The ITB Journal

Volume 3 | Issue 2 Article 10

2002

Web Services Technology Infrastructure

Geraldine Gray

Kieran O'Connor

Web Services Technology Infrastructure

Geraldine Gray & Kieran O'Connor (2002)

School of Informatics and Engineering
Institute of Technology Blanchardstown, Dublin, Ireland

Kieran.O'Connor@itb.ie

Abstract

Web Services using eXtensible Markup Language (XML) based standards are becoming the new archetype for enabling business to business collaborations. This paper describes the conceptual architecture and semantics of constructing and consuming Web Services. It describes how Web Services fit into the enterprise application environment. It discusses Web Services security. Finally, it outlines the flaws of Web Services in their current state.

Introduction to Web Services

> '*A Web Service is an application that accepts requests from other systems across the Internet or an Intranet, mediated by lightweight, vendor-neutral communication technologies*' [Kao, 2001, section II: Introduction]

Web Services expose business functionality over a network. It is not the exposed functionality that makes Web Services revolutionary; it is the lightweight, vendor neutral communication technologies. XML based standards to which vendors and developers will comply such as SOAP1.1 (and soon SOAP1.2) makes the process of inter-system communication easier. Particularly from a business-to-business perspective, Web Services as they stand today offer a huge technology leap in web enabling back end business logic.

Usually, for two businesses to share data, complex systems and communication agreements would potentially have to be developed because of the differing proprietary applications within each company. With Web Services the interaction layer is abstracted away so that developers can concentrate in providing back end functionality with the knowledge that such functionality can potentially be web enabled because Web Services will define the communication mechanisms in an interoperable way.

The most important industry defined standards that allow for interoperable communication are as follows:

- SOAP and WSDL: World Wide Web Consortium (W3C).
- UDDI: UDDI.org.

These standards define how to publish, describe and invoke a Web Service irrespective of the underlying operating system. SOAP (see page 5) defines how messages are transmitted. WSDL documents define how services are described. UDDI defines how to maintain Web Services and company information within an XML based registry.

Market Place

The market place today is influenced by many diverse companies. The contenders include Sun, IBM, Microsoft, and IONA among others. All these companies offer either Web Services toolkits or are involved in developing application servers that work with Web Services. These toolkits and Web Services compliant application servers are based upon the standards such as SOAP and WSDL. Each has its own implementation but because of the interoperable nature of Web Services most development should be portable between different proprietary software.

Sun has its SunONE Web Services architecture based on XML and Java, including its own Integrated Development Environment (IDE) and incorporating the iPlanet server family. This provides Web Services functionality to the J2EE environment. Microsoft has developed its .NET framework which provides support for web services using Microsoft specific technologies such as Visual Basic, C#, and the Microsoft server family. Other application servers have been developed with Web Services in mind, including Macromedia's JRun4.0, BEA's WebLogic, and Systinet WASP Server for Java to name but a few. There could be as many as twenty five or more SOAP APIs on the market today judging by the interoperability tests carried out by Apache AXIS. Open source projects supporting Web Services include: Apache's SOAP API AXIS and the JBoss application server.

Companies that have already implemented Web Services solutions include Amazon and Google. Amazon has produced a service that allows programmatic access to their catalogue, search engine, shopping cart and merchandising tools. Google has produced a service that allows programmatic access to their World Wide Web search engine. These services allow their functionality to be incorporated into any Web Service application.

Benefits & Shortcomings

There are a number of advantages and disadvantages to using Web Services over other similar architectures such as CORBA. Web Services claim to be platform independent, loosely coupled and can navigate firewalls over the HTTP transport protocol. Web Services

have industry backing which will facilitate their evolution. CORBA IIOP on the other hand is not platform independent. CORBA IIOP usually requires an infrastructure which includes a CORBA ORB; this limits developers to vendors who support CORBA [Monson-Haefel, 2002, p3]. A distributed application using CORBA is tightly coupled meaning that A must know about B and vice versa for the creation of Interface Definition Language (IDL) files to map data types. CORBA IIOP has no protocol specification for firewall navigation [De Jong, 2002, p4]. Disadvantages of Web Services include performance and security. Web Services rely on XML parsers to understand a particular message. This process can be slow depending on the size of the message being parsed. HTTP can be slow and unreliable as a method for transporting SOAP messages. Security is not yet well defined for Web Services. Some standards have been released and others are in the pipeline; it could be some time however before vendor support for these standards is realised. There are multiple SOAP engine implementations which can result in non portable code. Apache run tests to check how the popular SOAP implementations respond to AXIS clients which highlight these incompatibilities. Thus, Web Service code is not truly portable despite the standardisation of SOAP.

