
CS 696

Introduction to Grid Computing:
Lecture #6

Mary Thomas

San Diego State

Tuesday, 06-Feb-07

Misc:

- HW #2 will be posted Thursday
 - Topic: Developing Web Services using XML-RPC
- To get started:
 - Download from Apache website:
 - <http://ws.apache.org/xmlrpc/>
 - Get examples working - python
- Book: Diving into Python, by Mark Pilgrim
 - Pdf version: <http://diveintopython.org/index.html>

XML (Cont.)

Comments on XML Namespaces

- Namespaces are not located anywhere -- there is no 'pulling' of definitions
 - Not required to be present in document
- Parsing applications use them
 - Runs on server
 - has context: RSS reader will handle a RSS XML file differently than SVG rendering engine would handle it.
 - Uses XPATH to parse document and resolve namespaces, DTD/XSD
 - Uses XSLT's to generate a style
 - Uses Schemas (DTD/XSD) for datatype, etc.: note DTD/XSD can be contained in the docs
 - Schemas contain an element for the namespace that it represents
- Apache Xerces is a parser for Java

Structure of an XML data document

```
<?xml version="1.0" encoding="UTF-8"?>
```

Declares that this doc is XML document

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">  
</xsd:schema
```

Defines any namespaces (may appear also in schema docs

```
<mynode xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"  
  xsi:schemaLocation="/user/mthomas/test.xsd"  
<xsd:include schemaLocation="http://acel.sdsu.edu/myschema.xsd"/>
```

SchemaInstance/SchemaLocation: Physical URI to a schema document for use by parsing engine

```
<invoice id='123'>  
  <item>  
    <sku>100</sku>  
    <price>9.95</price>  
  </item>  
  <item>  
    <sku>100</sku>  
    <price>9.95</price>  
  </item>  
</invoice>
```

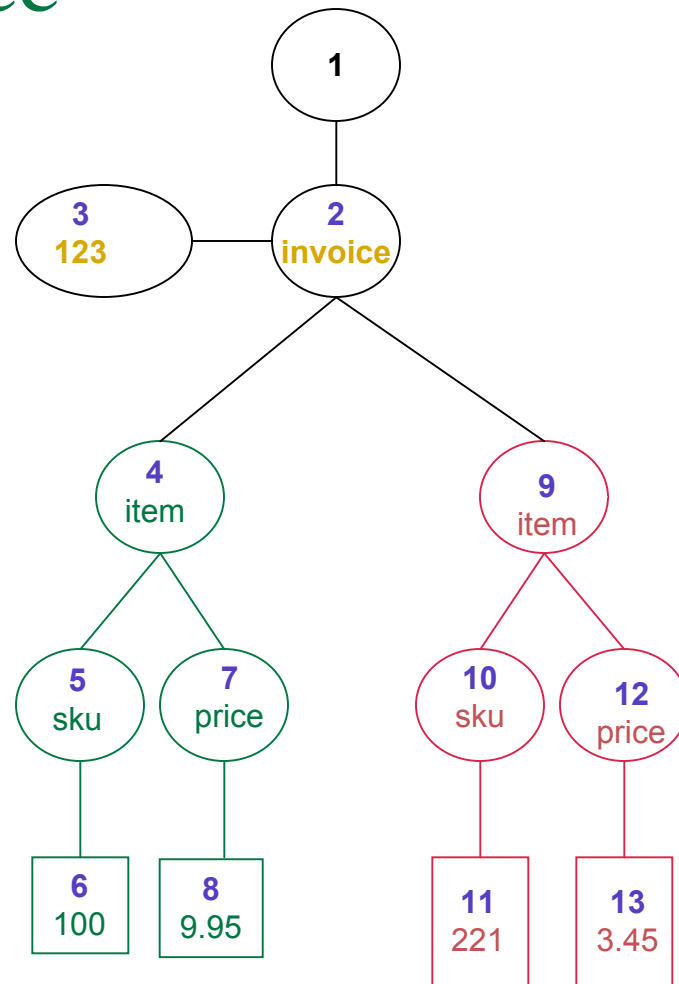
Main body of document

XML Path Language (XPath)

- W3C standard, uses XSLT
- Provides common syntax and semantics for functionality shared between XSL Transformations and XPointer.
- Used to address parts of an XML document.
- Tree-like structure
- Utilizes nodes, operations

XPATH Parsing Tree

