Using XML for Distributed Computing

XML-RPC and SOAP

Overview

- Motivation
- XML-RPC
- SOAP
- Comparison and analysis
- Conclusions

Motivation

- These protocols have been developed to provide a platform and language independent method of passing information between machines.
- They are not at this point a replacement for Java-RMI or CORBA, but they do provide simple standardized mechanisms for moving structured information.

XML-RPC (Remote Procedure Calls)

- This specification was created by UserLand Software Inc. to provide a simple way for processes on distant machines to exchange information.
- It specifies a common format for packaging information as well as submitting requests and returning the replies.

Request Example

```
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>
```

<methodCall>

- The XML data for an XML-RPC call must be contained in a single <methodCall> structure.
- It must contain a <methodName> sub-item that specifies the method to be called. It is up to the server to decide what to do with it.
- If parameters are expected a <params> sub-item must be given.

Data Tags

- The <params> element can have any number of <param> subelements each of which must contain a value.
- Scalar value tags
  - <i4> or <int>, <boolean>, <string>, <double>, <dateTime.iso8601>, <base64>

<struct>

- Structs contain members and can be recursive.

```
<struct>
   <member>
      <name>lowerBound</name>
      <value><i4>18</i4></value>
   </member>
   <member>
      <name>upperBound</name>
      <value><i4>139</i4></value>
   </member>
</struct>
```

<array>

- These contain a single <data> element and can be recursive.

```
<array>
   <data>
      <value><i4>12</i4></value>
      <value><string>Egypt</string></value>
      <value><boolean>0</boolean></value>
      <value><i4>-31</i4></value>
   </data>
</array>
```

Response Example

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>South Dakota</string></value>
      </param>
   </params>
</methodResponse>
```

Response Syntax

- The <methodResponse> tag contains one <params> which contains one <param> which contains a single <value>.
- It can also contain a fault structure.

Faults

- XML-RPC defines one structure which is the fault return structure.
- A <fault> contains one value which is a <struct> which has a faultCode that is an int and a faultString which is a string.

Notes on XML-RPC

- XML-RPC is simple in the extreme. The specification takes literally four pages with the examples.
- This might be a little too simple. The simplicity definitely restricts the applications that this can be used for.

SOAP (Simple Object Access Protocol)

- SOAP is actually intended to just be a protocol for exchange of information which gives it a somewhat more broad usage than simply RPC.
- It certainly addresses the argument that XML-RPC was too simple. I'm just not convinced that it is gaining much from the complication.

SOAP Request Example

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
   "http://schemas.xmlsoap.org/soap/envelope/”
  SOAP-ENV:encodingStyle=
   "http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <m:GetLastTradePrice xmlns:m="Some-URI">
         <symbol>DIS</symbol>
      </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Response Example

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
   "http://schemas.xmlsoap.org/soap/envelope/”
  SOAP-ENV:encodingStyle=
   "http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
      <m:GetLastTradePriceResponse
        xmlns:m="Some-URI">
         <Price>34.5</Price>
      </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

General Structure

- Envelope - This is the top level of the SOAP message and must be present.
- Header - This is optional but if it occurs it must be the first child of the envelope.
- Body - This must be present either as the first child or following the header. It is where the data of the message is stored.

Envelope

- It must be associated with the namespace “http://schemas.xmlsoap.org/soap/envelope”.
- It can have an encdongStyle attribute that contains a URI describing how the data should be serialized.

Header

- The actor attribute defines “who” needs to deal with a header element.
  - When an actor is specified the proper layer deals with that header element and doesn’t pass it on.
  - Default is the final recipient of the message.
- The mustUnderstand attribute is a boolean.

Body

- The data carried in the body is like data in the Header that is intended for the final destination with mustUnderstand="1".
- Body entries are identified by the namespace URI and the local name.
- The immediate children may be namespace qualified.

Faults

- As with XML-RPC, SOAP defines only one body entry which is a fault.
- SOAP Faults have four subelements: faultcode, faultstring, faultactor, and detail.
- The SOAP faultcodes are also somewhat more robust using strings that use the “dot” notation to refine the nature of the error like “Client.Authentication”.

Types

- The typing in SOAP is largely based on the XML Schema Parts1 and 2 (Structures and Datatypes)
- The typing goes a bit beyond this to model the types found in programming languages.
- The payload of a SOAP message is data only. It may not contain processing instructions.

Enumerations

- One way in which SOAP extends types is by adding enumerated values.

```
<element name="EyeColor" type="tns:EyeColor"/>
<simpleType name="EyeColor" base="xsd:string">
   <enumeration value="Green"/>
   <enumeration value="Blue"/>
   <enumeration value="Brown"/>
</simpleType>
```

Single-Reference vs. Multi-Reference

- SOAP allows values to be referenced multiple times using ids or hrefs.
- When a piece of data is single referenced the specification states that it should be defined in a way which implies this.
- SOAP allows optimization of some multi-reference types and when possible treating multi-reference data as single-reference data.

Structs

- SOAP allows for complex types where the elements have accessor names.
- If the accessor names are unique it is called a structure.
- The definition of a struct is accomplished by the XML Schema complexType. As such they can be recursive.

Arrays

- A complex type where ordinal position is the only way to distinguish elements is an array.
- All arrays have the type SOAP-ENC:Array or some subtype of it.
- Each element can have a different type by defining an xsi:type and arrays can be recursive or multidimensional.

Sparse and Partially Transmitted Arrays

- By supplying a SOAP-ENC:offset arrays can be only partially specified (ex. an array with 5 elements but only [2] and [3] are specified).
- Sparse matrices can be made using SOAP-ENC:position.
- Doesn’t say what the unspecified elements take as a value.

Polymorphism

- The SOAP spec acknowledges that many programming languages include the concept of polymorphism but doesn’t do much beyond that.
- It requires that a type be specified for parametric values instead of just allowing the default.

RPC with SOAP

- The SOAP specification is a bit more strict about the manner of passing information to a method in requiring the passed data must match the method signature.
- However, it does not require a different syntax for requests and responses.

Notes on SOAP

- The SOAP specification is in general much more specific and detailed than the XML-RPC specification but they needed 29 pages to do it.
- However, it doesn’t even attempt to address what I saw as many of the limitations of the XML-RPC implementation.

Analysis

- Order of elements is structures in not considered in XML-RPC.
- Restrictions
  - Neither allows distributed garbage collection, remote references, or activation.
- Since SOAP can’t include functionality it seems it should be SSAP instead.

Conclusions

- These two specification definitely have their applications though I'm not convinced that either is truly an acceptable way of doing RPC.
- They work well for fairly simple data transmission and for that reason I prefer the simpler of the two. It's not clear to me how many of the restrictions in SOAP can be enforced or used to help programmers.

Specs

- XML-RPC spec is at
  - http://www.xml-rcp.com/spec
- SOAP spec is at
  - http://msdn.microsoft.com/workshop/xml/general/soapspec.asp