A Critique of Web Services

Jan Newmarch
School of Network Computing
Monash University

Web services possibly represent the next stage of development of the Web where human/server interaction is replaced by application/application interaction. Web services are intended to provide remote procedure call mechanisms using the accepted standards of HTTP and XML. This paper argues that the current technology for Web services is a poor implementation of 30 year old technology and will need substantial revision before it becomes viable.

Introduction

Web services represent a development of the use of the Web. Initially the Web was used to transport pages of HTML from a filesystem somewhere on the internet to a browser which would render it and display it to a user. The static Web pages could be replaced by dynamically generated pages, but that in itself was not significant.

A major evolution was caused by the introduction of HTML Forms and formalisation of methods to transfer information from a browser to a server-side program. This allowed genuine two-way communication between a user of a browser and a server-side program and has lead to widescale Consumer-to-Business ecommerce. It should be noted that this showed up a major design flaw in the HTTP protocol in that it is session-less (to be precise, this was a design-plus for the original get-response model for fetching Web pages, but not for the new uses). A session-oriented model is needed to manage a shopping cart, for example. This has been handled not by changing HTTP, but by using hacks such as cookies, URL-rewriting and hidden fields in Forms. Normally such poor software engineering would quickly be exposed, but instead the user population has trained itself to work within these constraints (for example, by not turning cookies off). If they did not obey these constraints, then a large number of Web pages would become un-navigable.

Web services (in the strictly technical sense as used by the WWW Consortium) propose to replace the human element by a user-agent which is probably a program [1].

> *A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols*

These will allow agent-to-agent communications. For example, instead of a user searching the Web for the best deal on a product, an agent program will be able to perform this search on the user's behalf. Instead of looking at HTML pages, it will interact directly with server-side programs using XML.

While the language of "Web services" is new, the concepts themselves are not. Really, Web services are just a form of distributed computing, and are in fact just another form of Remote Procedure Call [2]. There are many such systems, from Sun's RPC (renamed as the IETF ONC), DCE, COM+ and object-oriented versions such as CORBA and Java RMI. There is clearly a business-need for a widely accepted RPC protocol, and proponents claim that whereas previous middleware systems have failed, Web services will fill this need.

This paper looks at Web services from a technical perspective. Most of the points made in this paper have been "obvious" for many years, but there does not appear to be a clear statement of them in the literature. In short, the thesis of this paper is that the current version of Web services is a particularly bad implementation of 30 year-old technology, and that if Web services succeeds they will do so only by the strenuous efforts of advocates to paper over the glaring flaws.

Components of a Web Service

Web services are generally considered to made up of three components [3]

- SOAP - simple object access protocol [4]
- WSDL - web services description language [5]
- UDDI - universal description, discovery and integration [6]

The function of each component is

- WSDL is the metalanguage used to describe a Web service. It corresponds to the IDL (Interface Definition Language) of other RPC systems
- SOAP is the wire format for remote method calls and their replies. It corresponds to the XDR level of Sun's RPC
- UDDI is the resource registration and lookup mechanism. It corresponds to the Naming Services of CORBA, RMI

The W3C is a little coy about how a Web service is invoked. They consider a number of models, but are only definite about one transport protocol: SOAP over HTTP. This reflects two aspects of the W3C: they only have control of the specification of HTTP and not of other protocols such as TCP or email, but perhaps more importantly reflects what appears to be a *mantra* within the W3C that since HTTP is so successful on the Web, then it should be the protocol for *everything*. This is despite the already noted hacks to add session management.

The W3C is (rightly) not concerned with implementations of their standards. However, it may be useful to summarise the current state of Web services and how they relate to RPC functions

RPC requirement	**SOAP**
defined data types	defined by SOAP
wire format for data	defined by SOAP using XML
wire format for messages	defined by SOAP over HTTP
an IDL	defined by WSDL
generation of client stubs	vendor specific
generation of server stubs	vendor specific
linking implementation	vendor specific
Address of service	URI
Locating service	UDDI

WSDL and specification languages

The role of a specification language is to convey information about a service in a manner that is independent of any implementation. There are many examples of specification languages used for distributed systems, such as CORBA IDL and Java interfaces. In all of these cases, the specification language is simple enough that it can be created by system designers before being given to programmers to implement in their chosen programming language.

The IDL for Web services is WSDL. WSDL is an XML Document Type Definition that controls a set of XML documents. A WSDL document is then an XML document that conforms to the WSDL DTD. Of course, the document is intended to have meaning: a WSDL document acts as the specification of a Web service. As such, it should be possible for a designer to create a WSDL specification without also specifying an implementation.

