

UNDERSTANDING THE SOAP PROTOCOL AND THE METHODS OF TRANSFERRING BINARY DATA

Andy TAN

Abstract

The scope of this research aims to give an overview of the SOAP protocol, such as its origins, the protocol structure and the way data is passed between SOAP nodes. However, this paper would also briefly touch on the XML-RPC protocol so that it may be compared with SOAP side-by-side. Another major focus of this research is the methods of transferring binary files using SOAP. Because SOAP is based on XML, a non-binary language, there are many issues involved when this protocol is required to send and receive binary files such as bitmap images and executables. This paper would describe the current SOAP implementations used to solve these issues.

Key Words

SOAP, XML-RPC, Web Services, Binary transfer, XML

1. Introduction

Over the recent years, the introduction of XML has revolutionized the way data is transferred between systems. XML, similar to the popular HTML of common web pages, is a meta-language made up of non-binary ASCII text. Because of the ability of XML documents to “describe” its own data, it allows the great flexibility of storing many different types of data in a structured manner. Moreover, since XML is a platform and application independent language, it is no surprise that it is quickly becoming a standard in data-representation and data-interchange today. [1]

The appearance of Web services technology has further proven the powerful ability of XML to exchange information between systems that are potentially different from each other. A web service, defined by the W3C (WWW Consortium) as “a software system identified by a URI, whose public interfaces and bindings are defined and described using XML” [2], utilizes XML as the base architectural layer for data transfer. With Web services, applications written in different languages and hosted on different platforms are able to call each other’s functions and utilize the returned data for it’s own procedures. Although RPC (Remote Procedure Calls) is not a novelty to the programming world, Web services has extended this idea and consequently created an exciting new model of distributed computing utilizing existing web technology.

One may argue that prior to the emergence of Web services there already exist technologies that facilitate RPC between differing systems. Middleware technologies such as CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model) and RMI (Remote Method Invocation), all promote simplified means of distributed computing. Although middleware has been successfully implemented on private networks such as LANs and Intranets, there are disadvantages to middleware in which many believe to have hindered its successful implementation on the Internet at large. One of these disadvantages includes the fact that middleware uses its own protocols and most firewalls are configured to disallow non-HTTP traffic. Another disadvantage is the difficulty, and therefore cost, of deploying a middleware application. Web services provide the right solution to these problems since application data is transferred via ordinary HTTP clients and servers. Through it’s simplicity, the cost of deployment is also minimized. [1]

Since the Web services model displays many advantages to the traditional middleware technologies, implementations were thus required and standardized. Two main Web services communication frameworks have emerged and they are XML-RPC and SOAP (Simple Object Access Protocol). Although both protocols are able to perform XML-based RPCs, further explanation later in this paper would describe the major differences between the two and why SOAP is the protocol of choice for most Web service implementations.

2. A Brief History of SOAP

SOAP's origin dates back to early 1998 when Dave Winer of UserLand, Don Box of DevelopMentor and Bob Atkinson and Mohsen Al-Ghosein of Microsoft came together to design a protocol that would allow RPC to be sent through the Internet over HTTP via XML [3]. At that time, what the developers privately refer as SOAP resembled more like the XML-RPC of today. However, further development of SOAP then became caught up with internal disputes within Microsoft. Winer soon became impatient and branched out of the current project with his own specifications of the XML-RPC protocol [5], a subset derived from SOAP. However, Winer did continue to contribute his input into the group and subsequently they came up with the finalized SOAP 1.0 specifications [3]. With the additional help of IBM in 2000, SOAP was improved and then version 1.1 was released in April 2000 [4].

The next major step forward for this project was when SOAP 1.1 was submitted and accepted by the W3C as a project of the XML Protocol Working Group [4]. At the time of writing, SOAP is not yet an official W3C Recommendation. Moreover, there are currently two versions of SOAP coexisting. They are:

- SOAP 1.1 – specifications of these W3C Notes can be found at:
 - <http://www.w3.org/TR/SOAP>
 - <http://www.w3.org/TR/SOAP-attachments>
- SOAP 1.2 – this has reached W3C Candidate Recommendation stage and specifications can be found at:
 - <http://www.w3.org/TR/soap12-part1/>
 - <http://www.w3.org/TR/soap12-part2/>

Although SOAP 1.2 is the newer version, most SOAP implementations at the time of writing are still adopting version 1.1 specifications as their framework. We may have to wait until SOAP 1.2 becomes a full W3C recommendation before we see more version 1.2 implementations on the market.

