

Dunelm

Services Limited



Consultants in Applied Research & Development for
Information & Communications Technology



Session 6: Core Technology ***Service Oriented Architecture:*** ***Preparing Your Business &*** ***IT Products***

2009 © Dunelm Services Limited

PUBLICATION INFORMATION

The publication history for this booklet is:

<i>Version</i>	<i>Date</i>	<i>Summary of Amendments</i>
A	June, 2009	The first release of this booklet.

License

This material is made available under a Creative Commons License. This License, termed ‘UK: England and Wales Attribution version 2.0’, enables anyone to copy, distribute, display and perform the work, and to make derivative works. The only condition is that attribution credit must be made to the original author; details for how this credit must be given are available on the Dunelm web-site at www.dunelm.com/projects and following the links for the Digital 20/20 project. Further details on the Creative Commons License are at: creativecommons.org/licenses/by/2.0/uk/



EXECUTIVE SUMMARY

This is the sixth and final session in the series on Service-oriented Architecture (SOA). Unlike the previous sessions, this one is focused on the technology that is used to turn SOA from a business tool into commercial advantage. A lot of the discussion that follows concerns Web Services. Web Services is one technology that can be used to realise SOA but it is important to remember that, in principle, it is NOT the only technology. However, Web Services is the only technology that is currently available to realise SOA. As far as this discussion is concerned, the newly touted Web-oriented Architecture is just a flavour of Web Services.

Of all the material covered in the six sessions, it is the content in this session which is most susceptible to change. Across the World, there is a large amount of development effort in making it easier to implement Web Services. There are a lot of relevant standards being developed and in many cases these are undergoing significant revision as we gain more implementation experience. Likewise, the tool-sets that make it possible to create a Web Service solution are also being changed and improved. Therefore, it is important to spend some time ensuring that any of the following recommendations still reflect the agreed best practices.

TABLE OF CONTENTS

Publication Information	2
Executive Summary	3
1. Introduction	5
2. Core Web Technologies	8
3. SOAP Messages.....	13
4. Web Service Description	16
5. WS-I Organisation	20
6. Deploying Web Services.....	24
7. WS-* Standards	27
8. Web Services Stack.....	30
9. APIs, Protocols & Interfaces	33
10. Interoperability & Innovation	38
11. Combining Services.....	40
12. Enterprise Service Bus.....	42
13. Over-the-Horizon.....	47
14. In Conclusion.....	50
Appendix A – Bibliography	53
Appendix B – Acronyms.....	55
Index	58

1. INTRODUCTION

Throughout the previous five sessions the significance of the implementation technology for SOA has been down played in favour of the importance of understanding and improving the business process. However, the implementation of a system based upon SOA must use technology that can reflect the characteristics of SOA e.g. loose service coupling, etc. Poor implementation will defeat the key SOA system properties such as flexibility i.e. the ability to create new business processes by novel reuse of established services. In Session 1 we briefly discussed the 2008 Gartner Technology Hype Cycle in which the use of basic Web Services was identified as established practice. In this Session we are going to take a closer look at the technology used in basic Web Services and discuss how it can be used to support SOA. We'll also look at the new developments in Web Services that are needed to be able to realise many of the functional characteristics required in SOA-based systems.

Within the context of this series on SOA there are three learning objectives for this, Session 6.

The first is to *explain what implementation technologies should be adopted in the short and mid-term*. The short term is assumed to be 1-2 years whereas the mid-term is the next 2-5 years. Given the pace of technology development and the inherent uncertainty in predicting technology changes there is little benefit in thinking beyond five years. Indeed, for most of us, any return on investment must be achieved within the short term

and so the relevance of long term technology developments are heavily coloured by short term experiences. Fortunately, new and significantly different technologies are usually accompanied by products that enable controlled migration from the old to the new solutions. We live in a World of rapidly changing technology and the marketplace has recognised that supporting change is essential for survival and long-term customer loyalty.

The second is to *explain the issues and concerns to be considered when looking to adopt Web Service solutions*. When the first Web Services solutions were being deployed there were few alternative technologies to consider but the tools for these technologies were primitive. For example, in Visual Studio (from Microsoft) the Web Services code generation tools were additional downloads and not part of the core distribution. Also, there were, and still are, problems when ensuring interoperability between different platforms and in particular the .NET and J2EE Worlds. These issues are compounded by the fact that we now have different versions for many of these technologies and so there are significant compatibility and interoperability issues to be addressed, particularly in enterprise-wide deployments. Throughout this session these issues of compatibility and interoperability will be identified and the recommended best practices to resolve them will be suggested.

The third is to *explain how technical interoperability must be addressed to support innovation*. Recall that one of the objectives of SOA is to enable a plug-and-play approach for services based upon the reuse and combination of services. At

a technical level interoperability is essential. Interoperability means that two systems can exchange the required information without new implementation work irrespective of which platforms are used to support the systems. This means that the systems must use the appropriate service capabilities defined in terms of both the interface (or application programming interface) and the protocol (or the Web Service). Interoperability must be addressed at many different levels. The semantics of the data in systems must be compatible otherwise syntactic interoperability becomes irrelevant. For example, if two systems use the concept of a Group differently, even though they use the same name, then exchanging meaningful data becomes almost impossible. However, once interoperability for each service has been established then new business processes can be created by innovative combination of those services.

2. CORE WEB TECHNOLOGIES

The are the minimal set needed to provide service-oriented computing. The core defines the mechanism for passing messages between a client (service consumer) and a service (service provider), and the way to describe services (the operations/messages they support). Over time two alternative approaches have been developed and these now have their own advocate and adoption communities. In both cases the Extensible Mark-up Language, more commonly referred to as XML, is the preferred data representation format and HTTP is used as the underlying Internet protocol: remember that this is all about extending the way traditional Web servers operate. The two communities then split into the SOAP and REST communities. The Representational State Transfer (REST) community abhors the use of SOAP, and thus the related specifications, claiming it is overly complex and changes the nature of true Web-based interactions. It is the SOAP approach that is usually assumed when the term Web Services is applied.

Figure 1 shows the key technology components of the SOAP and REST approaches to Web Services.

XML Schema [XML, 04] is one of the XML-based languages used to describe the structure, semantics and constraints of XML documents. Most Web Services specifications are defined by languages and documents expressed in XML Schema. The advantage of XML over HyperText Mark-up Language (HTML) is that user-defined document formats are possible using XML.