```
<?xml version="1.0"
  encoding="UTF-16"?>
<invoice id='123'>
  <item>
    <sku>100</sku>
    <price>9.95</price>
  </item>
  <item>
    <sku>221</sku>
    <price>3.45</price>
  </item>
</invoice>
```



XSLT

- Used by parsing engine to transform XML documents into other text formats or other XML docs if needed.
- Resolves namespaces

```
<?xml version="1.0" encoding="UTF-16"?>
<v1:emp xmlns:v1='urn:employee:v1'>
  <fname>BoB</fname>
  <lname>Smith</lname>
  <age>100</age>
</v1:emp>
</item>
<item>
  <sku>100</sku>
  <price>9.95</price>
</item>
</invoice>
```

Using `telnet` as a Diagnostic Tool

Telnet: Telecommunications Network

- a protocol that provides way for clients to connect to multiuser computers (or servers) on the Internet regardless of location
- text based communication program that allows you to connect to a remote server over a network.
- It is normal use is to login to a server that has shell access
 - But more and more places want SSH
- Telnet runs on Windows, Unix, linux...just about anywhere

Telnet Protocol

- Syntax (for full list type telnet help at a command prompt):

```
telnet <hostname> <port>
```
- Where:
 - <hostname> is name or IP address of remote server
 - <port> is the port number of the service to use for the connection.
 - The default is 23 (TELNET service), 80 (HTTP)
- Both parameters are optional, <port> can only follow <hostname>.
- If <hostname> is specified, telnet will attempt to connect to the remote system <hostname> on port <port> (default telnet port 23 if not specified).
- Running telnet without any parameters will invoke the telnet command mode prompt where you can enter commands to configure it.

Checking a Web Server: HTTP/1.0 or HTTP/1.1

- It is possible to use Telnet to perform some very basic checks to confirm if your web server is responding to requests.
 - Can use GET or POST methods (defined by W3C std)
- A web server typically listens for connections on port 80 (often referred to as HTTP)
- Popular alternative ports may be used:
 - Port 8080 is a common alternative or where a proxy is involved
 - Port 443 is normally used for secure (HTTPS) connections etc.
 - Note: secure connections cannot be tested with Telnet because it has no knowledge of initiating a secure encrypted connection with the web server.

You can use telnet to ‘talk’ to HTTP server - but must use protocol:

```
[gidget:com/ecerami/xmlrpc] mthomas% telnet acel.sdsu.edu 80
Trying 130.191.27.142...
Connected to acel.sdsu.edu.
Escape character is '^]'.
jjj
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
<title>Cannot process request!</title>
</head>
<body>
<h1>Cannot process request!</h1>
<p>The server does not support the action requested by the browser.\</p>
<p>If you think this is a server error, please contact
the <a href="mailto:%5bno%20address%20given%5d">webmaster</a>.</p>
<h2>Error 501</h2>
<address><a href="/">www.acel.sdsu.edu</a><br />
</body>
</html>
Connection closed by foreign host.
```

Telnet to HTTP: GET

```
[gidget:com/ecerami/xmlrpc] mthomas% telnet
localhost 80
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK

...web page contents
```

POST-ing Requests to Server

- Must follow an exact syntax and structure
- W3C: 5.1
 - The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF.
 - Elements are separated by SP characters.
 - No CR or LF is allowed except in the final CRLF sequence.
 - Request-Line = Method SP Request-URI SP HTTP-Version CRLF

The POST Method

- A POST request is different from a GET request in the following ways:
 - A block of data is sent with the request, in the message body. There are usually extra headers to describe this message body, like Content-Type: and Content-Length:.
 - Request URI is not a resource to retrieve; it's usually a program to handle the data you're sending.
 - HTTP response is normally program output, not a static file.

W3C: HTTP POST Request-Line Structure

- The first thing you type. Must be exact or server will close connection
 - Initial method followed by input data (tokens)
- The Request-Line begins with a **method token**, followed by the **Request-URI** and the **protocol version**, and ending with **CRLF**.
- The **elements** are separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.
- Request-Line = Method SP Request-URI SP HTTP-Version CRLF

W3C: HTTP Request Method Tokens

- The Method token indicates the method to be performed on the resource identified by the Request-URI. The method is case-sensitive.

| | | |
|--------|------------------|---------------|
| Method | = "OPTIONS" | ; Section 9.2 |
| | "GET" | ; Section 9.3 |
| | "HEAD" | ; Section 9.4 |
| | "POST" | ; Section 9.5 |
| | "PUT" | ; Section 9.6 |
| | "DELETE" | ; Section 9.7 |
| | "TRACE" | ; Section 9.8 |
| | "CONNECT" | ; Section 9.9 |
| | extension-method | |

extension-method = token

HTTP Request Header Fields

- Way to send some data to server
- The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server.
- These fields act as request modifiers
 - semantics equivalent to the parameters on a programming language method invocation.

HTTP Request Header Fields

```
request-header = Accept           ; Section 14.1
                 | Accept-Charset ; Section 14.2
                 | Accept-Encoding ; Section 14.3
                 | Accept-Language ; Section 14.4
                 | Authorization  ; Section 14.8
                 | Expect         ; Section 14.20
                 | From           ; Section 14.22
                 | Host          ; Section 14.23
                 | If-Match      ; Section 14.24

                 | If-Modified-Since ; Section 14.25
                 | If-None-Match     ; Section 14.26
                 | If-Range         ; Section 14.27
                 | If-Unmodified-Since ; Section 14.28
                 | Max-Forwards     ; Section 14.31
                 | Proxy-Authorization ; Section 14.34
                 | Range            ; Section 14.35
                 | Referer         ; Section 14.36
                 | TE              ; Section 14.39
                 | User-Agent      ; Section 14.43
```

Correct HTTP/1.0 Protocol