A Conceptual Model

The classic Web Services architecture defines three roles [Chappell & Jewell, 2002, pp 14-23].

- Service Requester
- Service Provider
- Service Registry/Broker

The Service Provider is the organisation responsible for creating the Web Service. First the business logic which possibly accesses databases and/or legacy/ERP systems is created. They then expose this functionality as a Web Service by publishing the organisation and Web Service details to a global registry (UDDI or ebXML). The Service Requester is the organisation that requires the use of the functionality exposed by the Web Service in question. This organisation queries the global registry to find and use a suitable Web Service. The Service Registry/Broker is an XML based data store for information including company name, contact information, and pointers to a Web Services Description Language (WSDL) file that details how to use a particular service.

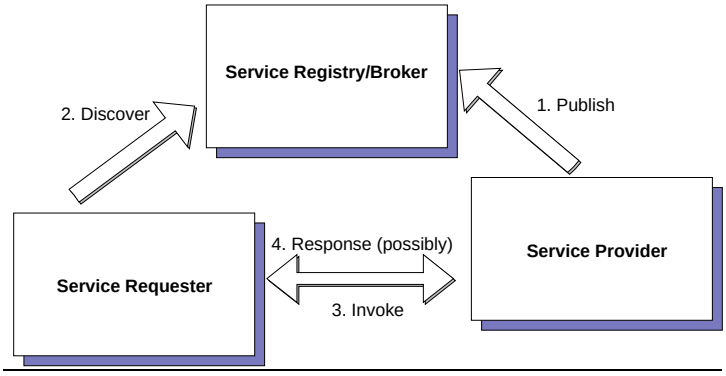

Figure 1 – Web Service Publish & Discovery Architecture.

Standards Overview

There are three major technologies that have enabled the development of Web Services and more importantly, enabled them to address the problems posed with integrating remote systems. These standards define the data transport mechanisms, how to describe what is being transported, and how to make it easy to locate Web Services. These standards are XML based, which is a universally accepted textual data format.

SOAP

The *Simple Object Access Protocol* is a lightweight distributed computing protocol that allows information to be exchanged in a decentralised environment [Hendricks et al., 2002, pp 33-61]. SOAP ensures interoperability between differing application environments by defining a messaging standard. A C++ client (for example) could create a SOAP message and send it to a remote Java Web Service over HTTP, invoking a remote procedure call (RPC) on that service. The Java Web Service would understand this request as it understands SOAP and could return a result, again encoded within a SOAP message.

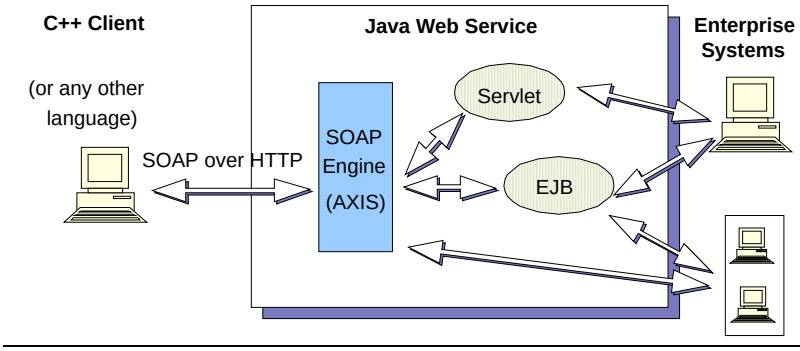

Figure 2 – Web Services Integration, through SOAP, with Enterprise Applications.