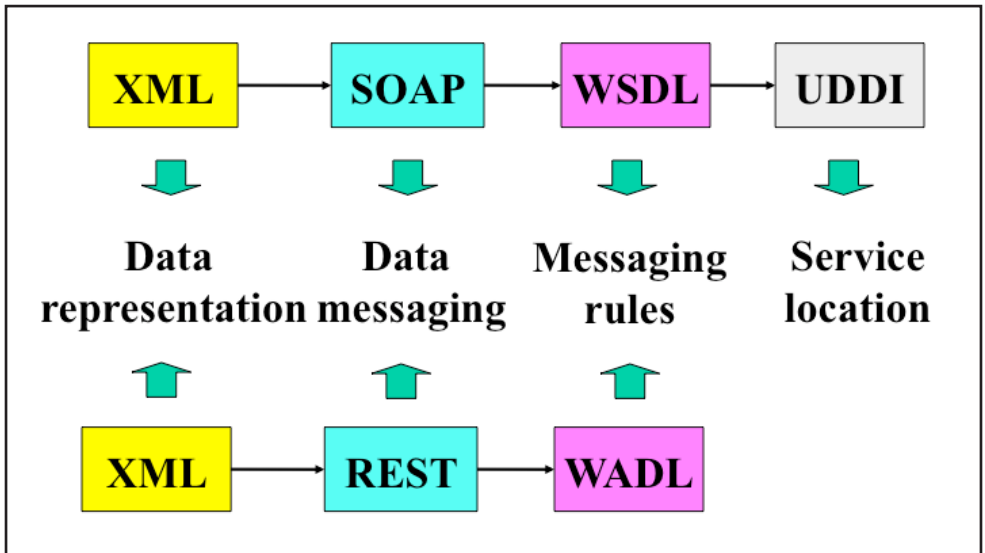


Figure 1 Core technologies for web services.

An XML Schema Definition (XSD) file is created and this is used to validate the XML instance documents. These XML instance documents can then be computer-processed with each structure in the XSD subject to the relevant processing. Another difference between XML and HTML is that XML has no intrinsic presentation information and so the presentation of XML data can be determined by the processing context and is not fixed at the time of authoring.

For Web Services the data messaging uses SOAP (originally an acronym for Simple Object Access Protocol but this usage has lapsed because it is not simple nor an access protocol). SOAP is a protocol for the exchange of messages described in XML over a distributed network e.g. the Internet. The SOAP specification [SOAP, 00] defines: (1) the XML format for messages; (2) the

rules that define how to process messages, and (3) a mechanism for defining how to transport messages. SOAP is most commonly used in a request-response messaging pattern e.g. remote procedure call, with messages transported over the HTTP protocol. Several versions of SOAP are deployed but SOAPv1.1 has widest adoption.

The Web Services Description Language (WSDL) is used to describe Web Services in terms of the message exchange patterns. A WSDL service description defines the operations, messages and end points (access points) for the service. Abstract descriptions are bound to actual network protocols and message formats. WSDL is most commonly used to describe document-oriented operations (with document definitions defined in XML Schema and the WSDL files themselves are instances of XML), communicated in SOAP messages over HTTP transport. WSDLv1.1 [WSDL, 01] is the most commonly used version with WSDLv2.0 [WSDL, 07] in early deployment.

For dynamically configured systems it is essential to be able to find a service. The Universal Description, Discovery and Integration (UDDI) specifications define how to establish an (XML) Internet registry of business service listings. UDDI describes the structure of a UDDI registry and the Web Service interface to the registry. UDDI data models are represented in XML and UDDI registries expose service interfaces defined in WSDL, accessible via SOAP web service invocations. UDDI is a complex specification to implement and unfortunately it has very limited adoption. Alternative approaches, such as

Microsoft's WS-Discovery are under development.

The alternative RESTful approach uses the core technologies of XML, HTTP and Uniform Resource Identifiers (URIs). Components and resources are identified through URI labels such that the web infrastructure can map a logical name to an actual hardware endpoint on the Internet that is the target for messages and where processing is undertaken. While this approach to services is technically viable, widely deployed (typically in providing data to browser-based clients), usually the basis for Web 2.0 compositions (mashups), and sometimes presented as sufficient for building Internet applications (with the inclusion of underlying secure transport protocols), it does not fully correspond to SOA.

REST [Fielding, 00] is formally defined as a collection of network architecture principles that can be bound to any collection of appropriate technologies. More commonly, REST (or RESTful) implies the use of the REST principles combined with basic Web protocols and standards. While REST shares the same technology bindings as Plain Old XML (POX), RESTful systems and their resources, component services and intermediaries impose additional architectural constraints to formally conform to REST. POX-based binding that do not impose these constraints are often denoted REST-like. However, in practice, the REST, RESTful, REST-like and POX labels are often used interchangeably.

The Web Application Description Language (WADL) provides an example of formal machine-processable representation

of POX-based and RESTful web services. WADL [WADL, 06] aids in automatic code generation for service interfaces. WADL provides mechanisms to map resources to their XML representations (including XSDs) and data/MIME types, and to map requests and associated errors to their corresponding HTTP request and response codes. WADL is to REST what WSDL is to Web Services.

3. SOAP MESSAGES

A SOAP message has as a structure similar to a letter sent using the traditional postal system. The main difference is that a SOAP message is written in XML. The message itself is contained in the SOAP Envelop, as shown in Figure 2. It is the envelop that is carried between systems using HTTP, or one of several other transport protocols. Inside the envelop are the SOAP Header and the SOAP Body. The SOAP Body contains the actual message itself and remember that this must be in the form of valid XML. The SOAP Header is used to carry important contextual information about the message itself. For example, a message from a service provider could contain information about the status of the requested transaction. More significantly, the Header can contain any number of Header Blocks. Other Web Services specifications define their own Header Blocks and these are used to extend the functionality of SOAP messages. For example, the Web Services specification WS-Addressing is used to provide more control over how instances of services are identified at an end-point and this information is placed in a SOAP Header Block.

The information in a SOAP Body can be used to support Remote Procedure Call (RPC) or Document based payloads. The SOAP specification was originally designed to replace proprietary RPC protocols by allowing calls between systems to exchange XML documents. In hindsight SOAP is a somewhat heavy-approach to RPC and so Document payloads are more commonly used.