```
gidget:com/ecerami/xmlrpc] mthomas% telnet acel.sdsu.edu 80
Trying 130.191.27.142...
Connected to acel.sdsu.edu.
Escape character is '^]'.
GET / HTTP/1.0 [CRLF]
[CRLF]
HTTP/1.1 200 OK
Date: Thu, 17 Mar 2005 17:00:04 GMT
Server: Apache/2.0.50 (Linux/SUSE)
Accept-Ranges: bytes
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="content-type"
content="text/html; charset=ISO-8859-1">
  <title>SDSU Advanced Computing Environments (ACE) Lab</title>
</head>
<body. . . >
This file last modified Wednesday, 23-Feb-2005 09:04:59 PST
</body>
</html>
```

XML- RPC

XML-RPC (Remote Procedure Calling)

- XML-RPC is a protocol that works over the Internet.
- Based on HTTP 1.0 and 1.1
- An XML-RPC message is an HTTP-POST request.
- The body of the request is in XML.
- A procedure executes on the server and the value it returns is also formatted in XML.
- Procedure parameters can be scalars, numbers, strings, dates, etc.
 - can also be complex record and list structures.

XML-RPC (Remote Procedure Calling)

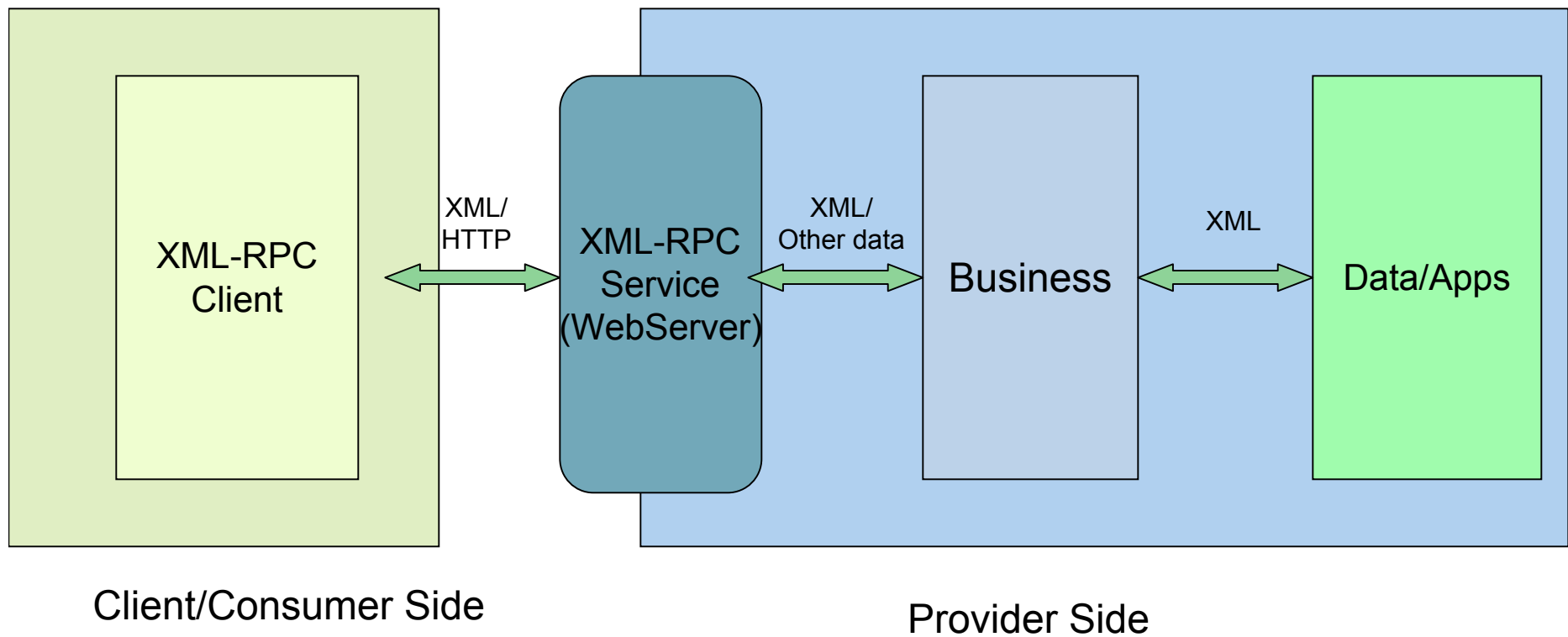
- Example: **HEADER IS RED**, **BODY IS BLUE**

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

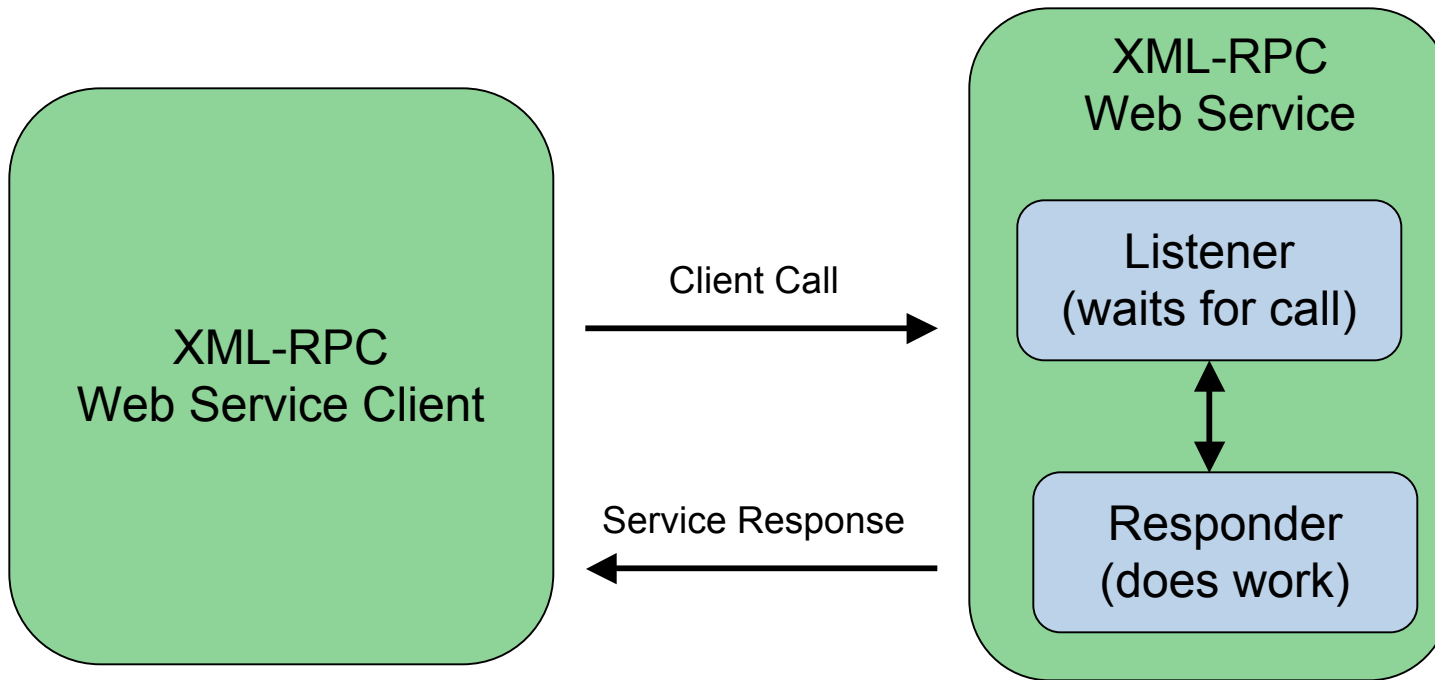
XML-RPC
Specific

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

Web Service Application: XML-RPC View



WSRF Architecture: Single Request, Single Response



Client is free to pursue other activities while service performs request

XML-RPC DataTypes

- int 32 bit integer
- double 64 bit floating point
- boolean 1==true, 0==false
- string restricted to ascii
- dateTime dateTim.iso8601 std
- base64 binary encoding
- Compound data types: arrays and structs

Compound data types: Arrays