A SOAP message can have four basic sections:

- Transport Information: Transport protocol specific information. This section also contains a SOAPAction element. This element defines the intent of a Web Service call. However, the first section of the HTTP header, i.e., the POST section, also defines the intent of the service call. Hence, the SOAPAction is generally left blank. The use of this SOAPAction element is still an on-going debate.
- <soapenv:Envelope>: The SOAP envelope (mandatory) defines SOAP message boundaries, i.e., where the SOAP message begins and ends. The envelope usually contains the following format directives. The <soapenv:encodingStyle> specifies the structure of the SOAP message. The <xmlns:soapenv> specifies the structure of the envelope element of a SOAP message. <xmlns:xsd> and <xmlns:xsi> specify the XML schema instance namespace and XML schema namespace respectively. These namespaces will define tag meaning etc.
- <soapenv:Header>: SOAP header (optional) provides directives for the SOAP processor, there are no rules as to what should be included in this section. Seen as a way to add features such as basic security, transaction management, and payment services to a SOAP message [Hendricks et al., 2002, p 111].
- <soapenv:Body>: The SOAP body (mandatory) contains the actual data such as service name, parameter values and method names.

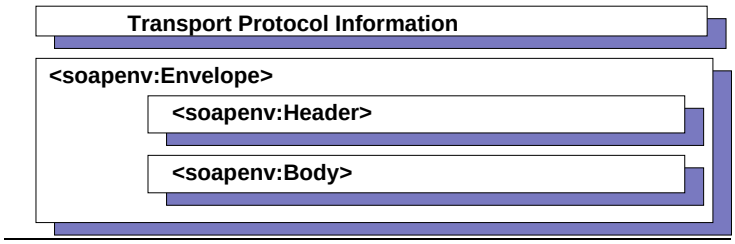

Figure 3 – SOAP Message Structure.

The following is an example of a SOAP request over HTTP. This message sends a complex type (Java Bean) called *Widget* consisting of two strings to the Web Service called *ServiceName* invoking the method *processWidget* on class *WidgetService*.

```
POST /axis/servlet/AxisServlet HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Content-Length: 800

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
            <soapenv:Body>
                  <ns1:processWidget xmlns:ns1="ServiceName">
                        <arg1 href="#id0"/>
                  </ns1:processWidget>
<multiRef id="id0" SOAP-ENC:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns2:Widget" xmlns:ns2="urn:WidgetService">
                        <colour xsi:type="xsd:string">Blue</colour>
                        <size xsi:type="xsd:string">24</size>
                  </multiRef>
            </soapenv:Body>
</soapenv:Envelope>
```

WSDL

The *Web Services Description Language* describes a Web Service in a universally understandable way [Hendricks et al., 2002, p 148]. The WSDL file associated with a Web Services describes where the service is located, what operations are available, how to invoke those operations (parameters), and what transport protocol to use. WSDL is based on XML to ensure interoperability between platforms. An example is a Java Web Service exposing a WSDL file describing itself; a C++ (or any other language) client that is XML/WSDL aware can parse and understand this document and hence understand how to invoke the Web Service.

A WSDL file can have seven basic sections:

- <definitions>: Like the SOAP envelope, this delimits the beginning and end of a WSDL file. XML namespaces are also defined. These include namespaces to define the WSDL framework, WSDL SOAP binding, and the XML schemas.
- <types>: The data types used.
- <message>: The request and response messages exchanged.
- <operation>: Defines the actual methods to be invoked and the invocation style to use. This element contains the <input> and <output> tags which define how the request and response messages are to be encoded.
- <portType>: Collection of related operations.
- <binding>: Defines the communication protocol to be used, e.g., SOAP over HTTP.
- <service><port>: Location of Web Services, i.e., the endpoint Uniform Resource Indicator (URI).