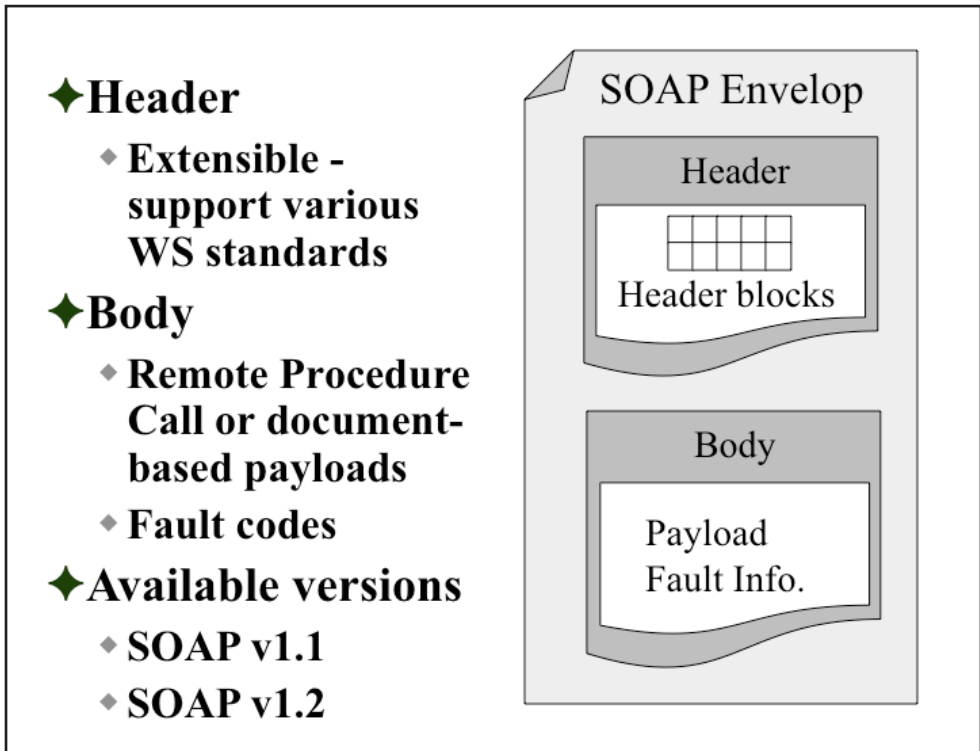


Figure 2 SOAP messages.

This reflects SOAP's bias towards larger payloads, coarser interface operations and reduced message transmission volumes between services. A SOAP Body can also contain Fault Codes that can be used to denote system exception conditions. There is a natural tendency to think of SOAP as an end-to-end protocol; however, SOAP messages can be routed across intermediate SOAP nodes and so a fault could occur at any intermediate node as well as an endpoint.

Two versions of SOAP, 1.1 [SOAP, 00] and 1.2 [SOAP, 07] are commonly deployed. Many of the differences between these two versions are small and result in subtle changes and so beyond this discussion. Those changes of a more substantial nature in SOAP v1.2 are:

- Additional elements following the SOAP ‘Body’ element are forbidden;
- New fault codes with new fault code extension syntax is added;
- The processing of incoming messages at a node is defined;
- Semantics for dealing with messages from different versions of SOAP are added to ensure backwards compatibility;
- SOAP 1.1 defines a single binding to HTTP whereas SOAP 1.2 defines an abstract binding framework. SOAP 1.2 also defines a concrete binding to HTTP and a non-normative email binding.

One of the problems with SOAP messages is that they cannot be used to carry validated non-XML information. Other specifications have been developed to remove this limitation but again there is more than one possible solution and this has caused interoperability issues.

4. WEB SERVICE DESCRIPTION

Web Services are described using the Web Services Description Language (WSDL). A WSDL description is expressed in XML in a WSDL file. A WSDL document defines services as collections of endpoints (WSDLv2.0), or ports (WSDLv1.1). In WSDL, the abstract definition of endpoints/ports and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged; interfaces (WSDLv2.) port types (WSDLv1.1) that are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. An endpoint or port is defined by associating a network address with a reusable binding and a collection of ports defines a service.

A WSDL document, as shown in Figure 3, uses the following elements in the definition of network services:

- Types – a container for data type definitions using some type system (XSD is widely used but any form of XML-based data format can be used e.g. RDF);
- Message – an abstract, typed definition of the data being communicated. In general there are many parts to a message and messages are either sent to a service provider (in) or sent from a service provider (out);
- Operation – an abstract description of an action supported

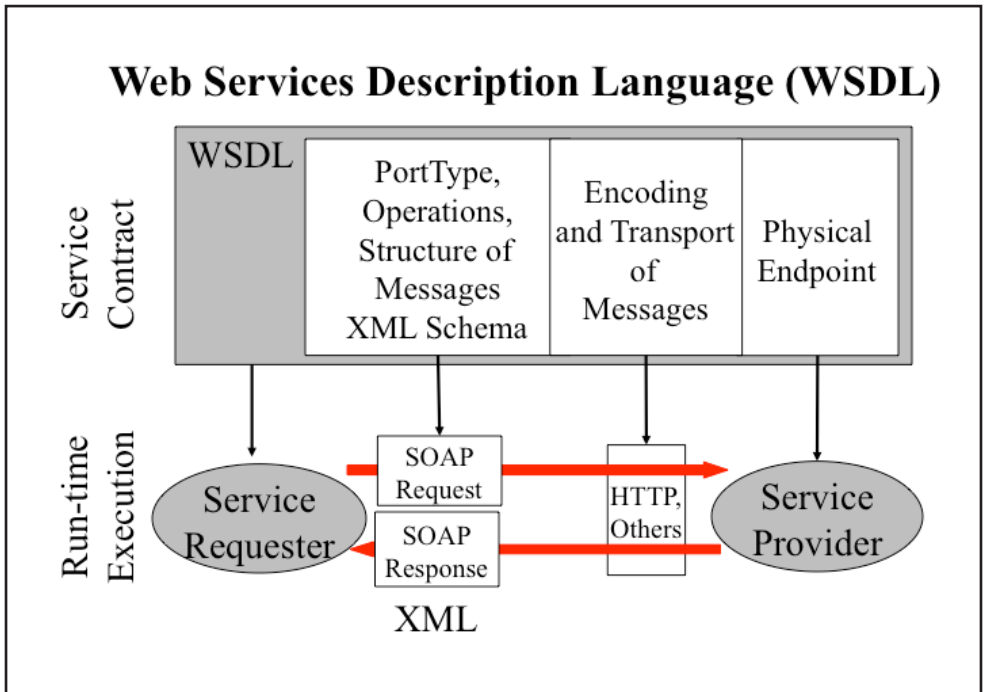


Figure 3 The usage of WSDL.

by the service. In general there are many operations each associated with one or two messages. These operations define the external functionality exposed by a service;

- **Port Type or Interface (WSDv2.0)** – an abstract set of operations supported by one or more endpoints. There may be more than one Port Type defined and each can have any number of operations;
- **Binding** – a concrete protocol and data format specification for a particular port type. There is a separate binding for every concrete protocol that is available. For example SOAP

is one such concrete protocol;

- Port or Endpoint (WSDv2.0) – a single endpoint defined as a combination of a binding and a network address. More than one port or endpoint may be defined for each service;
- Service – a collection of related endpoints. More than one service can be described in the WSDL file.

A WSDL file can contain the definition of more than one service and each service can have several ports to which any combination of operations can be mapped. A WSDL description can be composed of one or more physical files. There are four categories by which the WSDL file can be realised using physical files (we'll assume that the data is formatted using XSDs):

- Single combined WSDL/XSD file – this is the recommended approach because many code generation tools cannot handle the split file combinations;
- Separate WSDL and XSD file – the type descriptions are contained in an external XSD file;
- Multiple WSDL and single XSD file – the WSDL file is split into its abstract parts and the service specific parts. The latter contains the mapping to the specific binding technology e.g. SOAP;
- Multiple WSDL and multiple XSD files.

