Web Service XML RPC

AMRO NAJJAR
TELECOM SAINT-ETIENNE
MASTER WEB-INTELLIGENCE

References

- **Web Services: Principles and Technology,** Michael Papazoglou
- **Java Web Services: Up and Running**, Martin Kalin
- **Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTFUL web services,** Robert Daigneau
- **Web Services Essentials (O'Reilly XML), Ethan Cerami**

XML-RPC

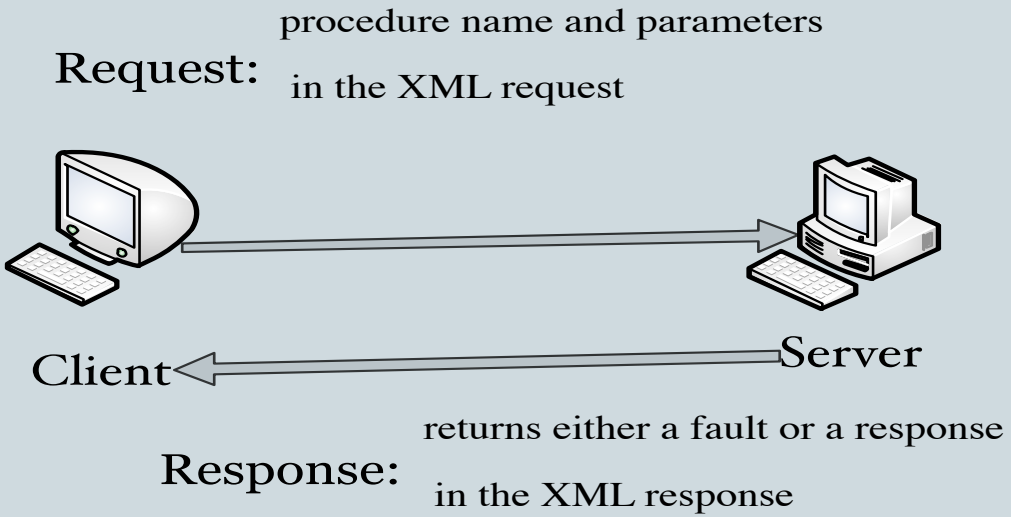
- provides an XML- and HTTP-based mechanism for making method or function calls across a network
- Very simple & useful
- This slides present:
  - An introduction to the main concepts and history of XML-RPC
  - XML-RPC usage scenarios

XML-RPC Intro

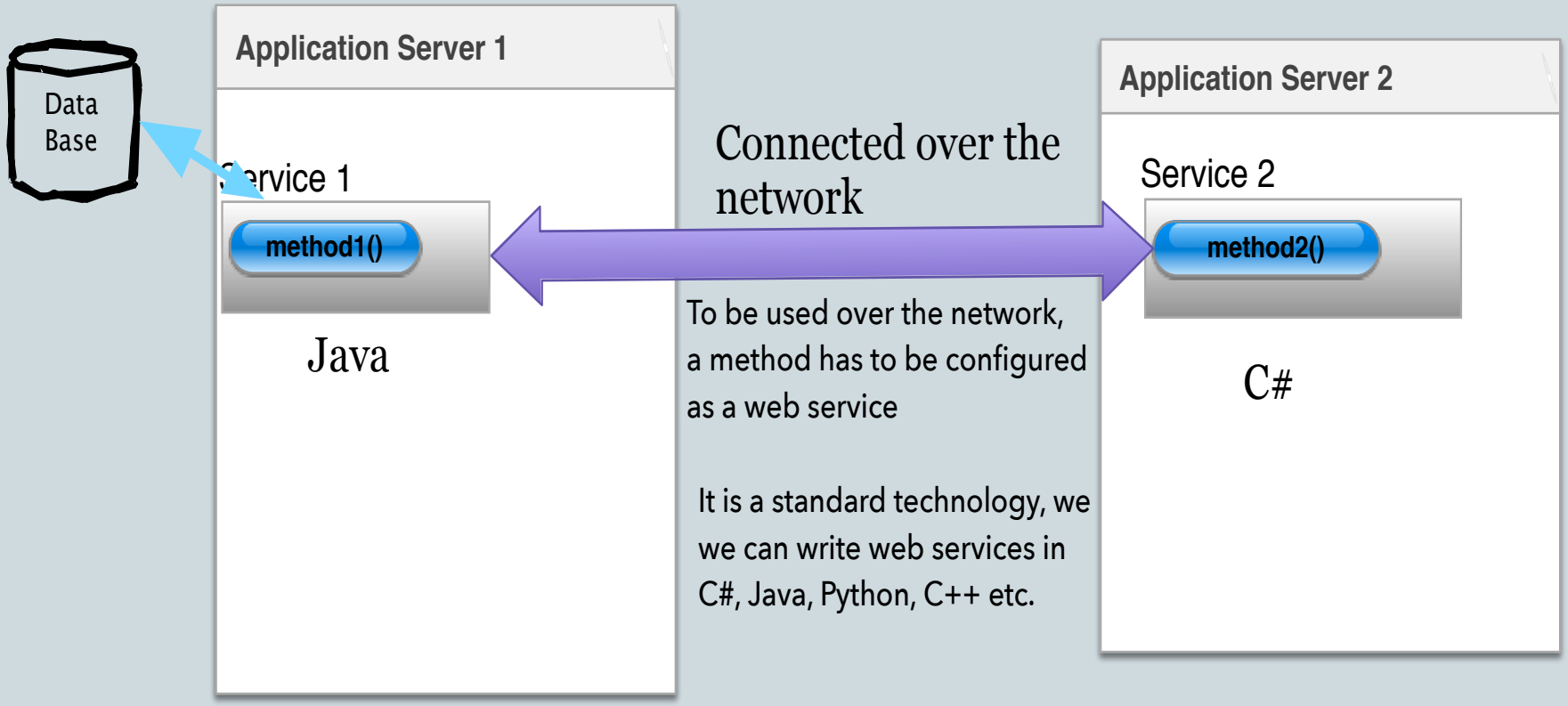
- Developped by Dave Winer of Userland in the late 1990
- Is a very lightweight RPC system
  - Support for elementary data types (basically, the built-in C types together with a boolean and a datetime type)
  - Few simple commands
  - Follows the request/response pattern
- Two key features
  - use of XML marshaling/unmarshaling to achieve language neutrality
  - Reliance on HTTP for transport