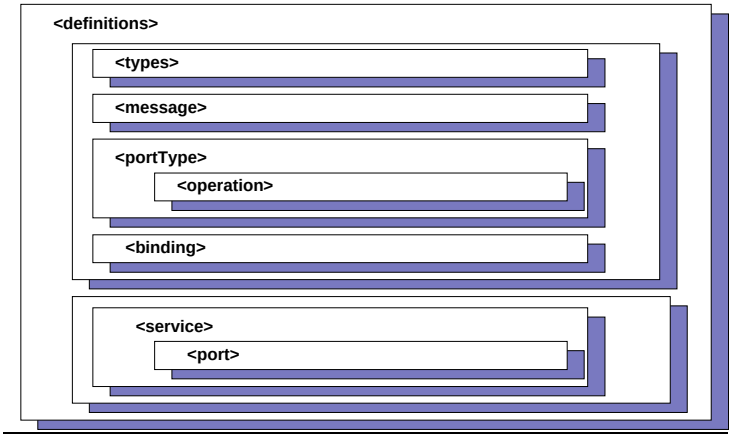

Figure 4 – WSDL Message Structure

The following is an example of a WSDL file. The request parameters consist of an integer and a string. The response is an integer. There is one operation available: *reserve*, which takes the two parameters as input. The invocation style will be *RPC* over *HTTP*. The service is accessible through the endpoint: *http://127.0.0.1:8101/compass/services/Reservation*.

```
      <?xml version="1.0" encoding="UTF-8"?>
      <wsdl:definitions
targetNamespace="http://127.0.0.1:8101/compass/services/Reservation"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:intf="http://127.0.0.1:8101/compass/services/Reservation"
      xmlns:impl="http://127.0.0.1:8101/compass/services/Reservation-
impl" xmlns:SOAP-  ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
            <wsdl:message name="reserveRequest">
                  <wsdl:part name="in0" type="xsd:int"/>
                  <wsdl:part name="in1" type="xsd:string"/>
            </wsdl:message>
            <wsdl:message name="reserveResponse">
                  <wsdl:part name="return" type="xsd:int"/>
            </wsdl:message>
            <wsdl:portType name="Reservation">
            <wsdl:operation name="reserve" parameterOrder="in0 in1">
      <wsdl:input message="intf:reserveRequest"/>
      <wsdl:output message="intf:reserveResponse"/>
            </wsdl:operation>
            </wsdl:portType>
            <wsdl:binding name="ReservationSoapBinding"
                        type="intf:Reservation">
            <wsdlsoap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
            <wsdl:operation name="reserve">
      <wsdlsoap:operation soapAction="" style="rpc"/>
      <wsdl:input>
            <wsdlsoap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://127.0.0.1:8101/compass/services/Reservation"/>
      </wsdl:input>
      <wsdl:output>
            <wsdlsoap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://127.0.0.1:8101/compass/services/Reservation"/>
      </wsdl:output>
            </wsdl:operation>
```

```
        </wsdl:binding>
        <wsdl:service name="ReservationService">
        <wsdl:port name="Reservation"
                    binding="intf:ReservationSoapBinding">
<wsdlsoap:address
location="http://127.0.0.1:8101/compass/services/Reservation"/>
        </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

UDDI

The *Universal Description Discovery and Integration* specification provides a framework for publishing Web Services and for the discovery and consumption of those services by interested parties [Hendricks et al., 2002, pp 151-193]. The UDDI specification defines a global registry structure for holding XML based data. UDDI is not designed specifically with Web Services in mind; it is also a registry for general company information. Note also that while UDDI is the current leader in XML based registries, electronic business XML (ebXML) is fast gaining ground. This is a superset of the UDDI specification and includes more features such as Business Process Management. A provider can publish a Web Services to a registry by providing business information such as the company name and a technical specification of the Web Service(s) provided including the location of a defining WSDL file. A Web Service requester can then discover a Web Service based on search criteria.

The publish and discovery process can be done manually or using the Java API for XML Registries (JAXR) developed by Sun. Registry's can be accessed manually, e.g., a website, (IBM's test registry: http://www.ibm.com/services/uddi/testregistry/protect/registry.html), or using some other proprietary tool. JAXR provides programmatic interfaces through which access to XML based registries are possible. Private registries can be used and are useful for intranets and local corporate networks where Web Services are not being made globally available.

The following is an example of the information contained within a SOAP message to query a registry and the returned result. The query can be created and executed using the JAXR API. For brevity, some of the result information and the SOAP message specifics have been omitted. The query on the UDDI registry is done on business name 'Microsoft'. This returns an XML representation of all the information contained about Microsoft in the particular registry. While the query given in this instance is find_business, other queries can be given such as find_service and find_relatedBusiness.

Query:

```
<uddi:find_business generic="1.0" xmlns="urn:uddi-org:api">
                <uddi:name>Microsoft</uddi:name>
</uddi:find_business>
```

Result:

```
<businessList generic="1.0 operator="Microsoft Corporation"
truncated="false" xmlns="urn:uddi-org:api">
  <businessInfos>
    <businessInfo businessKey="0076B468-EB27-42E5-AC09-
9955CFF462A3">
            <name>Microsoft Corporation</name>
            <description xml:lang="en">
                          . . .
            </description>
            <serviceInfos>
                    <serviceInfo
                      businessKey="0076B468-EB27-42E5-AC09-99"
                      serviceKey="1FFE1F71-2AF3-45FB-B788-0A4">
                                  <name>. . .</name>
                            </serviceInfo>
                    </serviceInfos>
            </businessInfo>
        </businessInfos>
 </businessList>
```

The structure of the data held within a UDDI registry consists of five elements [Chappell & Jewell, 2002, pp 104-106]:

- <businessEntity>: The items contained within this element describe business information such as name and address.
- <businessService>: Describes the services offered by the business.
- <bindingTemplate>: Contains pointer to technical descriptions of services, their associated URL, and possibly a textual description of the service.
- <tModel>: Contains the specifics of how to interact with a Web Service including a pointer to the WSDL file.
- <publisherAssertion>: Outlines business-to-business relationships.

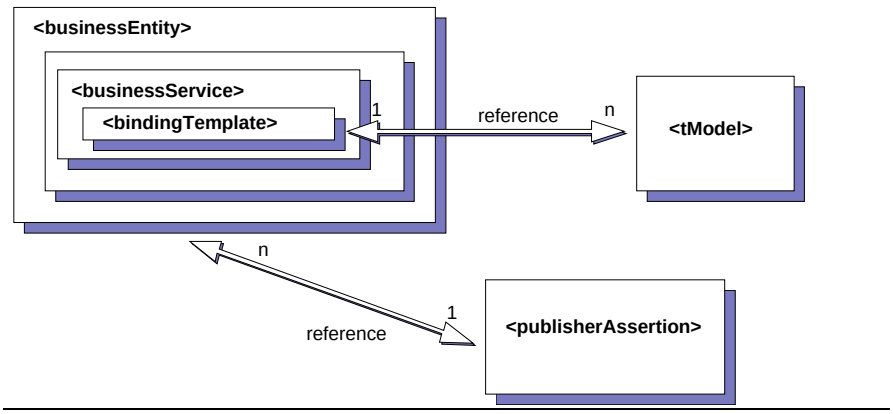

Figure 5 – UDDI Data Structure.

Clients

Currently there are two workable ways for a client wishing to use a Web Service to invoke that service.

Local stub

This involves using a tool to generate proxy stubs [Chappell & Jewell, 2002, p 166]. The tool typically reads a WSDL file and outputs the required proxy/stub files for interaction with the Web Service. This is probably the most common and convenient method used. Almost all vendors will provide such tools with their Web Services toolkits.

The use of stubs allows us to interact with a remote object, through the stub, as if they were local to our runtime. The stubs take care of details such as connecting to the remote machine and data types. Essentially the stub class would act as the Web Service and we can just call methods directly on this class. Using the same tools, skeleton files can also be created for use on the server side. Example tools include:

- proxygen: IBM
- xrpcc: Sun
- wsdl2java: Apache (AXIS)

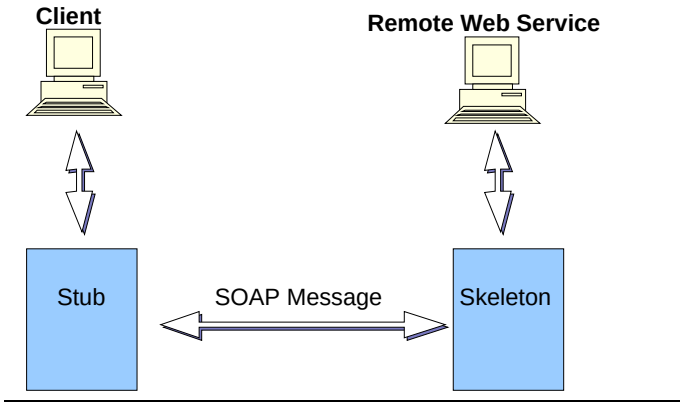

Figure 6 – Stub File Web Service Interaction.

Dynamic Invocation Interface (DII)

DII uses the *javax.xml.rpc.Call* object, which is part of the JAX-RPC, to build up parameters to be sent to a web service [Chappell & Jewell, 2002, pp 166-171]. Method name, method parameters, service endpoint, and return type are all set in the object and an invocation takes place. Generic objects are passed between the client and the servers and cast to the more specific objects on arrival. This is less efficient than the local stub method which deals with

actual objects. Clients are more difficult to code, debug, and test. However it is useful for one way RPC, for when services are discovered dynamically, or where stub creation tools may not be available.

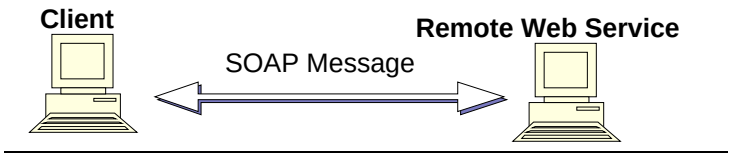

Figure 7 – Dynamic Web Service Invocation.

Security

One of the main concerns with any application made available over a network is security. There are many efforts aimed at developing security standards for various aspects of Web Services. However it should be noted that these standards are immature and clear standards for securing web standards are still a thing of the future.

Concern

Web Services expose many obvious security risks, particularly due to the fact that SOAP using HTTP on port 80 can send requests through corporate firewalls. Hence many traditional security measures are bypassed.

Solutions

Current efforts are aimed at standardising the way in which Web Services will be secured. These efforts include SOAP Digital Signature and XML Encryption. However these standards have either been released recently or are still at the specification stage meaning vendor support may not be realised for some time to come. For example, the IBM Security Toolkit provides an implementation of the XML Encryption standard, but this is just an experimental reference implementation.