An important feature of WSDL is that different message

exchange patterns can be defined. In WSDLv1.1 four message patterns are permitted:

- One way – a single message (SOAP Request) to the service provider;
- Notification – a single message (SOAP Request) from the service provider;
- Solicit Response – a single message (SOAP Request) from the service provider followed by a single response back (a SOAP Response) to the service provider from the service consumer;
- Request-Response – a single message (SOAP Request) to the service provider with a single response (SOAP Response) message returned to the service consumer.

In WSDLv2.0 eight message patterns are supported. These new patterns include support for robust and optional message exchanges.

Creating your service descriptions in WSDL is essential. This allows you to use code generation tools to do most of the work of implementing the services. It is also the standardised way of publishing the Web Services interface to your service.

5. WS-I ORGANISATION

The Web Services Interoperability Organization (WS-I) is an open industry consortium chartered to establish Best Practices for Web Services interoperability, for selected groups of Web Services standards, across platforms, operating systems and programming languages. WS-I comprises a diverse community of Web Services leaders from a wide range of companies and standards development organizations; members include Accenture, Ford, Fujitsu, Hewlett-Packard, IBM, Intel, Microsoft, NEC, Oracle, SAP, Toshiba. The point for listing those members is to show that this is a serious organization with broad industrial support to ensure that WS-I produces useful material that is readily adopted.

WS-I committees and working groups create Profiles and supporting Testing Tools based on best practices for selected sets of Web Services standards. The Profiles and Testing Tools are available for use by the Web Services community to aid in developing and deploying interoperable Web Services. Further information on WS-I is available at: <http://www.ws-i.org>.

So, what does WS-I produce that is useful to the rest of us. The primary deliverables are Profiles that provide implementation guidelines for how related Web Services specifications should be used together to achieve interoperability. To date, WS-I has finalised the Basic Profile, Attachments Profile and Simple SOAP Binding Profile. Work on a Basic Security Profile is currently underway along with revisions of the Basic Profile.

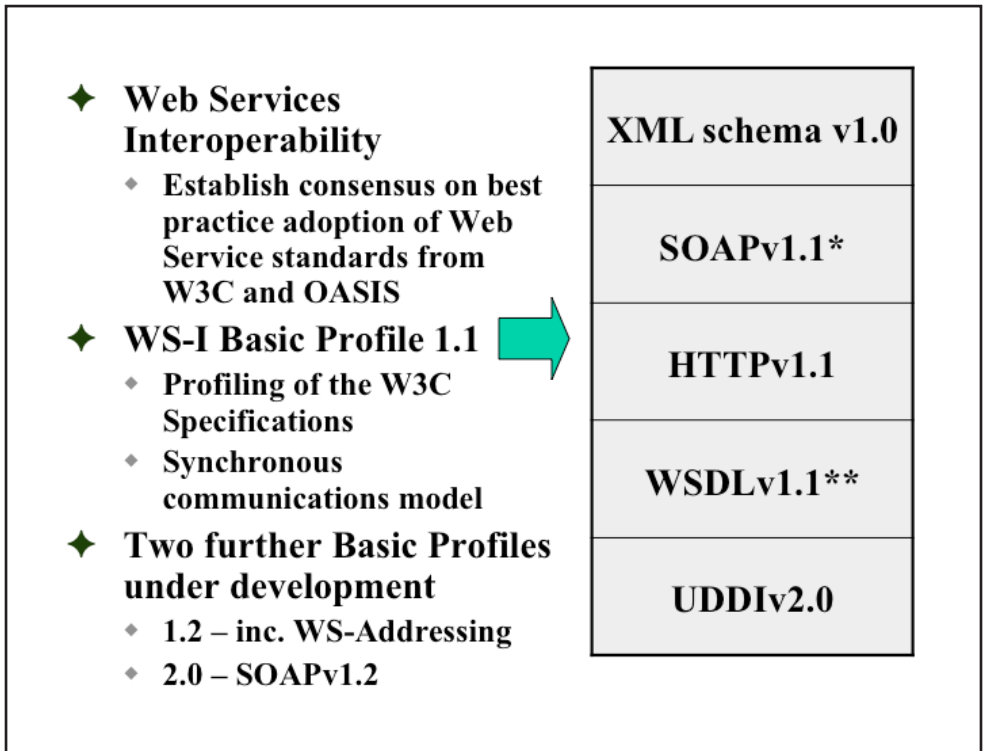


Figure 4 The WS-I Basic Profile.

So, what is a Profile and why is it useful? A Profile is a refinement of one or more standards to meet the needs of a specific community; usually the aim is to improve interoperability. Therefore, a Profile should be defined by the community itself. But why is a Profile needed, shouldn't the standards be sufficient themselves? Most technology standards contain optional features and do not **impose** a best practice. Instead the standards consist of many forms of compromise and facilitate practice. Also, a Profile will, in general, cover more than one standard and so will describe how the standards should

be combined in the most effective way. So, a Profile is used to agree BEST PRACTICE for the community creating it.

The core deliverable from WS-I is the Basic Profile. Version 1.1 (second edition), the latest version of the WS-I Basic Profile was released in April 2006 [WSI, 06]. Figure 4 is a schematic representation of the core standards from which this profile is derived. This recommends the use of SOAPv1.1 messaging, over HTTPv1.1 to exchange XML data for Web Services described using WSDLv1.1 with the Universal Description Discovery and Integration (UDDI) v2.0 used for service publication and discovery. Strictly speaking, the detailed conformance statements for the SOAP messaging are contained in an accompanying Profile called the WS-I Simple SOAP Binding Profile v1.0. The combination of the Basic Profile v1.1 and the Simple SOAP Binding Profile v1.0 has the same coverage as the earlier version, 1.0, WS-I Basic Profile.

One of the characteristics of the Basic Profile is that it makes use of well-established versions of the core technologies. So SOAPv1.1 is used as opposed to SOAPv1.2 and WSDLv1.1 is used as opposed to WSDLv2.0. In both cases, the tools available to developers for the older versions are more mature and reliable making it easier to establish guaranteed interoperability; the word guaranteed is very important. A set of test tools are also available, supplied as a flavour of open source, and these can be used to inspect the SOAP messages being exchanged to make sure that they conform to the Basic Profile.

In 2008, ISO/IEC (the international standards organization)

released the WS-I Basic Profile v1.1 and the accompanying WS-I Simple SOAP Binding Profile v1.0 as draft standards; this provides further credibility to work of the WS-I. WS-I is currently working on versions 1.2 and 2.0 of the Basic Profile. Version 1.2 adds more Web Service standards to the Basic Profile such as WS-Addressing and Message Transmission Optimization Mechanism (MTOM) to addresses asynchronous communications. Version 2.0 replaces the use of SOAPv1.1 with SOAPv1.2, adds support for MTOM and the XML-binary Optimised Packaging (XOP) to replace the use of SOAP with Attachments. Draft releases of version 1.2 and 2.0 are already available for review.