XML-RPC Overview

- Permits programs to make function or procedure calls across a network
- Use HTTP Protocol to pass information from a client computer to a server
  - Nature of requests and responses with a small XML vocabulary



XML RPC


Data Base
Application Server 1
Service 1
method1()
Java
Connected over the network
To be used over the network, a method has to be configured as a web service
It is a standard technology, we we can write web services in C#, Java, Python, C++ etc.
Application Server 2
Service 2
method2()
C#

XML RPC : Parameters

- XML-RPC parameters are a simple list of types and content
  - structs and arrays are the most complex types available.
- XML-RPC has no notion of objects
- No mechanism for including information that uses other XML vocabularies

Remark
This is a limitaton to be addressed by the SOAP and co

XML RPC Use Scenarios

- **Glue Code with XML-RPC**
- **Publishing Services with XML-RPC**

Glue Code with XML-RPC

- Developpers need to address the problem of integrating distributed systems
  - Unix systems need to speak with Windows etc.
- Instead of creating custom systems, developers can use XML-RPC to connect programs running on different systems and environments
- Use existing APIs and add connections to those APIs as necessary
- No need for a clear distinction between Client & Server

Publishing Services with XML-RPC

- XML-RPC can be used to publish information to the world
- It allows information recipients to be any kind of client that understands the XML-RPC interface

XML-RPC Technical Overview

- XML-RPC consists of three relatively small parts
  - *XML-RPC data model*
    - A set of types for use in passing parameters, return values, and faults (error messages)
  - *XML-RPC request structures*
    - An HTTP POST request containing method and parameter information
  - *XML-RPC response structures*
    - An HTTP response that contains return values or fault information

XML RPC is the combination of the three:

The data structures are used by both the request and response structures

XML-RPC Data Model: Basic Types

- six basic data types and two compound data types
  - this is a restricted set of types
  - Attempts to hit the lowest common denominator for many kinds of program-to- program communications
- Basic types are represented by simple XML elements
  - <string>Hello World!</string>
  - Basic types are always enclosed in value elements
    - <value><int>7</int></value>

XML-RPC Data Model: Basic Types Overview

Table 2-1. Basic data types in XML-RPC

Type	Value	Examples
`int` or `i4`	32-bit integers between -2,147,483,648 and 2,147,483,647.	`<int>27</int>` `<i4>27</i4>`
`double`	64-bit floating-point numbers	`<double>27.31415</double>` `<double>-1.1465</double>`
`Boolean`	true (1) or false (0)	`<boolean>1</boolean>` `<boolean>0</boolean>`
`string`	ASCII text, though many implementations support Unicode	`<string>Hello</string>` `<string>bonkers! @</string>`

`dateTime.iso8601`	Dates in ISO8601 format: *CCYYMMDD*T*HH*:*MM*:*SS*	`<dateTime.iso8601>20021125T02:20:04`
		`</dateTime.iso8601>`
		`<dateTime.iso8601>20020104T17:27:30`
		`</dateTime.iso8601>`

`base64`	Binary information encoded as Base 64, as defined in RFC 2045	`<base64>SGVsbG8sIFdvcmxkIQ==` `</base64>`