Web Services security can be broken down into two distinct areas:

1. Back end web application security: can leverage existing application security frameworks such as the Java Authentication and Authorisation Service (JAAS).
2. Communication layer security: focuses on the transmission of data and hence incorporates such security mechanisms as data encryption/decryption and transport layer security protocols such as Secure Sockets Layer (SSL).

While organisations such as W3C and the Organisation for the Advancement of structured Information Standards (OASIS) work on Web Service security standards, Web Services will be secured using traditional transport layer protocols such as SHTTP and SSL. Also, although SOAP can navigate through firewalls, a certain level of intelligence can be built into firewalls to check the content of the SOAP message and do authentication/authorisation based on that.

For the big players with big budgets, packages exist that offer Web Services security 'frameworks' providing a patchwork of security features specifically tweaked for use with Web Services. For example, Quadrasis has an Enterprise Application Security Integration system called EASI Security Unifier which does just that. However, these proprietary security systems are often expensive. Some organisations will require such stringent identification, authentication, authorisation, integrity, privacy and non-repudiation checks for securing critical data, others will either carefully review what their Web Services expose, or wait for security standards definitions to advance.

Future

Web Services are the way forward for many organisations wanting to expose functionality to a wider audience over a network. Through its standards based approach it will receive widespread buy-in from vendors and developers. However, in its current immature state, it has its flaws. Many of the security standards are still at specification stage.

OASIS has formed a technical discussion group to open up the floor to the *WS-Security* standard as a trusted means for applying security to Web Services. First published in April 2002 as part of a working partnership between Microsoft, IBM, and VeriSign, the WS-Security specification defines a standard set of SOAP extensions, or message headers, which can be used to set security technologies such as encryption and digital signatures, for instance, onto Web Services applications. The first meeting of the technical committee was held during the first week of September.

Liberty Alliance is another movement towards defining web application security. As outlined on the Liberty Alliance Project website, the objectives of the project are to:

- Develop specifications that enable service providers to protect consumer privacy.

- Provide an open single sign-on specification that includes federated authentication from multiple providers operating independently.
- Enable organisations to control, maintain and enhance relationships.
- Create a network identity infrastructure that supports all current and emerging network access devices.

Bloor Research [July 2002], identifies seven outstanding issues that need to be addressed before Web Services can be considered an enterprise class distributed systems architecture. These are: security/privacy, messaging/routing, quality-of-service/reliability, transaction processing, management, performance, interoperability. While these are stumbling blocks for the advancement of Web Services, generally there exist vendor products and solutions to address these problems [Bloor Research, 2002, p11].

Also on the horizon is the concept of 'smart' Web Services. These are services that can maintain a shared context, have multinet capabilities and have quality of service metrics. A shared context involves understanding under what conditions it was invoked. Multinet capabilities relate to a Web Services ability to receive requests from multiple device types, e.g., PDAs, mobile phones, and so on. Quality of service metrics relates to the reliability of communicated information; this encroaches upon the territory of security and reliable messaging [Hendricks et al., 2002, p 471].

Research