Whereas there are some very real advantages in adopting the WS-I profiles, further refinement of the profile can be undertaken. For example, UDDI has very little adoption and SOAP with Attachments (this is used to send non-XML based data with the SOAP messages and was the subject of its own WS-I profile) is being replaced by MTOM. However, I stress that starting with the WS-I Profiles is established best practice.

6. DEPLOYING WEB SERVICES

Now that the basic set of Web Services technologies have been described we need to address how these are used to implement and deploy real services. Like all software engineering projects the first step is to agree the functional requirements of the service. The service needs to be described in terms of its external interface (this is used to generate the corresponding interfaces and protocols) and the internal functional operation. The functional description includes defining the data structures and the functional behaviour. The internal data structures do not need to be expressed in the same form as the data structures for the interface. For example, a Web Service based interface will use XML for the expression of the data structures whereas many internal data structures will be realised using a database. Remember the interface to the service is used to separate the local realisation of the service from its interoperability implementation.

We'll assume that an organization knows how define the functionality for a software system using an appropriate methodology. Instead we will focus on the interface and interoperability development. Once the interface definition has been created (some organizations call the purely abstract functional definition the Interface its *Information Model*) its technology realisation, or *Binding*, is defined. In the case of Web Services the binding is expressed in WSDL. It is important that an organisation creates consistent WSDL bindings that ensure interoperability. The best way to achieve this is to use

established best practice recommendations such as the profiles from WS-I. Further profiling can be undertaken to remove those features of WS-I that are irrelevant e.g. the use of UDDI, etc. It is also important to document the WSDL files themselves i.e. use the WSDL documentation commenting features. Many WSDL authoring tools also provide documentation features that will create a HTML description of the WSDL file.

If your Web Service uses HTTP (recall that SOAP can use SMTP, FTP, etc.) then you will need a Web Server to host your service. The Web Server processes the HTTP calls and passes the SOAP requests onto the service implementation. Any corresponding SOAP response is returned using HTTP.

The WSDL contains the description of how to interact with the service. If the service is externally visible, then the WSDL file(s) should be made available. A service repository should be maintained in which all of the WSDL and related files can be stored. In Enterprise systems it is recommended that some repository discovery and search protocol be supported, for example UDDI. Once the WSDL files are available externally, it is possible for other organisations to build service clients. An example of this approach is described in the Session on three cases studies and the Amazon Web Services. If WSDL is not used some other documentation format must be used to supply the description of the service. The advantage of WSDL is that most integration development environments now provide code generation tools that work from a WSDL file and so this simplifies access to your service.

There is one important note of caution. By definition, a WSDL file defines a path through your firewall to the server hosting the service. It is important that external facing services are isolated from internal-only services and systems. Your security architecture must take into account any support of Web Services.

7. WS-* STANDARDS

While the core Web Service specifications are sufficient to achieve basic service interoperability, they leave many details unspecified; business partners must agree on these implementation details. Furthermore, the core specifications are specific to only limited technologies. The broader collection of Web Services specifications, denoted WS-* (pronounced WS Star), define a more comprehensive service architecture and further detail solutions to different interoperability problems. They also uncouple the different behaviours and representations, providing a more comprehensive model for abstracting capabilities from underlying representations. Therefore many of the WS-* specifications are designed to be combined with other specifications to define a complete Web Services or SOA model. Given the range of WS-* specifications and their ongoing development, profiles, such as WS-I, provide guidelines for interoperability across the collection of WS-* specifications.

The WS-* standards of relevance, listed in Figure 5, are:

- MTOM describes features for optimizing the (wire) transport of SOAPv1.2 messages. It defines how parts of the XML message i.e. binary data, may be encoded while still permitting XML processing of the message. It uses XOP to define a processing model for encoding. MTOM also provides an “include” feature designed to enable the mechanism to efficiently transport binary objects and replace other models, such as SOAP with Attachments and MIME for

WS Standard	Description
MTOM	Message Transmission Optimisation Mechanism for sending non-XML data
WS-Addressing	End-point addressing
WS-Security	Security framework for authentication and authorisation
WS-Policy	Attaching policy data to messages
BPEL	Service composition using the Business Process Execution Language

Figure 5 Key WS-* standards.

Web Service. Both MTOM and XOP are parts of the SOAP family of specifications;

- Web Services Addressing (WS-Addressing) [WSA, 06] is an XML-based language used to describe message transport for Web Services. Rather than rely on the properties of the underlying transport protocol in a service call, WS-Addressing provides a standardized way, which is transport protocol independent, to include message addressing within the XML message itself e.g. include HTTP specifics in the SOAP messages. The addressing information includes the service end point and message parameters. In addition to the transport neutral model, the specification includes bindings to SOAP and WSDL;

- Web Services Policy (WS-Policy) is an XML-based language used to describe the policies that apply to a Web Service. Policies define characteristics such as authentication scheme, security, encryption, transport protocol, privacy, quality of service business rules, etc. Policies are defined independent of application and scope, via the Policy Framework, and are “attached” to service end points and other elements such as WSDL service definitions and UDDI service descriptions;
- The Business Process Execution Language (BPEL) is an XML-based language used to describe executable processes formed through the assembly and orchestrated interaction of multiple Web Services supporting business processes. BPEL provides a programming language to describe long-running transactions and sequences across a collection of Web Services (defined in WSDL, interacting through SOAP messages). Business processes defined in BPEL may be executed under a BPEL processing engine, invoking and controlling the constituent services;
- Web Services Security (WS-Security) describes a security model for use in a Web Services environment. The core WS-Security specification defines extensions to SOAP to transmit security credentials and to ensure end point-to-end point message confidentiality and integrity to secure web services. The core specification is independent of security approach and has been profiled for use with PKI (X.509), SAML, Kerberos, username/password pairs and MPEG21 rights expressions.

8. WEB SERVICES STACK

The current (and evolving) Web Services model of SOA is defined using additional interoperability standards, developed primarily by the World Wide Web Consortium (W3C) and Organization for the Advancement of Structured Information Standards (OASIS) and denoted as WS-*. These WS-* standards are combined and layered, providing the Web Services stack. In the WS-* stack, standards provide abstraction layers i.e. the standards are defined using more basic standards and groups of related standards are classified by their function. There are several manifestations of the Web Services stack depending on what characteristics are to be stressed. As shown in Figure 6, one perspective is:

- Internet Core Communications – WS-*, as with the web architecture, uses core Internet standards as its basis (URI naming, HTTP transport, MIME);
- Basic Technologies – XML is the core representational language and is used to define all vocabularies including those that define the data models for domain application data. Core domain-independent XML vocabularies e.g. XML Schema, XPath, are used throughout the WS-* collection of standards to define the other parts of WS-*;
- Messaging – WS-* defines how to describe and send messages between services to support communications, primarily through SOAP and other messaging standards.