XML-RPC Data Model: Arrays

- Array Example:

```
<value>
  <array>
    <data>
      <value><int>7</int></value>
      <value><int>1247</int></value>
      <value><int>-91</int></value>
      <value><int>42</int></value>
    </data>
  </array>
</value>
```

Remark

This array is composed of only one type (integer)

XML-RPC Data Model: Arrays & Struct

- Array Example: mixed array

```
<value>
  <array>
    <data>
      <value><boolean>1</boolean></value>
      <value><string>Chaotic collection, eh?</string></value>
      <value><int>-91</int></value>
      <value><double>42.14159265</double></value>
    </data>
  </array>
</value>
```

- Structs:
  - contain unordered content, identified by name (name is string)
  - Each struct element contains a list of member elements
  - Member elements each contain one name element and one value element

XML-RPC Data Model:Struct Example

- A Struct Example:

```
<value>
  <struct>
    <member>
      <name>givenName</name>
      <value><string>Joseph</string></value>
    </member>
    <member>
      <name>familyName</name>
      <value><string>DiNardo</string></value>
    </member>
    <member>
      <name>age</name>
      <value><int>27</int></value>
    </member>
  </struct>
</value>
```

  - Structs can contain other structs, or even arrays
  - Arrays can also contain structs

XML-RPC Request Structure

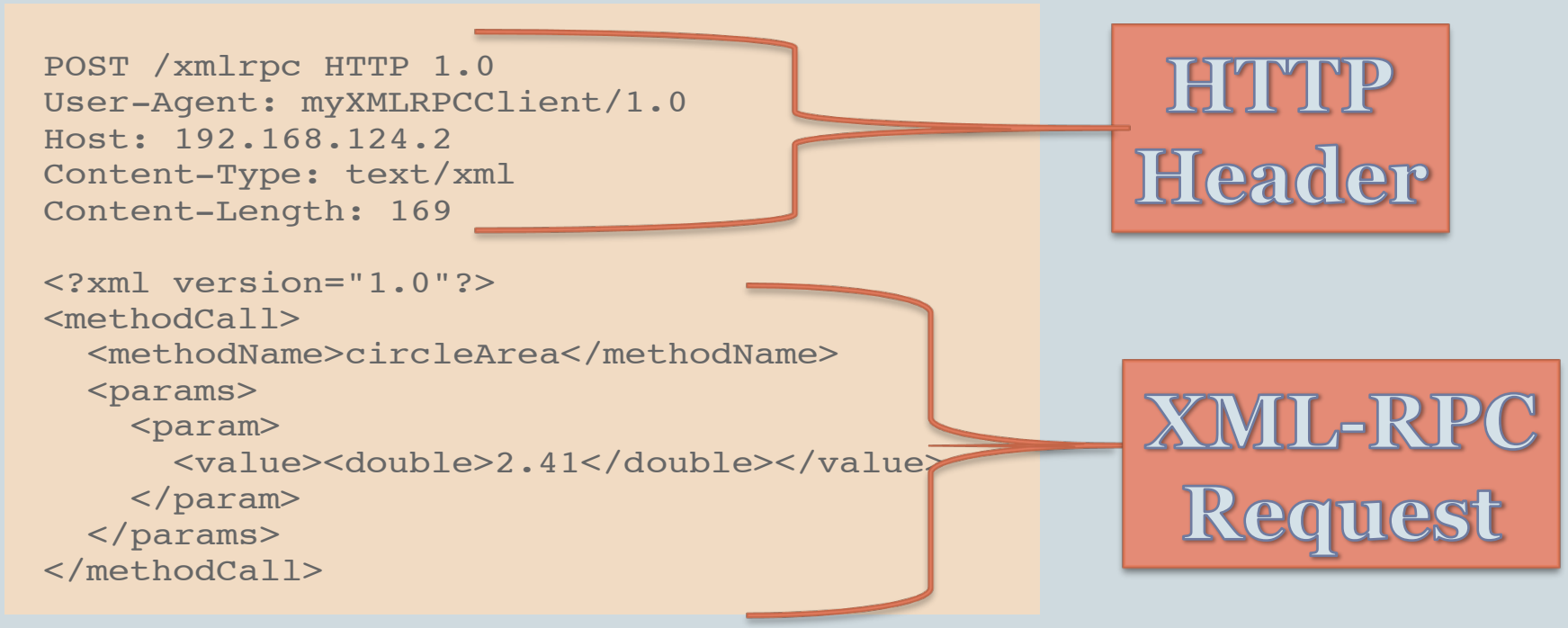
- Requests are a combination of XML content and HTTP headers.
- The XML content
  - uses the data typing structure to pass parameters
  - Contains information identifying the procedure being called
- HTTP headers provide a wrapper for passing the request over the Web
- A request contains a single XML document,
  - whose root element is a methodCall
  - Each methodCall element contains a methodName element and a params element