More about the differences between the two versions of SOAP would be discussed in the section 2.2 of this paper detailing the structure of the protocol.

2.1 Comparison with XML-RPC

Before we analyze the make-up of SOAP messages, it would be helpful to understand its cousin, XML-RPC. Figure 1 is an example of an XML-RPC request message to call a function on a remote server over HTTP [6][9].

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

<?xml version="1.0"?>
<methodCall>
  <methodName>NumberToText</methodName>
  <params>
    <param>
      <value><int>41</int></value>
    </param>
    <param>
      <value><string>English</string></value>
    </param>
  </params>
</methodCall>
```

Figure 1: XML-RPC Request Call

The structure of XML-RPC requests, compared to SOAP requests, as we will discover later, is straightforward and easy to understand. The `<methodName>` tag defines the name of the method on the remote server application and the tags within the

<params> tag define the parameters of the method. Notice that a tag also represents the type definition of the value, which in this case, the <int> tag represents an integer type. [6]

The following is a response generated by the server back to the client.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Wed, 30 Apr 2003 18:00:00 GMT
Server: Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
<string>forty-one</string></value>
      </param>
    </params>
  </methodResponse>
```

Figure 2: XML-RPC Response

Likewise to the request message, the XML-RPC response uses a similar format. From both of these message structures, one may notice that there are inadequacies to the protocol design [1].

- There is no clean method of transporting XML documents using XML-RPC. “XML document” is not a pre-defined type.
- The structure of XML-RPC is not extensible to include more complex data types.
- XML-RPC does not follow the W3C XML Schema recommendation as it does not use XML namespaces and it defines its own data types.
- XML-RPC is “bound to” HTTP whereas some applications might require a more sensible transfer protocol such as SMTP.

The next section analyzes SOAP messages and in comparison, we would discover that SOAP is the solution to the various drawbacks of XML-RPC.

2.2 The Structure of SOAP

To understand how a SOAP message is made up, it would be helpful to firstly take a step back to see the overall layout.

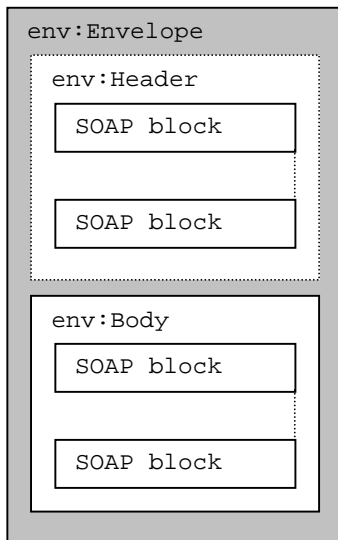


Figure 3: SOAP Message Framework

As we can see in Figure 3, a SOAP message is made up of three main elements – The Envelope, the Header and the Body. Apart from the Header, which is optional, the other two are mandatory [7][8].

The Envelope acts as a container encapsulating the other two elements and is also used to define XML namespace information. The Header contains information relevant for processes such as routing and authentication. Thirdly, the Body contains the actual message content. Another special block that may appear in a message is the Fault block. If this block were present, it would appear within Body. More information about SOAP faults would come later in this section.

When a client node sends a SOAP message to the server node, the message may directly reach the server or it may pass through one or many “intermediary nodes” before reaching the server. It is likely that some, or even all of the SOAP blocks within the Header targets these intermediary nodes. When a SOAP message is passed through an intermediary node, the Header may be modified or removed completely when reaching the target node. [1]

As a side-note, the term “SOAP block” is used by SOAP to describe a block of data seen by the processor of the message as a single computational unit of data. The SOAP blocks within the Header section, are known as the “Header blocks” and likewise, the blocks within Body are called the “Body blocks”. [1]

Figure 4 below demonstrates a simple SOAP RPC message similar to the XML-RPC call in Figure 1.

```

<?xml version='1.0' ?>
<env:Envelope
xmlns:env="http://www.schemas.xmlsoap.org/soap/envelope/"
encodingStyle="http://www.schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <env:Body>
    <app:NumberToText
env:encodingStyle="http://www.schemas.xmlsoap.org/soap/envelope/"
xmlns:app="http://www.andy-tan.com">

      <inputnum
xsi:type="xsd:int">41</inputnum>

      <language
xsi:type="xsd:string">English</language>

    </app:NumberToText>
  </env:Body>

</env:Envelope>

```

Figure 4: SOAP RPC Request Message

One of the first differences you may notice in SOAP when compared with XML-RPC is the use of namespaces. XML namespaces is a W3C standard mechanism that avoids conflicts with tag names. SOAP is fully compliant with namespaces and, interestingly, SOAP uses different namespaces for its versioning. Because of the differences, both versions are not compatible with each other. A simple way to distinguish SOAP 1.1 from 1.2 messages is to notice the different namespaces defined at the envelope tag.