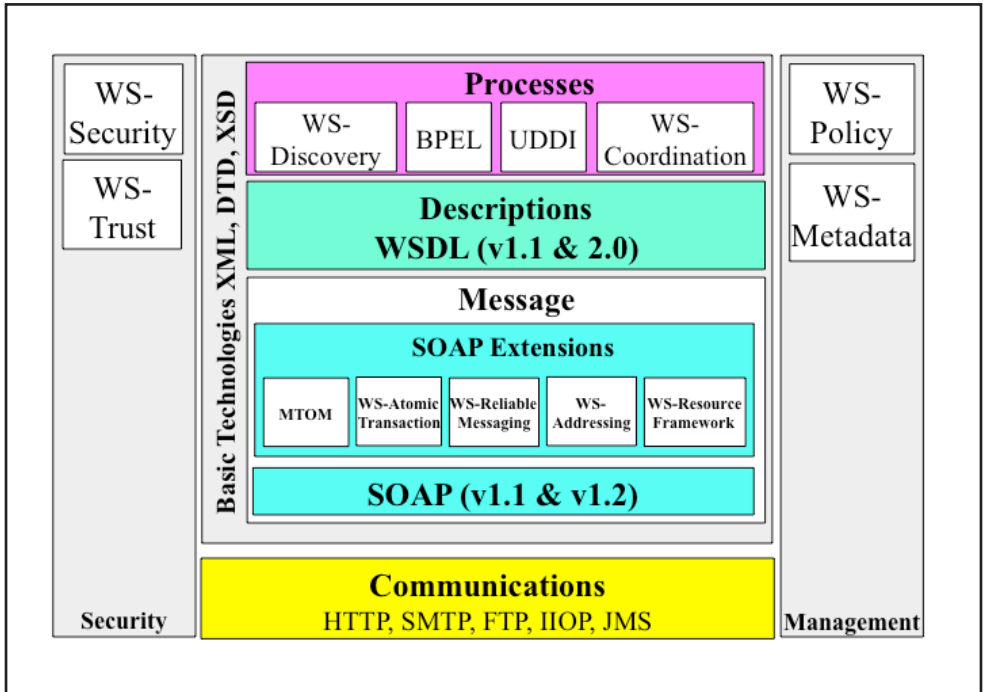


Figure 6 Web Services stack.

The relevant standards are MTOM, WS-Atomic Transaction (to describe the coordination and transactional processing of distributed Web Services), WS-Reliable Messaging (to provide SOAP level guaranteed delivery message patterns), WS-Addressing and WS-Resource Framework (defines how stateful resources are to be supported – Web Services normally being stateless interactions);

- Descriptions – this is the usage of WSDL to describe the services available;
- Business Processes – WS-* business process standards

describe how to model and manage the services within a specific business process or across the business enterprise so that automated systems can coordinate the operation of the various services to meet the business requirements. These standards include the discovery of services (UDDI and WS-Discovery) and the combination of services using BPEL and WS-Coordination;

- Management – service management and quality of service characteristics are also described through machine-readable XML vocabularies. These definitions can be combined with metadata (WS-Metadata) and policy constraints (WS-Policy) to define services, interfaces, management, and their bindings to network end points in deployments;
- Security – a key set of WS-* are those that address security. WS-Security and WS-Trust are the two relevant standards but these need to be profiled to fit the broader security architecture for the systems, within which the service will operate, as a whole.

It is important to stress that this is very early days for such a stack. There is no deployment implementation experience for all but a few of these standards. Also, we have no understanding of the interoperability problems that will be encountered once deployment begins. Therefore, only the bravest of organizations should consider deployment of such a stack.

9. APIs, PROTOCOLS & INTERFACES

Flexible service implementation requires the definition of a clear interface and messaging protocol. Full interoperability requires agreement on both the interface and the protocol. A simple, but important, view is that the interface is a reflection of the business application facing interpretation of the service whereas the protocol is the technical infrastructure facing interpretation of the service. A Service Adapter provides the glue between the Interface and the Protocol. This separation provides flexibility. This flexibility is required because the implementation of a service will have to change if the environment in which it must operate changes. For example, one protocol may be ideal to provide a fast response whereas another may be better suited to provide reliable communications over an error-prone network. The costs of implementation are considerably increased if every change in the protocol requires the interface to be changed and vice-versa. So, the Interface hides the details of the Protocol from the application and the Protocol ensures that the application-specific functionality is supported using a common network infrastructure. The Service Adapter is the means by which the Interface or Protocol can be changed independently of each other; it realises the mapping between the Interface and the Protocol and provides all of the platform-specific implementation details e.g. calls to the host operating system, exception handling, etc.

Established best practice for creating applications is to reuse

components as widely as possible. Such reuse is only possible when the component has a clearly defined interface – termed its Application Programming Interface (API). The same approach is used for services. An application can make use of one or more services and the application is responsible for orchestrating the interaction of the services. A closer consideration of the interface to a service reveals two different roles that have to be supported:

- Data – the normal role of the API. This consists of the set of operation calls that are used to access the functionality of the service;
- Management – the calls that are used to manage the interface as a whole. This includes the configuration of the data interface and may involve dynamic binding of the description of the interface to the appropriate implementation technology;

From the perspective of a protocol for a service the main issue is to define the set of messages and the corresponding messaging sequence. Three standard generic communications messaging models must be considered:

- Fire-and-forget – this is when a message is sent and no form of response or confirmation is to be returned (this also known as datagram or send-and-pray). For Web Services this is one of the native message patterns supported by WSDL;
- Synchronous – this is a request/response message exchange in which the service consumer is blocked until the response message from the service provider is received. Again, for

Web Services this is one of the native message patterns supported by WSDL;

- Asynchronous – this is a request/acknowledgement and response/acknowledgement message choreography during which the service consumer is unblocked i.e. more than one request message can be issued before a response message is received. This is handled by creating two WSDL definitions and using BPEL to establish the choreography of the two sets of message patterns.

Further messaging models can also be constructed, for example:

- Polled – this is a request/acknowledgement and polled/acknowledgement message choreography during which the initiator is unblocked i.e. more than one request message can be issued before a response message is received. The server only returns the response message when the poll message has been received;
- Publish & Subscribe – the server publishes, or announces, its service availability and the service consumers subscribe.

For both of these messaging models, multiple WSDL definitions must be created and BPEL used to establish the choreography of the message patterns.

