.out.println( "Starting Server" );
      server.run();
   }
}
```

Could put WebServer in a thread to run it as a separate thread

Client

```
import helma.xmlrpc.XmlRpcClientLite;
import java.util.Vector;

public class XmlRpcClient {
   public static void main(String args[]) throws Exception {
      XmlRpcClientLite client =
         new XmlRpcClientLite("rohan.sdsu.edu", 5432);
      Vector parameters = new Vector();
      parameters.addElement( new Integer( 5));
      parameters.addElement( new Integer( 2));
      Object sum = client.execute( "MyAdd.add", parameters);
      System.out.println( sum);
   }
}
```

Output

```
7
```

Client Sends

```
<?XML VERSION="1.0"?>
<methodCall>
  <methodName>MyAdd.add</methodName>
  <params>
    <param>
      <value><int>5</int></value>
      </param>
    <param>
      <value><int>2</int></value>
      </param>
    </params>
  </methodCall>
```

Server Response

```
HTTP/1.0 200 OK
Server: Helma XML-RPC 1.0
Connection: close
Content-Type: text/xml
Content-Length: 135

<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
  <params>
    <param><value><int>7</int></value></param>
  </params>
</methodResponse>
```

SOAP
Simple Object Access Protocol

Developed by Userland, Microsoft, IBM, Lotus

Super set of XML-RPC

Central part of Microsoft's .NET

More complex than XML-RPC

Implementations in:
Java, Perl, C#, Frontier, C++, Visual Studio

The next Big Thing?

Corba

Common Object Request Broker Architecture

CORBA provides peer-to-peer distributed computing between objects

Client - the role of making a request of some other object in the distributed program

Server - the role of providing an implementation of a object that a client uses

Transparencies

Location transparency

A remote object appears the same as a local object

An Object Request Broker (ORB) is used to locate objects

Language transparency -
Remote objects do not have to be implemented in the same language as local objects

An Interface Definition Language (IDL) is used to define the interface to each object in a language independent way

Interface Definition Language (IDL)

Corba objects are defined by an IDL but implemented with an existing programming language

The mapping (binding) from IDL to the language needs to be specified

Currently the bindings are specified by OMG for

C, C++, Ada, Smalltalk, Cobol, Java, Python, Lisp

Corba IDL Types
Data Types

Basic Types

Type	Description
(unsigned) short	16-bit integer
(unsigned) long	32-bit integer
float	16-bit IEEE float
double	32-bit IEEE float
char	ISO Latin-1 character
boolean	boolean type (TRUE, FALSE)
string	variable length string
octet	8-bit uninterpreted type
enum	enumerated type with named integer values
any	can be any type

Constructed Types

Structure - record with named members

```
struct struct_type_name {
   type1 member_name1;
   type2 member_name2;
};

struct studentRecord {
   stringname;
   float     gradePoint;
   booleanisGraduateStudent;
}
```

Discriminated Union - a type that takes on one of several possible types depending on a discriminator of scalar types

```
union union_type_name switch(discriminator_type) {
   case value1: type1 member_name1;
   case value2: type2 member_name2;
   default : type3 member_name3;
}
```

Array - indexed list of fixed length

```
typedef array_type_name1 member_type1(20);
typedef array_type_name2 member_type2(10)(31);

typedef classList studentRecord(25);
```

Sequence - indexed list of variable length

```
typedef bounded_type_name sequence <memberType, 20>;
typedef unbounded_type_name sequence <memberType2>;
```

Exception

```
exception exception_name{
   type1 member_name1;
   type2 member_name2;
};
```

Attributes and Operations

Actions that can be requested of a Corba object

Attribute - way to specify pair of operations to get/set a value

readonly attributes are get only

```
attribute type message_name;
readonly attribute type message_name;
```

Example -

```
attribute long sum;
```

Translates to (methods in an interface):

```
public void sum(int sum) throws CORBA.SystemException;
public int sum() throws CORBA.SystemException;
```

The name of the method depends on the language

Operation - have name, a return type, a list of parameters, a raises clause, and a context clause

```
interface TheaterBooking {
   enum seating_section {floor, gallery, balcony, noseBleed};
   struct date {
      short day;
      short month;
   };

   exception no_seats {
      sequence <seating_section> seats_still_available_in;
   };

   exception no_performance { };

   typedef short reservation_code;

   reservation_code make_booking(
      in date performance,
      in seating_section position,
      out float price)
   raises (no_performance, no_seats)
   context( ROW_PREF );
};
```

raises - list exceptions

context - a list of string names, which if exist on client end are sent to the server

Parameters types

- in - argument is passed from client to server
- out - argument is returned from server to client
- inout - in and out, can be changed by server

Issue 1

The server and client are in different name spaces (machines)

in parameters must be sent from client to server

If the parameter is an object, it and all its data members must be sent

Issue 2

Java does not directly support out or inout parameters!

Solution is to wrap the parameters in an object

The wrapper objects are passed back and forth

Inheritance

Corba's IDL has multiple inheritance

But the IDL only defines interfaces, so maps to Java interfaces

```
interface TheaterService : TheaterBooking {

   readonly attribute date next_performance;

   short number_free(
      in date performace,
      in seating_section position)
   raises (no_perfromance)
}
```

Name Scopes

Modules define a name scope, like Java's package

```
module Botony {
   interface Leaf {
      readonly attribute float size;
      void grow( in float amoutOfSun);
}

module SearchTree {
   interface Leaf {
      attribute any value;
}

::SearchTree::Leaf
::Botony::Leaf
```

A Simple Example

Using Visigenic's Visibroker

Will implement a simple counter object

It keeps a count, which can be increased and accessed

This example is from Client/Server Programming with Java and CORBA

Step 1) Create and Compile an IDL file