As Web Services are still in such an early stage of development, producing real-world applications of this technology is a current challenge within the IT community. Web Services opens up many areas of possible research including streamlined integration and web enabling legacy applications. Industries that have traditionally stayed away from technological advancement may be convinced to adopt Web Services as a means of simplifying many paper driven processes and legacy systems integration. For example, sections of the financial services sector will look to reduce cost by automating as many of their business processes as possible. Web Services are the ideal technology to allow businesses to provide services to outside parties thus delegating work elsewhere resulting in reduced operating costs. Web Services are also the ideal technology for allowing disparate systems integration.

Conclusion

With Web Services we are moving to service oriented development. Among the main principals of Web Service oriented development [Bloomberg, 2002, all] are:

- **Dynamic services replace static components**. Through WSDL the location of particular services can change dynamically without disruption.
- **Service exposure replaces traditional systems integration**. Through WSDL and UDDI multiple services, with their own specific functionality, can be assembled to produce a system.
- **Scalability handled differently**. Registries can be used to hold lists of backup services.
- **Platform irrelevance**. Theoretically, because of SOAP standardisation, disparate platforms will interact seamlessly.
- **Federated application development**. Applications will be composed of several modular Web Services components that each provides a different piece of functionality.

For example, with the Federation Model we move towards application development that incorporates functionality aspects from multiple organisations within a group. This could open up numerous opportunities for companies to develop applications at a fraction of the effort and cost.

The shift towards such a development environment comes about because of the standards outlined within this paper. The SOAP specification forms the basis for interoperable communication. The WSDL defines interoperable Web Service descriptions. The UDDI (and other registry implementations) specification defines Web Service publishing and discovery.

These standards are currently well defined but they are still advancing. It is up to vendors to keep pace with these developments to produce suitable APIs to ensure Web Services continue their evolution. However, for now, there are also stumbling blocks to using Web Services such as performance and security, which must be addressed.

References

Bloomberg (2002). Bloomberg, J., The Seven Principles of Service-Oriented Development. Retrieved August 26, 2002, from XML & Web Services Magazine website
http://www.fawcette.com/xmlmag/2002_08/magazine/focus/jbloomberg/default.asp
Bloor Research (2002). Web Service Gotchas. Retrieved August 30, 2002 from IBM website
ftp://www6.software.ibm.com/software/developer/library/wsgotchadoc.pdf

Chappell & Jewell (2002). Chappell, D.A., & Jewell, T., Java Web Services. O'Reilly
De Jong (2002). De Jong, I., Web Services/SOAP and CORBA. Retrieved September 2, 2002, from http://www.xs4all.nl/~irmen/http://www.xs4all.nl/~irmen/comp/CORBA vs SOAP.doc
Hendricks et al. (2002). Hendricks, M., Galbraith, B., Irani, R., Milbery, J., Modi, T., Tost, A., Toussaint, A., Basha, J., Cable, S., Professional Java Web Services. Wrox
Kao (2001). Kao, J., Developer's Guide to Building XML-Based Web Services with the Java 2 Platform Enterprise Edition. Retrieved August 7, 2002, from TheServerSide.com website http://www.theserverside.com/resources/article.jsp?l=WebServices-Dev-Guide
Monson-Haefel (2002). Monson-Haefel, R., EJB2.1 Web Services: Part 1. Retrieved August 20, 2002, from TheServerSide.com website http://www.theserverside.com/resources/article.jsp?l=MonsonHaefel-Column2
Security: ebXML, SAML, XACML, WS-Security http://www.oasis-open.org/
Security: Application Security Company http://www.quadrasis.com/
Security: Liberty Alliance Security Project http://www.projectliberty.org/
SOAP 1.1 http://www.w3.org/TR/2000/NOTE-SOAP-20000508
SOAP 1.2 Messaging Framework http://www.w3.org/TR/2002/WD-soap12-part1-20020626