```
<value>
  <array>
    <data>
      <value><array><data>
        ...
      </data></array></value>
    </data>
  </array>
</value>
```

Compound data types: Structs

```
<value>
  <struct>
    <member>
      <value>data1</value>
      <value>data2</value>
    </member>
    <member>
      <value><array><data>
        ...
      </data></array></value>
    </member>
  </struct>
</value>
```

XML-RPC Example: adding doubles

■ Three Classes

- ❑ DummyServer: launch server on host:port
 - Registers methods and classes
 - Input: port number
- ❑ DummyClient: sends requests to server and accesses methods registered with server
 - Inputs: port#, two numbers
- ❑ DummyHandler
 - Contains the public methods available on server

XML-RPC Ex.: DummyServer.java

```
package com.ecerami.xmlrpc;
import java.io.IOException;
import org.apache.xmlrpc.WebServer;
import org.apache.xmlrpc.XmlRpc;

public class DummyServer {
    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println(
                "Usage: java DummyServer [port]");
            System.exit(-1);
        }
        try {
            startServer(args);
        } catch (IOException e) {
            System.out.println("Could not start server: " +
                e.getMessage());
        }
    }
}
```

```
public static void startServer(String[] args) throws IOException {
    // Start the server, using built-in version
    System.out.println("Attempting to start XML-RPC Server...");
    try {
        WebServer server = new WebServer(Integer.parseInt(args[0]));
        server.start();
        System.out.println("Started successfully.");

        // Register our handler class as area
        server.addHandler("dummy", new DummyHandler());
        System.out.println(
            "Registered DummyHandler class to dummy.");
        server.addHandler("serv", new DummyServer());
        System.out.println(
            "Registered DummyServer class to server");