So, as shown in Figure 7, technical interoperability is composed of three parts, any or all of which can be changed according to implementation needs. A ‘Service Adapter’ is used to link the ‘On-the-wire Interface’ and the ‘Interface’. The aim is

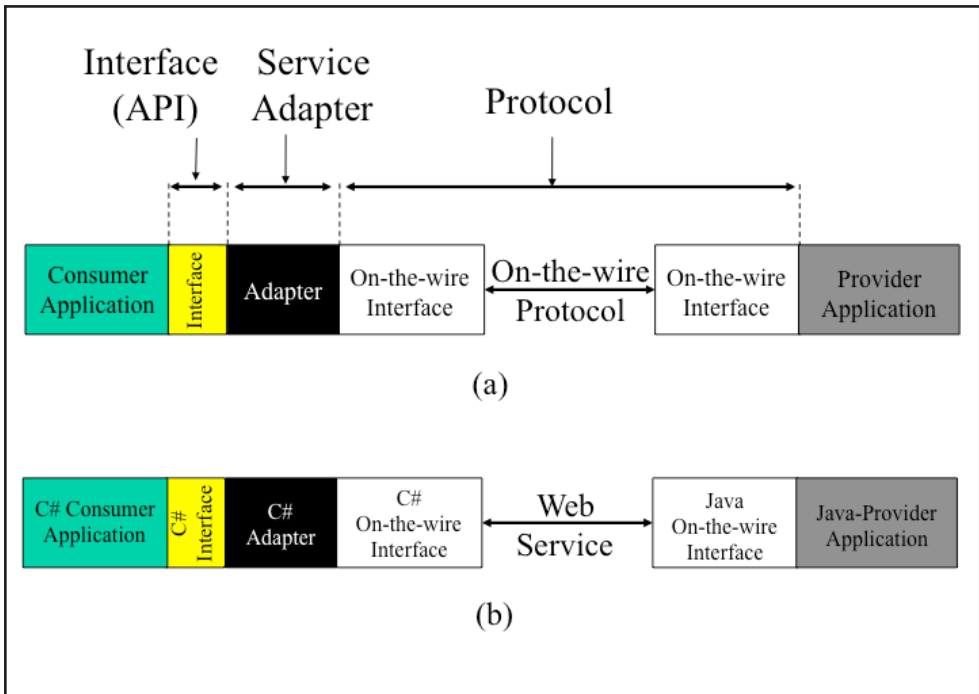


Figure 7 Interface, protocol and service adapter relationships.

to provide interoperability between the ‘Service Provider Application’ and the ‘Service Consumer Application’ such that these implementations can be changed independently. It is stressed that interoperability requires agreement for both the ‘On-the-wire Interface’ and the ‘Interface’; the ‘On-the-wire Interface’ is used to encapsulate the ‘On-the-wire Protocol’ that for a Web Services implementation would be the SOAP messaging. A further feature of Web Services interoperability is that the service provider and service consumer platforms are not required to use the same implementation technology. For example, the service consumer could be implemented using a

C#/.NET combination whereas the service provider could be a Java-based server application, or vice-versa.

So, what does this mean? APIs, interfaces and protocols are all essential. Any service implementation must address each of these otherwise it will not be able to realise the intention of the overarching SOA requirements.

10. INTEROPERABILITY & INNOVATION

When a typical system, application or tool is turned into its service-based equivalent there are technical and commercial opportunities that should be exploited. The technical opportunities are based on the interoperability capabilities. These capabilities are contained in the set of operations and the corresponding data objects. This means that:

- The data itself should be defined in a language that is independent of the implementation platform. For this reason, XML is ideal. XML is the native language for the Web and separates presentation information from the content itself;
- The operations for the service define how the service can be used by other services. Access to these operations can be limited to internal systems (i.e. inside of the corporate firewall) or can be made visible externally e.g. to act as a SAAS solution. External access is best supported through the use of WSDL to describe the service;
- Different systems can now use the service (internally and externally) to undertake the required business processes. The 'clean' interface means that non-externally visible or behavioural changes to the service have no implementation consequences on other services. This is a very powerful benefit for maintenance and support activities.

While the internal technical advantages could be sufficient justification for undertaking SOA product development the

commercial opportunities should be considered as part of the strategic decision-making. The commercial exploitation is based on the innovation opportunities that become available:

- If the service descriptions are published (either as an open source or proprietary API) then third party products can integrate with this service. This has the advantages that the costs of third party integration are borne externally while providing new marketing opportunities. Such third party integration could be undertaken by system integrators, suppliers of high products/systems e.g. Oracle and SAP, and small specialist products developers;
- If the service contains complex data descriptions then this data specification could be released as a candidate *de facto* standard. In many cases, *de facto* standardisation is the first step in *de jure* standardisation. Advocacy of such standardisation can place the organisation as a ‘thought leader’ for its market sector while also helping to establish its technology as a ‘leading edge’;
- If an original product is split into its constituent services components then new favours of the same product can be created by small changes in the ways in which these services are combined. This can be used to provide a ‘sell up’ capability for a range of products. Alternatively new services could be added to create completely new product lines. These new products could be aimed at new market sectors thereby allowing commercial growth into a new sector from a position of strength in an established market.

11. COMBINING SERVICES

While the “divide and conquer” approach for identifying separate services is an important technique for producing clear unambiguous services, real business processes will, in general, require the combination of several services. In the case of services, their combination is based upon defining the order in which the relevant operations for each service must be invoked. A related problem is defining how error conditions must be handled should some operation fail to provide the required information.

There are several standards under development for orchestrating/choreographing Web Services but BPEL is the most commonly used approach; this is because BPEL has been available in standardised form for several years (version 2.0 was released in 2007 [BPEL, 07]). BPEL is an XML-based language used to describe executable processes formed through the assembly and orchestrated interaction of multiple (Web) services supporting business processes. BPEL provides a programming language to describe long-running transactions and sequences across a collection of web services (defined in WSDL, interacting through SOAP messages). Business processes defined in BPEL may be executed under a BPEL processing engine, invoking and controlling the constituent services. Open source BPEL engines are available e.g. from Microsoft.

In broader terms, BPEL can also be used to define general Workflow. For example, once a set of services has be deployed,

different business processes can be supported by defining the appropriate workflow in BPEL. New processes can be defined by creating new BPEL descriptions and established ones amended by changing their BPEL files. This provides a very powerful mechanism to create flexible working practices in which deployment is based upon a small BPEL file distributions executed by a BPEL engine as opposed to requiring new service implementation. BPEL can also be used to create more complex message patterns for a service. For example, an asynchronous service can be supported using two synchronous message patterns that are linked using a BPEL description (in turn this means that simple WSDLv1.1 can be used to describe more complex message patterns).

12. ENTERPRISE SERVICE BUS

The Enterprise Service Bus (ESB) provides a new way to build and deploy enterprise service-oriented architectures. ESB provides an effective approach to solving common problems such as service orchestration, application data synchronization, and business activity monitoring. In its most basic form, an ESB offers the following key features:

- Web Services – support for SOAP, WSDL and UDDI, as well as standards such as WS-Reliable Messaging and WS-Security;
- Messaging – asynchronous store-and-forward delivery with multiple qualities of service;
- Data transformation – XML to XML;
- Content-based routing – publish and subscribe routing across multiple types of sources and destinations;
- Platform-neutral – connect to any technology in the enterprise e.g. Java, .Net, mainframes, and databases.