SOAP 1.1 has an envelope namespace of <http://www.schemas.xmlsoap.org/soap/envelope/>

- SOAP 1.2 is <http://www.w3.org/2002/12/soap-envelope>

From Figure 4, you may see that the request is a SOAP 1.1 message. Apart from the different envelope namespaces, there are also other differences between the two versions [7]. However, this paper would not elaborate on this issue for the sake of brevity and keeping within the research scope.

After this message has been received and processed by the server node, a response message (Figure 5) will then be generated and returned to the method-invoking node.

```

<env:Envelope
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
encodingStyle="http://schemas.xmlsoap.org/soap/envelope/">
<env:Body>

<app:NumberToText
env:encodingStyle="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:app="http://www.andy-tan.com">
<answer>forty-one</answer>
</app:NumberToText>

</env:Body>
</env:Envelope>

```

Figure 5: SOAP RPC Response Message

Both the request and response messages above share the same SOAP envelope structure. In fact, there is also another type of message that SOAP uses which shares the same structure.

Whenever there is an error while processing a SOAP request, a server node may return a SOAP fault message to the originator, indicating the type of error that has been encountered. As mentioned above, a fault block (written as `<env:Fault></env:Fault>`) is located within the Body element. There are four elements that could be included within this block [1][8]:

- `<faultcode>` - This element is mandatory for inclusion in the fault block. It describes the fault with five different faultcode values:
 1. **VersionMismatch** – this indicates that the processing node could not recognize the envelope namespace of the request message. This may be due to different SOAP versions in use by the SOAP nodes in communication.
 2. **MustUnderstand** – this indicates that the processing node could not understand the SOAP Header element.
 3. **Client** – this indicates that the message was not formed correctly, or didn't contain the appropriate information.
 4. **Server** – this indicates that the error is not with the message itself, but rather the state of the server node.
 5. **DataEncodingUnknown** – this indicates that the server node does not recognize the encodingStyle. This faultcode is only defined on SOAP 1.2.
- `<faultstring>` - A mandatory element describing the error in words.
- `<faultactor>` - This is an optional element indicating the URI of the source of the fault.
- `<detail>` - This is an optional element that contains further information of the error if the error occurred in the Body element. Otherwise, it is left out.

If an error had occurred after the request of Figure 4, the following error message in Figure 6 might result.

```

<env:Envelope
xmlns:env="http://schemas.xmlsoap.org/sc
ap/envelope/"
encodingStyle="http://schemas.xmlsoap.or
g/soap/envelope/">
<env:Body>

<env:Fault>
<faultcode>env:Client</faultcode>
<faultstring>Client error</faultstring>
<detail xmlns:app="http://www.andy-
tan.com">
<app:errorMessage>
This language is not supported.
</app:errorMessage>
</detail>
</env:Fault>

</env:Body>
</env:Envelope>

```

Figure 6: SOAP Fault Message

2.3 SOAP Transfer of Binary Data

There are some instances when the transfer of binary files is necessary through SOAP. For example, a Web service that allows users to call a method from a remote client and convert their BMP graphic files into JPEG. However, XML on its own does not handle binary data very well and often produces significantly large overhead [10], so hence the designers of SOAP have employed some techniques to get around the problem.

All of these methods use a model termed Compound SOAP as the base structure. Compound SOAP is defined in the SOAP 1.2 Attachment Feature working draft [11] to be consisting “of a primary SOAP message part and zero or more related secondary parts.” The secondary parts are also known as “Attachments”. Figure 7 represents a structure diagram to better understand the data structure.

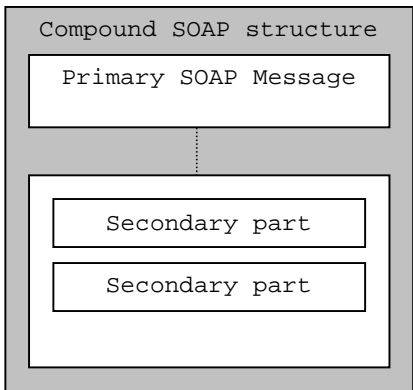


Figure 7: A diagrammatic view of Compound SOAP

It should be noted that the working draft specifications declared the area of implementation of the compound SOAP model to be outside its scope. Because of this, implementation methods are open to designers adopting this framework.