        System.out.println("Now accepting requests. (Halt program to stop.);");
    } catch (Exception webServerStartError) {
        System.err.println("JavaServer " + webServerStartError.toString());
    }
}
```

This will register a method class with the server. Its public methods will be available to clients. In this case, the methods are in another class

XML-RPC Ex: DummyHandler.java

```
package com.ecerami.xmlrpc;
import java.io.IOException;
import java.util.Vector;

public class DummyHandler {
    public String dummyTester(String str) {
        return str;
    }

    public Double addDbl(double x, double y) {
        return new Double ( x + y );
    }

    public static void main(String args[]) { //this is just for testing my service providers
        if (args.length < 2) {
            System.out.println(
                "Usage: java DummyHandler [num1 num2]");
            System.exit(-1);
        }
        Double a = new Double ( args[0] );
        Double b = new Double ( args[1] );
        DummyHandler dum = new DummyHandler();
        Double res = dum.addDbl(a.doubleValue(), b.doubleValue() );
        System.out.println("the answer is: " + dum.addDbl(a.doubleValue(),b.doubleValue()) );
        System.out.println( dum.dummyTester("dummyTester working" ) );
    }
}
```

XML-RPC Ex.: DummyClient.java

```
package com.ecerami.xmlrpc;
import java.io.IOException;
import java.util.Vector;
import org.apache.xmlrpc.XmlRpc;
import org.apache.xmlrpc.XmlRpcClient;
import org.apache.xmlrpc.XmlRpcException;

public class DummyClient {
    public static void main(String args[]) {
        if (args.length < 3) {
            System.out.println(
                "Usage: java DummyClient [port#] [num1] [num2]");
            System.exit(-1);
        }
        DummyClient client = new DummyClient();

        try {
            String str = client.dumbClient(args);
            // Report the results
            System.out.println("returned from dummy");
            System.out.println("The answer is: " + str);
        } catch (IOException e) {
            System.out.println("IO Exception: " + e.getMessage());
        } catch (XmlRpcException e) {
            System.out.println("Exception within XML-RPC: " + e.getMessage());
        }
    }
}
```

XML-RPC Ex.: DummyClient.java

```
public String dumbClient (String args[] )
    throws IOException, XmlRpcException {
    //grab port number and build host string
    String hostStr = "http://localhost:" + Integer.parseInt(args[0]) + "/";
    System.out.println("Server host Port# is: " + hostStr);
    // Create client, identify server
    XmlRpcClient client =
        new XmlRpcClient(hostStr);

    System.out.println("created XmlRpcClient to: " + hostStr);
    Vector params = new Vector();
    Double a = new Double ( args[1] );
    Double b = new Double ( args[2] );
    params.addElement(new Double(a.doubleValue()));
    params.addElement(new Double(b.doubleValue()));

    // Issue a request
    Object result = client.execute("dummy.addDbl", params);
    System.out.println("client execute complete");

    String resultStr = result.toString();
    return resultStr;
}
}
```

Connection to
Server on Port

All calls must pass
Vector data

Connect to method
"addDble"

XML-RPC Issues

- HTTP Connection: no persistence/stateless
- All methods and classes must be known -
 - Not very flexible
- Designed for small, short messages.

Telnet to XML-RPC Service

```
mthomas% telnet localhost 8888
Trying ::1...
Connected to localhost.
Escape character is '^]'.
POST /RPC HTTP/1.0
User-Agent: DummyServer
Host: localhost:8888
Content-Type: text/xml
Content-length: 500
[CRLF]
JUNKINPUTHERE
```

```
Running DummyServer on Port #8888
Attempting to start XML-RPC Server...
Started successfully.
Registered DummyHandler class to dummy.
Registered DummyServer class to serv.
Now accepting requests. (Halt program
to stop.)
```

```
Fatal error parsing XML:
org.xml.sax.SAXParseException:
expected Element
org.apache.xmlrpc.ParseFailed:
org.xml.sax.SAXParseException:
expected Element
```

Telnet to XML-RPC Service