XML-RPC Request Example

- CircleArea , takes a Double parameter (for the radius), the XML-RPC request would look like

```
POST /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 192.168.124.2
Content-Type: text/xml
Content-Length: 169

<?xml version="1.0"?>
<methodCall>
  <methodName>circleArea</methodName>
  <params>
    <param>
      <value><double>2.41</double></value>
    </param>
  </params>
</methodCall>
```



XML-RPC Response Structure

- much like requests, but we distinguish two possible cases:
  - If the response is successful :
    - the procedure was found, executed correctly, and returned results
  - If there was a problem in processing the XML-RPC request
    - The result will containt a fault message

XML-RPC Response : Success

- The response will look like the request
  - The methodCall element is replaced by a methodResponse element
  - There is no methodName element

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><double>18.24668429131</double></value>
    </param>
  </params>
</methodResponse>
```

Remark

An XML-RPC response can only contain one parameter (this parameter can be an array or struct)

XML-RPC Response :Failure

- The methodResponse element will contain a fault element instead of a params element
- The fault element, like the params element, has only a single value
- that value indicates that something went wrong
- Fault example:

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value><string>No such method!</string></value>
  </fault>
</methodResponse>
```

XML RPC : Pros & Cons

- limitations
  - In the world of OOP, XML RPC seems too simple
  - Complex types are not represented
    - You can, in some cases, use these complex types to represent object structures, but at some point you may find it easier to use SOAP for that kind of complex transfer.
  - Does not exploit HTTP well
    - All XML-RPC responses use the 200 OK response code, even if a fault is contained in the message
  - XML-RPC won't do anything to guarantee that arrays have a consistent number or type of values. You'll need to make sure that you write code that consistently generates the right number and type of output values if consistency is necessary for your application.

XML-RPC vs SOAP

XML RPC

- Very simple and limited (designed to be efficient)
- Short specification (originally 6 pages)
- Order of passed parameters is relevant
- More Python Support

SOAP

- Has a complete relevant ecosystem
- Long specification (about 50 pages)
- The order of passed parameters is irrelevant
- More powerful in general

Optional: XML RPC Cod Example

- In the circle example, we need the following simple method:

```
package com.ecerami.xmlrpc;

public class AreaHandler {

     public double circleArea(double radius) {
          double value=(radius*radius*Math.PI);
          return value;
     }
}
```

Java XML Server Example


```
package com.ecerami.xmlrpc;

import java.io.IOException;
import org.apache.xmlrpc.WebServer;
import org.apache.xmlrpc.XmlRpc;

public class AreaServer {

     public static void main(String[] args) {


    if (args.length < 1) {
       System.out.println("Usage: java AreaServer [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...");
    WebServer server = new WebServer(Integer.parseInt(args[0]));

    System.out.println("Started successfully.");

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

    System.out.println("Now accepting requests. (Halt program to stop.)");

  }
}
```

Needed Libraries

Main (): verifying If the input is pertinent If yes: start the server

The method that launches the server the argument is the port

Java XML Server Example


```
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 AreaClient {

    public static void main(String args[]) {
        if (args.length < 1) {


            System.out.println(
                "Usage: java AreaClient [radius]");
            System.exit(-1);
        }
        AreaClient client = new AreaClient(  );
        double radius = Double.parseDouble(args[0]);

        try {
            double area = client.areaCircle(radius);
            // Report the results
            System.out.println("The area of the circle would be: " + area

        } catch (IOException e) {
            System.out.println("IO Exception: " + e.getMessage(  ));
        } catch (XmlRpcException e) {
            System.out.println("Exception within XML-RPC: " + e.getMessage(  ));
        }
    }
```

Needed Libraries

Verify the Input

Create the client method : *areaCircle()*

The method : addCircle()


```
public double areaCircle (double radius)
  throws IOException, XmlRpcException {

        // Create the client, identifying the server
        XmlRpcClient client =
            new XmlRpcClient("http://localhost:8899/");

        // Create the request parameters using user input
        Vector params = new Vector(  );
        params.addElement(new Double (radius));

        // Issue a request
        Object result = client.execute("area.circleArea", params);
        String resultStr = result.toString(  );
        double area = Double.parseDouble(resultStr);
        return area;
}
```

The line above is important

Create the client& Identify the server

Create the Parameters using The user's input

Issue a request with the Parameters ang retrieve the results

The generated XML request


```
POST / HTTP/1.1
Content-Length: 175
Content-Type: text/xml
User-Agent: Java1.3.0
Host: localhost:8899
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall><methodName>area.circleArea</methodName>
<params>
<param><value><double>3.0</double></value></param>
</params>
</methodCall>
```

And the server responds with a methodResponse

XML RPC is technology-independent

We can create another client in C++ and another in Perl etc.