There are currently two popular methods of handling attachments in SOAP and they are SwA (SOAP with Attachments) [12] and WS-Attachments [13]. SwA is a W3C note (December 2000) developed by Microsoft and Hewlett Packard that employs Copyright © Freestylemedia. The information contained within this document subsists under the Copyright Laws of Australia and through international treaties and the laws of other countries. It is owned by Freestylemedia. All rights reserved. Except as expressly permitted in this document no part of this document may be reproduced, used, manipulated, adapted or transmitted without the specific written consent of Freestylemedia.

MIME (Multipart Internet Mail Extensions) as the attachment structure. WS-Attachments, developed by Microsoft and IBM, is similar to SwA but uses DIME (Direct Internet Message Encapsulation) to arrange the attached data. We will now look at both attachment methodologies in more detail.

SwA messages look very similar to a normal email message because emails utilize the same MIME structure to include attachment data. Please refer to the example SwA message in Figure 8.

It is not too difficult to understand a multi-part MIME message as a unique string separates each part in the message. For this example, the unique string is written as "xXxXxXx". The "--" prefix and suffix indicate the start and finish of the parts. This partition technique assumes that the SOAP message or the attachments do not contain that unique string. Each part also contains a Content-ID so both the SwA header and the SOAP message can refer to it.

Under comparison, WS-Attachment messages are more difficult to understand. This is because DIME itself uses binary code to describe each part rather than using a simple unique partition string like MIME. Moreover, unlike MIME, DIME breaks attachments into pieces of data called "records" of a set size. This may mean that a large file may be "chunked" into a few records during transfer. DIME's binary encoding at the headers actually describes the data of each record rather than every individual attachment part like MIME. The binary includes information such as the record length, whether the record is the first or last record, or whether the contents of the message have been chunked [14][15]. Figure 9 gives an example of a WS-Attachment message.

```
Content-Type: multipart/related;
type=text/xml;
boundary="xXxXxXx";
start="<start-1234.56@www.andy-tan.com>"

This is movie file you have ordered.

--xXxXxXx
Content-Type: text/xml; charset="UTF-8"
Content-ID: <start-1234.56@www.andy-
tan.com>

<env:Envelope>
<env:Body>
<tns:Duration>99</tns:Duration>
<tns:Rating>PG</tns:Rating>
<tns:Movie href="cid:part1-
23456.78@www.andy-tan.com"/>
</env:Body>
</env:Envelope>

--xXxXxXx
Content-Type: application/mpeg
Content-Transfer-Encoding: 8bit
Content-ID: <part1-23456.78@www.andy-
tan.com>

<< The binary data goes here >>

--xXxXxXx--
```

Figure 8: A Sample SwA Message

```
000011000010000000000000000000000000
00000000000000000000000000000101000
0000000000000000000000000000110110101
http://schemas.xmlsoap.org/soap/envelope
<env:Envelope>
<env:Body><msg:GetMediaFile>
<msg:fileName>MovieFile.mpg
</msg:fileName>
<msg:file
ref:location="uuid:F2DA3C9C-74D3" />
</msg:GetMediaFile>
</env:Body></env:Envelope>
000010010001000000000000000000000000
000000000010100100000000000001010
000000000010101101010101011100000
uuid:F2DA3C9C-74D3
application/mpeg
<< First 1.42 MB of binary data here >>
000010100000000000000000000000000000
000000000000000000000000000000000000
00000000000010000110110001000000
<< Remaining 552 KB of binary data >>
```

Figure 9: A Sample WS-Attachment Message

From the Figure 9 example, you may see that the WS-Attachment specifications require a unique ID to be included after the DIME encoding. At the header there exists a URI of the envelope type. However, this only applies to SOAP 1.1. For SOAP 1.2, there should instead be a MIME type of “application/soap+xml” as described in the Internet draft presented to the IETF [16].

If SOAP 1.2 finally does become standardized, we should expect the same with one of these two attachment frameworks. SwA is simple and easily implemented, but WS-Attachments seem to give better performance with less overhead. Another advantage of WS-Attachments over SwA is the ability to stream data, whereas SwA can only be processed once all parts are received [10].

2.4 SOAP Transport Options

Although most SOAP implantations today use HTTP as the main transport protocol of SOAP, it should be noted that SOAP has other transport options too. SOAP’s modular design will also enable it to have “binding” with non-HTTP protocols since transport is at a lower layer.