SOAP 1.2 Adjuncts http://www.w3.org/TR/2002/WD-soap12-part2-20020626
SOAP-DSIG, XML Encryption, XML Signature, XKMS http://www.w3.org/
SOAP Interoperability Tests http://www.apache.org/~rubys/ApacheClientInterop.html
WSDL 1.1 http://www.w3.org/TR/2001/NOTE-wsdl-20010315
UDDI 3.0 http://uddi.org/uddi-v3.00-published-20020719.htm

Appendix A - Web Service Standards Status

General

Technology Name	Ve	Release Date
Web Services Description Language (WSDL)	1.1 1.2	W3C Released 15-3-2001 W3C Working Draft 9-7-2002
Simple Object Access Protocol (SOAP)	1.1 1.2	W3C Released 8-5-2000 W3C Working Draft 26-6-2002
Universal Description Discovery & Integration (UDDI)	3.0	UDDI.org Released 19-7-2002

Security

Technology Name	Ver	Release Date
SOAP Digital Signature (SOAP-DSIG)	n/a	W3C Released 6-2-2001
XML Encryption Syntax and Processing	n/a	W3C Candidate Recommendation 2-8-2002
XML Signature Syntax and Processing	n/a	W3C Recommendation 12-2-2002
XML Key Management Specification (XKMS)	n/a	W3C Released 30-3-2002
Security Assertions Markup Language (SAML)	1.0	OASIS Standard Maturity Level 5-11-2002
XML Access Control Markup Language (XACML)	n/a	OASIS Working Draft 15, 12-7-2002
Liberty Alliance Project	1.0	Liberty Alliance 11-7-2002
WS-Security	1.0	IBM, Microsoft and VeriSign released in April 2002. Now taken over by OASIS for discussion.

Appendix B - Glossary

AXIS
: Apache eXtensible Interaction System. An open-source implementation of the SOAP specification. It is a follow on from the Apache SOAP project and represents a complete re-architecture.

ebXML
: electronic business eXtensible Markup Language. Modular suite of specifications that provides standards for message exchanges between businesses.

HTTP
: Hyper Text Transport Protocol.

J2EE
: Java based component oriented enterprise application development environment.

JAXR
: Java API for XML Registries. Provides a standard way to use different kinds of XML registries.

JAX-RPC
: Java API for XML based Remote Procedure Calls. Enables developers to build remote procedure call functionality into SOAP requests.

Liberty Alliance Project
: Single sign-on standard based on SAML which lets users who sign on to one Web Service carry over that authenticated status when moving to other Web sites. A more feature-rich second phase is expected in the near future.

OASIS
: Organisation for the Advancement of Structured Information Standards. OASIS is a non-profit, global consortium that drives the development, convergence and adoption of e-business standards.

SAML
: Secure Assertion Markup Language, Allows organisations to exchange authentication, authorisation, and profile information securely with their partners. Currently two Java specification requests (JSR 115 and JSR 155), which are associated with SAML. SAML not included as part of WS-Security which may prove a stumbling block.

SOAP
: Lightweight standard that facilitates the transport of XML data over an underlying network protocol.

SOAP-DSIG
: SOAP digital signature specifies the syntax and processing rules of a SOAP header entry to carry digital signature information.

UDDI
: Provides a framework for publishing Web Services and for the discovery & consumption of those services by interested parties.

WS-Security
: Extensions to the SOAP designed to make Web Services applications confidential and secure.

WSDL
: Provides a way to describe a Web Service in a universally understandable way.

XACML
: XML Access Control Markup Language. Expected to complement SAML. Implemented in IBM's XML Security Suite.

XKMS
: XML Key Management (submitted to the w3c in March 2001). Protocols for distributing and registering public keys, resolution/retrieval of public key, and association and retrieval of attributes in the form of 'trust assertions' with public keys.

XML
: eXtensible Markup Language. Subset of the Standardised General Markup Language.

XML Encryption
: Process for encrypting and decrypting XML documents. JSR 106 is a community request for the XML Digital encryption API. Part of WS-Security initiative.

XML Signature
: XML compliant syntax used for representing the signature of Web Services and portions of protocol messages and procedures for computing and verifying such signatures.