Specification languages can act at the syntactic or semantic level. Syntax is usually easy, as is seen with CORBA IDL. Semantics is hard, and there are many languages such as Z to deal with semantics. However, this is far from a settled area, and there are ongoing attempts to come up with usable semantic specification languages. But WSDL does not function at the complex semantic level, only at the simpler syntactic level.

Consider a simple example. Suppose we have the following informal specification of a service

```
Converter service:
   float inchToMM(float)
   float mmToInch(float)
```

Using CORBA IDL this becomes

```
interface Converter {
    float inchToMM(in float value);
    float mmToInch(in float value);
};
```

Using Java RMI it would be

```
    public interface Converter implements Remote {
        public float inchToMM(float value) throws RemoteException;
        public float mmToInch(float value) throws RemoteException;
    }
```

Both of these are syntactic specifications only - there is no semantics (apart from data types) in these.

A WSDL specification is syntax only as well (with a minor - regretful - addition discussed later). The WSDL specification for this service is

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns="urn:Converter" targetNamespace="urn:Converter" name="ConverterService">
    <message name="inchToMMRequest">
        <part name="param1" type="xsd:float"/>
    </message>
    <message name="inchToMMResponse">
        <part name="return" type="xsd:float"/>
    </message>
    <message name="mmToInchRequest">
        <part name="param1" type="xsd:float"/>
    </message>
    <message name="mmToInchResponse">
        <part name="return" type="xsd:float"/>
    </message>
    <message name="java_rmi_RemoteException">
        <part type="xsd:string" name="java_rmi_RemoteException"/>
    </message>
    <message name="com_iona_xmlbus_webservices_ejbserver_ConversionException"
>
        <part type="xsd:string" name="com_iona_xmlbus_webservices_ejbserver_ConversionException"/>
    </message>
    <portType name="ConverterPortType">
        <operation name="inchToMM">
            <input message="tns:inchToMMRequest" name="inchToMM"/>
            <output message="tns:inchToMMResponse" name="inchToMMResponse"/>
            <fault message="tns:java_rmi_RemoteException" name="java_rmi_RemoteException"/>
        </operation>
        <operation name="mmToInch">
            <input message="tns:mmToInchRequest" name="mmToInch"/>
            <output message="tns:mmToInchResponse" name="mmToInchResponse"/>
            <fault message="tns:java_rmi_RemoteException" name="java_rmi_RemoteException"/>
            <fault
                message="tns:com_iona_xmlbus_webservices_ejbserver_ConversionException" name="com_iona_xmlbus_webservices_ejbserver_ConversionException"/>
        </operation>
    </portType>
    <binding name="ConverterBinding" type="tns:ConverterPortType">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http/" style="rpc"/>
        <operation name="inchToMM">
            <soap:operation soapAction="" style="rpc"/>
            <input name="inchToMM">
                <soap:body use="encoded" namespace="urn:Converter" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output name="inchToMMResponse">
                <soap:body use="encoded" namespace="urn:Converter" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
            <fault name="java_rmi_RemoteException">
                <soap:fault name="java_rmi_RemoteException"
                    use="encoded" namespace="urn:Converter" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </fault>
        </operation>
        <operation name="mmToInch">
            <soap:operation soapAction="" style="rpc"/>
            <input name="mmToInch">
                <soap:body use="encoded" namespace="urn:Converter" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output name="mmToInchResponse">
                <soap:body use="encoded" namespace="urn:Converter" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
            <fault name="java_rmi_RemoteException">
                <soap:fault name="java_rmi_RemoteException"
                    use="encoded" namespace="urn:Converter" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </fault>
            <fault name="com_iona_xmlbus_webservices_ejbserver_ConversionException">
                <soap:fault
                    name="com_iona_xmlbus_webservices_ejbserver_ConversionException"
                    use="encoded" namespace="urn:Converter" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </fault>
        </operation>
    </binding>
    <service name="Converter">
        <port name="ConverterPort" binding="tns:ConverterBinding">
            <soap:address location="http://www.xmlbus.com:9010/ionasoap/servlet/Converter"/>
        </port>
    </service>
</definitions>
```

This verbosity is craziness! Just for syntax! This is so bad, that the universal recommendation for the creation of a WSDL document is this (e.g. [8]):

> *Write the implementation in your favourite language - which you understand. Then reverse-engineer this to a WSDL specification. Publish this widely in the knowledge that no human will ever read it. If someone-else wishes to implement this service or write a client for this*

service, then get them to forward engineer it to their *favourite programming language - which they will understand.*

Software engineering is sufficiently young that there are not many dead software engineers - which is the only reason that not many will turn in their graves. This is, quite simply, appalling software engineering and should be rejected on these grounds alone.

Structure of a WSDL document

WSDL documents have structure. For the example above, it is:

Data types
- XML data types are okay
- Add extra data types here

Messages
- `inchToMMRequest`
- `inchToMMResponse`
- `java_rmi_RemoteException`
- ...

Ports
- `inchToMM(inchToMMRequest, inchToMMResponse, java_rmi_RemoteException)`
- ...

Bindings
-
  ```
  ConverterBinding
  operation: inchToMM/rpc
             "inchToMM"
             "inchToMMResponse"
             "java_rmi_RemoteException"
  ```
- ...

Service