In the near future, SMTP may also become a successful binding protocol since there are instances when HTTP may not be the most appropriate option. For example [17]:

- Some firewalls may block out HTTP traffic.
- A request/response messaging model does not fit into your application model.
- Your application may not be real time. Applications that may take more than the 300 seconds would usually reach a timeout with the HTTP server.

3. Conclusion

SOAP is a lightweight and extensible XML protocol, which makes it a protocol of choice when it comes to Web Services. It should be pointed out that because of the modular nature of SOAP, it is not limited to Web Services or RPC. Unlike XML-RPC, SOAP is more like a toolbox. A technology that demonstrates this and is growing in popularity is ebXML. The ebXML framework is an initiative to create a consistent and uniform XML messaging format for small and medium size businesses, and other organizations or governments that have not been able to benefit from traditional EDI frameworks employed by larger organizations [8]. In fact, ebXML borrows the SOAP envelope model and the transport bindings and builds on top of the framework with its own encoding rules [1]. This is only one example of the extensible nature of SOAP.

The next major step forward for SOAP would be the standardization of version 1.2. Since this has already achieved Candidate Recommendation status, it would not be too long until we see it become a full recommendation and see many new distributed applications that would follow.

References

- [1] R. Brunner, F. Cohen, F. Curbera, D. Govoni, S. Haines, M. Kloppmann, B. Marchal, K.S. Morrison, A. Ryman, J. Weber, M. Wutka, *Java web services unleashed* (Indiana USA: Sams Publishing, 2002)
- [2] D. Austin, A. Barbir, C. Ferris, S. Garg, Web Services Architecture Requirements - W3C Working Draft 14 November 2002
<http://www.w3.org/TR/wsa-reqs>
- [3] D. Winer, Foreword for the XML-RPC book, 16 May 2001.
[http://www.xmlrpc.com/stories/storyReader\\$1726](http://www.xmlrpc.com/stories/storyReader$1726), accessed April 2003.
- [4] D. Box, A Brief History of Soap, 4 April 2001.
<http://webservices.xml.com/pub/a/ws/2001/04/04/soap.html>, accessed April 2003.
- [5] R. Salz, What is XML-RPC?, 18 December 2002.
<http://webservices.xml.com/pub/a/ws/2002/12/18/endpoints.html>, accessed April 2003.
- [6] D. Winer, XML-RPC Specification, 15 June 1999.
<http://www.xmlrpc.com/spec>, accessed April 2003
- [7] N. Mitra, SOAP Version 1.2 Part 0: Primer - W3C Candidate Recommendation 19 December 2000.
<http://www.w3c.org/TR/soap12-part0/>, accessed April 2003.
- [8] D.A. Chappell, V. Chopra, J.J. Dubray, C. Evans, B. Harvey, T. McGrath, D. Nickull, M. Noordzij, B. Peat, P. van der Eijk, J. Vegt, *Professional ebXML foundations* (Birmingham UK: Wrox Press Ltd, 2001)
- [9] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, R. Neyama, *Building web services with java: making sense of XML, SOAP, WSDL, and UDDI* (Indiana USA: Sams Publishing, 2002)
- [10] R. Salz, Transporting Binary Data in SOAP, 28 August 2002.
<http://webservices.xml.com/pub/a/ws/2002/08/28/endpoints.html>, accessed April 2003.
- [11] H. F. Nielsen, H. Ruellan, SOAP 1.2 Attachment Feature - W3C Working Draft 24 September 2002.
<http://www.w3.org/TR/soap12-af/>, accessed April 2003.

- [12] J.J. Barton, S. Thatte, H.F. Nielsen, SOAP Messages with Attachments - W3C Note 11 December 2000
<http://www.w3.org/TR/SOAP-attachments>, accessed April 2003
- [13] H.F. Nielsen, E. Christensen, J. Farrell, WS-Attachments, 17 June 2002.
<http://www-106.ibm.com/developerworks/library/ws-attach.html>, accessed April 2003
- [14] M. Powell, Understanding DIME and WS-Attachments, October 2002
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/dimewsattach.asp>, accessed April 2003.
- [15] J.H. Gailey, Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation, December 2002.
<http://msdn.microsoft.com/msdnmag/issues/02/12/DIME/default.aspx>, accessed April 2003.
- [16] R. Salz, Brother, Can You Spare a DIME?, 18 September 2002.
<http://webservices.xml.com/pub/a/ws/2002/09/18/ends.html>, accessed April 2003.
- [17] M. Olson, U. Ogbuji, Sending and receiving SOAP requests over SMTP, 4 March 2003
<http://www-106.ibm.com/developerworks/webservices/library/ws-pyth12.html>, accessed April 2003.