```
POST /RPC HTTP/1.0
User-Agent: DummyServer
Host: localhost:8888
Content-Type: text/xml
Content-length: 500
[CRLF]
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>dummy.addDb1</methodName>
  <params>
    <param>
      <value><double>5.0</double></value>
    </param>
    <param>
      <value><double>3.0</double></value>
    </param>
  </params>
</methodCall>
```

```
<?xml version="1.0"
  encoding="ISO-8859-1"?>
<methodResponse>
  <params>
    <param>
      <value><double>8.0</double>
    >
      </value>
    </param>
  </params>
</methodResponse>
Connection closed by foreign
host.
```

Running DummyServer

```
[gidget:com/ecerami/xmlrpc] mthomas% cat rundums
#!/bin/tcsh
#
# script to start the server
# avoid useless retyping of command line args
#
setenv JAVA /usr/bin/java
setenv PKG_LOC com.ecerami.xmlrpc
setenv PORT $argv[1]
echo "Running DummyServer on Port #${PORT}"
$JAVA $PKG_LOC.DummyServer $PORT
```

Scripts can be used to help avoid repetitive typing and speed up the development and testing process.

I used these for my examples.

```
gidget:com/ecerami/xmlrpc] mthomas% ./rundums 8888
Running DummyServer on Port #8888
Attempting to start XML-RPC Server...
Started successfully.
Registered DummyHandler class to dummy.
Registered DummyServer class to serv.
Now accepting requests. (Halt program to stop.)
```

Running DummyClient

```
[gidget:com/ecerami/xmlrpc] mthomas% cat rundumc
#!/bin/tcsh
#
# script to start the client
#
setenv JAVA /usr/bin/java
setenv PACKAGE_LOC com.ecerami.xmlrpc
setenv PORT $argv[1]
setenv NUM1 $argv[2]
setenv NUM2 $argv[3]
echo "Running DummyClient on Port #${PORT} for values $NUM1 + $NUM2"
$JAVA $PACKAGE_LOC.DummyClient $PORT $NUM1 $NUM2
```

```
gidget: com/ecerami/xmlrpc] mthomas% ./rundumc 8888 14 12.2

Running DummyClient on Port #8888 for values 14 + 12.2
Server host Port# is: http://localhost:8888/
created XmlRpcClient to: http://localhost:8888/
client execute complete
returned from dummy
The answer is: 26.2
```

Installing XML-RPC-hints

- You can mostly follow the README.txt file
- However, an XML doc is missing.
 - I made a 'best guess'
 - Contents on next slide
- To monitor your services:
 - `telnet` to 'ping'
 - `netstat -ln` (listener ports)
 - `lsof -P` (show port numbers)

XML-RPC:

```
[gidget:apps/xmlrpc/xmlrpc-1.2-b1] mthomas% cat build.properties
```

```
<!------- XML-RPC build.properties ----->
```

```
<!--
```

The file was missing so I made some guesses at what the contents should be. You may need to make changes depending on your OS

```
----->
```

```
<!-------          Mary Thomas      ---->
```

```
<!-------          Mac OS X         ---->
```

```
<!-------          14-Mar-05        ---->
```

```
src.dir=/Users/mthomas/local/apps/xmlrpc/xmlrpc-1.2-b1/src
```

```
final.name=xmlrpc-1.2-b1
```

```
build.dir=build.dir
```

```
build.dest=package
```

```
dest.dir=jarfiles
```

```
test.reportsDirectory=reports
```

```
javadoc.destdir=javadoc
```

```
build.test.dest=test
```

```
jsse.jar=/System/Library/Frameworks/JavaVM.framework/Versions/1.4.2/Classes  
/jsse.jar
```

Useful Links

- Good Books:

- D. Chappel, T. Jewell, “Java Web Services,” Orielly, 2002
- E. Cerami, “Web Services Essentials,” Orielly, 2002
- Oellermann, “Architecting Web Services,” AI Press, 2001

- **ANT: Manual**

- <http://ant.apache.org/manual/index.html>

- Web Services Arch:

- <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>

- XML 1.0 Standard:

- <http://www.w3.org/TR/2000/REC-xml-20001006>

- XML Schema:

- <http://www.w3.org/TR/xmlschema-0/>