```
name: Converter
      address: http://...
...
```

While there is logic here, there is also a large degree of redundancy. In addition to this, there is also an element which has no place in a specification: the *location* of the service. Oh darn: I have two identical services but at different locations, so *of course* the specifications are different... This criticism has been identified by the UDDI community who recommend as "best practise" that an obscure part of WSDL (an "include" mechanism) be used to partly separate specification and location [9]. It may be addressed in future versions of WSDL.

SOAP and transport protocols

The meaning of SOAP is officially: Simple, Object Access Protocol. It also claims to be "a lightweight protocol for exchange of information in a decentralized, distributed environment"

The first claim is that it is an object protocol. There are several issues here. An "object" is an entity that maintains state and can be invoked via method calls. In addition, it can inherit implementation mechanisms from "super classes". In using the phrase "object" the purist understands that some "white box" information is available: for example in Java reflection can be used to determine the inheritance hierarchy. A "component" on the other hand is just an entity that maintains state and there is no notion of inheritance.

A SOAP "object" is defined by a WSDL document. This has no notion of inheritance, so from a purist viewpoint a SOAP "object" is not an object at all. At most, it could be a component. Of course, an implementation of a SOAP "object" could be a real object in an O/O language such as C#, but there is no requirement for this and it could just as easily be Fortran routines.

Is it even a component? It is accessed by methods, but does it maintain state? There is no requirement for it to do so. There are no identified state variables, and no agreement on get/set methods which might imply state. A WSDL specification just defines a set of procedure calls at best. To claim that it is an "object" protocol is simply false.

The second claim is that it is simple. A SOAP message is also defined by an XML DTD. A typical (small) request might be

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">

    <SOAP-ENV:Body>
        <ns1:getTemp xmlns:ns1="urn:xmethods-Temperature"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"&g
t;
            <zipcode xsi:type="xsd:string">11217</zipcode>
        </ns1:getTemp>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

which essentially calls `getTemp(zipcode)`. Again there is a level of verbosity here which is quite unneeded. Simplicity does not appear to be the right word to use here.

The author is unaware of any performance studies of this protocol, but all of my colleagues who are aware of the structure of this protocol laugh at the suggestion that you might use it to stream data which might require high bandwidth such as A/V data. This is not lightweight by any means! There is even a group within the W3C working on compression techniques for XML documents, suggesting that even the W3C feels the same way [10].

UDDI and service discovery

UDDI appears to be an independent effort to establish the "directory for everything". It has been linked to Web services as a means for publishing Web services.

UDDI contains several different directory structures

- White pages
- Yellow pages
- Service pages

The stated intent is that there will be a small group of UDDI directories replicating directory structures for all services and organisations in the world. The design of UDDI reflects this global, largescale intent. While Web services may need directories to in order to locate them, such a highly-centralised system seems an odd way to go: the UDDI central directories are far more centralised than even the internet root name servers (which delegate as much as possible), and is a long way from the central idea of the Web: no central servers, just a hyperlinked arbitrary directed graph.

UDDI allows a service to register information. A service can alter its information or remove its entry. There does not seem to be any mechanism to "clean up" stale entries (i.e. garbage collect them). During the dot.com boom, if UDDI registries had existed, then there could be a lot of entries with no-one able to remove them...

On the plus side, UDDI separates specification (tModel) from implementation (binding template). WSDL puts both specification and binding in the same document. UDDI "best practice" recommends that WSDL documents should be in two parts, with binding "including" specification. Thus, it points out a weakness in the WSDL format.

UDDI appears to be a poor fit to Web services: designed for a different purpose, it does not mesh well with either concepts from the Web or details of WSDL.

HTTP and session management