```
// count.idl file

module Counter
{
  interface Count
  { attribute long sum;
    long increment();
  };
};
```

```
Unixprompt-> idl2java -T count.idl -no_comments
Creating: Counter
Creating: Counter/Count.java
Creating: Counter/Count_var.java
Creating: Counter/_st_Count.java
Creating: Counter/_sk_Count.java
Creating: Counter/CountOperations.java
Creating: Counter/_tie_Count.java
Creating: Counter/_example_Count.java
```

What is all this stuff?

Counter/Count.java

A Java interface for the Counter example

```
package Counter;
public interface Count extends CORBA.Object {
  public void sum(int sum) throws CORBA.SystemException;
  public int sum() throws CORBA.SystemException;
  public int increment() throws CORBA.SystemException;
}
```

_st_Count.java, _sk_Count.java

The stub (client) and skeleton (server) code

```
package Counter;
public class _st_Count extends pomoco.CORBA.Stub implements Counter.Count {
  public _st_Count() {
  }
  public String[] _getRepIds() {
    return _repIds;
  }
  static {
    pomoco.CORBA.Global.addMapping(
      "IDL:Counter/Count:1.0",
      "Counter::Count",
      new Counter._st_Count().getClass()
    );
  }
  public static String[] _repIds = {
    "IDL:Counter/Count:1.0"
  };
  public int increment() throws CORBA.SystemException {
    CORBA.IOStream _stream = this._create_request("increment", true);
    _invoke(_stream, true);
    int _result;
    _result = _stream.read_int();
    return _result;
  }
  public void sum(int sum) throws CORBA.SystemException {
    CORBA.IOStream _stream = this._create_request("_set_sum", true);
    _stream.write_int(sum);
    _invoke(_stream, true);
  }
  public int sum() throws CORBA.SystemException {
    CORBA.IOStream _stream = this._create_request("_get_sum", true);
    _invoke(_stream, true);
    int _result;
    _result = _stream.read_int();
    return _result;
  }
}
```

_example_Count.java

An example skeleton for the Count object

```
package Counter;
public class _example_Count extends Counter._sk_Count {
  public _example_Count(java.lang.String name) {
    super(name);
  }
  public _example_Count() {
    super();
  }
  public int increment() throws CORBA.SystemException {
    // implement operation...
  }
  public void sum(int sum) throws CORBA.SystemException {
    // implement attribute writer...
  }
  public int sum() throws CORBA.SystemException {
    // implement attribute reader...
  }
}
```

Step 2) Implement the Counter, Server, and Client

```
package Counter;
// CountImpl.java: The Count Implementation
class CountImpl extends Counter._sk_Count implements Counter.Count
{
  private int sum;

  // Constructor
  CountImpl(String name)
  { super(name);
    System.out.println("Count Object Created");
    sum = 0;
  }
  // get sum
  public  int sum() throws CORBA.SystemException
  { return sum;
  }

  // set sum
  public  void sum(int val) throws CORBA.SystemException
  { sum = val;
  }

  // increment method
  public int increment() throws CORBA.SystemException
  { sum++;
    return sum;
  }
}
```

Implement a Server

```
package Counter;
// CountServer.java: The Count Server main program

class CountServer
{ static public void main(String[] args)
  { try
    { // Initialize the ORB.
      CORBA.ORB orb = CORBA.ORB.init();

      // Initialize the basic object adapter (BOA).
      CORBA.BOA boa = orb.BOA_init();

      // Create the Count object.
      CountImpl count = new CountImpl("My Count");

      // Export to the ORB newly created object.
      boa.obj_is_ready(count);

      // Ready to service requests.
      boa.impl_is_ready();
      }
      catch(CORBA.SystemException e)
      { System.err.println(e);
      }
   }
}
```

Implement a Client

```
package Counter;
// CountClient.java  Static Client, VisiBroker for Java

class CountClient
{ public static void main(String args[])
  { try
    { // Initialize the ORB
     CORBA.ORB orb = CORBA.ORB.init();

      // Bind to the Count Object
     Count counter =Count_var.bind("My Count");

      // Set sum to initial value of 0
     counter.sum((int)0);

     System.out.println("Incrementing");
      for (int i = 0 ; i < 1000 ; i++ )
      { counter.increment();
      }

     System.out.println("Sum = " + counter.sum());
    } catch(CORBA.SystemException e)
    { System.err.println("System Exception");
      System.err.println(e);
    }
  }
}
```

Step 3) Running the System

a) Compile client, server, and Count classes with javac

b) Make sure the ORB is running:

```
UnixPrompt-> osagent &
```

c) Run the server

```
java Counter.CountServer
```

d) Run the client

```
java Counter.CountClient
```

CORBA Services

Life Cycle Service
Operations for creating, copying, moving objects

Persistence Service
A single interface for storing objects in databases and files

Naming Service
Allows objects to be accessed by name

Objects can be bound to existing naming context including ISO X.500, OSF DCE, Sun's NIS+, Novell's NDS and Internet's LDAP

Event Service
Objects can dynamically registered/unregistered interest in events

Concurrency Service
Provides locks for transactions and threads

Transaction Service
Two-phase commit process for transactions

Relationship Service
Create dynamic associations between objects that know nothing about each other

Externalization Service
- Standard way to get data in/out of objects using stream like mechanism

Query Service
- A superset of SQL, based on SQL3 and Object Query Language (OQL)

Licensing Service
- Allows for charging for use of objects

Properties Service
- Associate properties with objects

Time Service
- Synchronization in distributed environment
- Handles time-triggered events

Security Service
- Authentication, access control, confidentiality, etc.

Trader Service
- Yellow pages for objects
- Objects can publicize services, bid for jobs

Collection Service
- Supports common collections

Startup Service
- Enables requests to automatically run when an ORB is started