HTTP is a session-less protocol [11]. HTTP is a protocol built above TCP which is session-oriented. HTTP uses a TCP session to send a request and return a response. Once a response is received the connection is dropped. That was sufficient for simple document return. With complex documents such as a page with multiple images the setting up and tearing down of TCP sessions proved too expensive, so HTTP 1.1 allows multiple HTTP sessions to re-use a single TCP session. But this does not alter the fundamental characteristic of HTTP being session-less.

Session management for browsers has already been mentioned. The techniques used are cookies, URL rewriting and hidden fields in forms. These are all hacks. None of these will work as general methods for Web services

- Cookies are passed at the HTTP layer and are made available to the application layer (CGI scripts, servlets, etc) through a defined mechanism in HTML. SOAP does not require use of the HTTP protocol and is not HTML. Even if a SOAP request is run over HTTP, there is no defined way of passing information between the layers
- Similarly with URL rewriting, there is no defined way of passing information from the service URL to a SOAP method call
- Hidden Form fields? What are Forms to a service where the clients have no UI?

The current state of Web services with regard to session management is: "roll your own". The restriction to session-less HTTP and the lack of specification with regard to all other protocols means that you have to decide your own session management system. Never mind, you can always read the TCP specification to see how they managed it over IP... For example, Microsoft Web services use cookies in a <session> tag in the SOAP header. This does not address session timeout information or management of cookies, and just assumes that all components of the session management system will function satisfactorily

Security

The W3C has a group working on encryption and authentication mechanisms for XML documents [12]. The Web services group appears content to use this work [13]. With regards to encrytion, there are several ways of encrypting an XML document or portions of the document

- encrypt the entire document
- encrypt an element - both the element name and its content
- encrypt an element’s content

Note that encrypting the content of an element will encrypt all of the sub-elements names and their contents

If the entire document is encrypted, then it is no longer a SOAP document but is instead an encrypted XML document - no stage of the transport mechanism will be able to identify this as a SOAP document. Intermediate SOAP processing nodes will not be able to deal with this document unless it is first decrypted.

If an element is encrypted, then the document has been transformed into one that no longer follows the schema for the document - its structure has been altered. Any agents that need to know the structure will not be able to deal with it. In particular, the transport mapping from SOAP document to HTTP transport will be broken, because a different document is being transported.

A similar consideration occurs when the content of an element is encrypted. While it may appear that only a single datatype is involved, in fact the content may be a complex XML document, so that it has the same implications for document structure as encrypting an element.

Conclusion

While there is a strong business need for a universally acccepted remote procedure call mechanism, there are fundamental problems with Web services that will make it hard for them to fulfil this role. There are problems with SOAP, WSDL and UDDI which will take substantial efforts to overcome. For example, one of the criticisms cited by the Web services community against against RPC systems such as CORBA is that they are "overweight" compared to "lightweight" Web services. While it is already demonstrably false that Web services are lightweight, the changes required to fix the problems with Web services may result in criticisms of this type becoming totally vacuous.

References

[1] WWW Consortium *Web Services Architecture*
http://www.w3.org/TR/2003/WD-ws-arch-20030514
[2] A. S. Tanenbaum *Modern Operating Systems* Prentice-Hall 1992
[3] G. Glass *WS components*
http://www-106.ibm.com/developerworks/webservices/library/ws-peer1.html?dwzone=ws
[4] World Wide Web Consortium *SOAP Version 1.2 Part 0: Primer*
http://www.w3.org/TR/2002/WD-soap12-part0-20020626/

[6] WSDL 1.0 Specification, http://http://www.ibm.com/developerworks/web/library/w-wsdl.html
[7] UDDI home page http://www.uddi.org
[8] G. Glass *e.g create WSDL from impl*
http://www-106.ibm.com/developerworks/webservices/library/ws-peer4/
[9] UDDI *Using WSDL in a UDDI Registry, Version 2.0*
http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm
[10] WWW COnsortium *SOAP Message Transmission Optimization Mechanism*
http://www.w3.org/TR/soap12-mtom/
[11] WWW Consortium *Hypertext Transfer Protocol* -- *HTTP/1.1*
http://www.w3.org/Protocols/rfc2616/rfc2616.html
[12] WWW Consortium *XML Signature WG* http://www.w3.org/Signature/
[13] WWW Consortium *Web Services Architecture Usage Scenarios*
http://www.w3.org/TR/2003/WD-ws-arch-scenarios-20030514

Jan Newmarch (http://jan.netcomp.monash.edu.au)

jan.newmarch@infotech.monash.edu.au
Last modified: Tue Nov 4 03:55:20 EST 2003