More advanced ESBs typically offer a number of additional value-added features, including:

- Adapters to enable connectivity into packaged and custom enterprise applications;
- Distributed query engine, for easily enabling the creation of data services out of heterogeneous data sources;

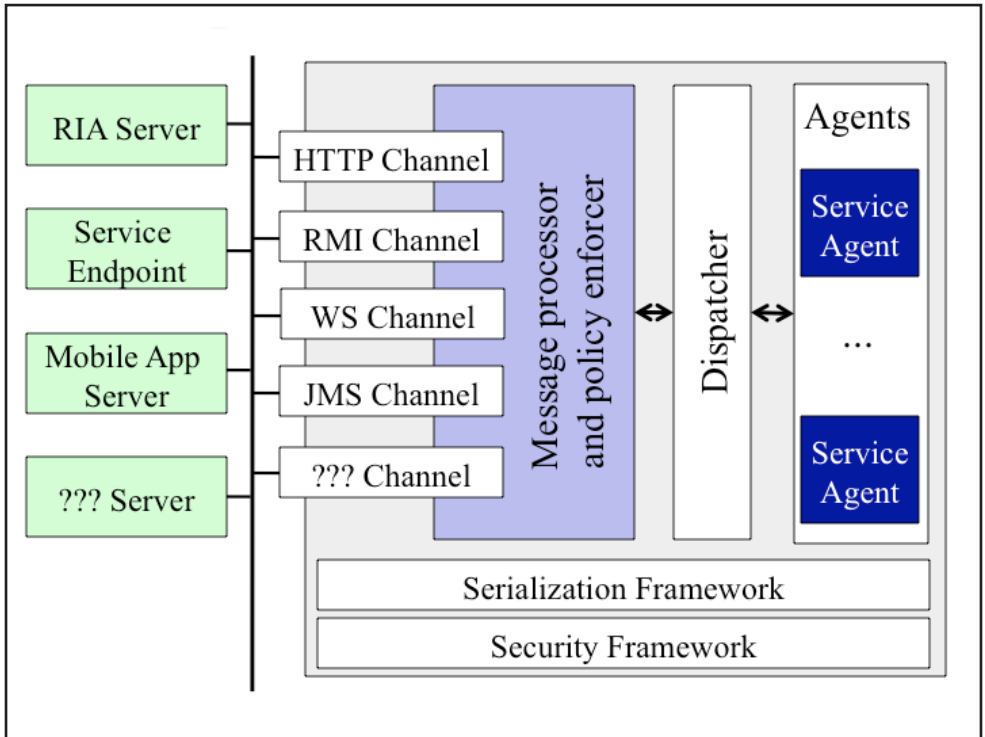


Figure 8 Internal structure of an Enterprise Service Bus.

- Service orchestration engine, for both long-running (stateful) and short-running (stateless) processes;
- Application development tools, to enable the rapid creation of user-facing applications;
- Presentation services, to enable the creation of personalized portals that aggregate services from multiple sources.

ESBs are the next step for middleware infrastructure technology. Previously, developers have used a variety of technologies to

support program-to-program communication, such as object request brokers (ORBs), message-oriented middleware (MOM), remote procedure calls (RPC), and most recently, point-to-point Web Services. These technologies are frequently grouped under the “middleware” category. ESBs are attractive for enterprise solutions because they combine features from previous technologies with new services, such as message validation, transformation, content-based routing, security, load balancing, etc. ESBs use industry standards for most of the services they provide, thus facilitating cross-platform interoperability and becoming the logical choice for companies looking to implement SOA.

As shown in Figure 8, the architecture of an ESB is centred on a bus. The bus provides message delivery services, based on standards such as SOAP, HTTP and Java Messaging Service (JMS), and is typically designed for high-throughput, guaranteed message delivery to a variety of service producers and consumers. Most ESBs support XML as a native data type, while also offering alternatives for handling other data types. The types of components that can be connected to an ESB:

- Routing and transformation – a high performance message broker is a core component of an ESB. It enables content-based routing of messages and data transformation, using standards such as XQuery and XSLT.
- Adapters, typically built to the Java Connector Architecture (JCA) specification, enable integration with a wide variety of enterprise applications;

- Distributed query engine, which is normally based on XQuery or SQL, enables the creation of data services to abstract the complexity of underlying data sources;
- Custom applications, based on standards like J2EE, which can plug into the ESB to provide a user interface to enterprise services;
- Service orchestration (or BPEL) engine, which can sequence the execution of services and keep state for long-running processes.

Although many people correlate an ESB with integration and mediation, its primary function is actually that of service platform, or more specifically the “virtualization of service agents.” A service agent is the application code that implements service functionality, and virtualization of service agents is the true breakthrough for ESB. An ESB provides a service container that virtualizes a service and insulates it from its protocols, invocation methods, MEPs, quality of service requirements, and many other infrastructure concerns.

Infrastructure refers to nonfunctional aspects of a service, including its protocols, invocation methods, etc. Middleware technologies evolve at a different rate from application functionality, and therefore it is desirable to separate these concerns. In the past the Common Object Request Broker Architecture (CORBA) was a common implementation technology but very few people would consider adopting it today. Instead Web Services would be used. ESB allows

a developer to build a service, using service agents, that is completely independent from the technology that will be used to expose its capabilities. The service agent running in the platform's agent container is completely separated from the technology used to expose its capabilities to the outside world. The message pipeline processor and policy enforcer can expose the service through any number of communication channels, supporting a wide assortment of client systems, including Rich Internet Applications (RIAs) and mash-ups, rich mobile applications (RMAs), Rich Desktop Applications (RDAs), remote service endpoints, and others. The pipeline processor also mediates access to the service agent by enforcing whatever policies apply to the service, such as security, reliability, or transformational policies.

This type of abstract component model, now found in most ESBs, enables the kind of clean separation of concerns between application and infrastructure. It supports the concept of an infrastructure services model in which infrastructure capabilities are externalized from applications and their application platforms and modelled as services that can be applied to service agents and service interactions via declarative policies. The infrastructure itself becomes responsible for ensuring that policies are properly enforced.

13. OVER-THE-HORIZON

As with all technologies, there is a very rapid development cycle for Web Services. New standards are under development and new versions of established standards are released every few months. The set of technologies that need to be monitored closely include:

- Full Enterprise Service Bus – ESB deployment and interoperability is still in its infancy. Many ESB solutions support only the messaging infrastructure. Next generation ESB solutions will focus on full ESB service support as well as sector-specific features. For example, a Higher Education ESB could provide native support for e-learning and e-research using established service interface definitions;
- New profiles of WS-* standards – there are many new WS-* standards under development and refinement. Until best practice experience has been obtained and the accompanying best practice profiles have been created it is unlikely that these standards will be successfully deployed in Enterprise systems. Typically, these profiles are going to lag behind the release of the standard by 2-3 years;
- More powerful RESTful implementations – the original advantage of REST-based implementations was that they were simple and based upon deployed technology e.g. Web servers. SOAP requires special tools and is considered overly complex. However, once Enterprise-oriented solutions are

required, REST is too simple. For example, there is no security mechanism apart from HTTPS. Therefore, REST will be combined with other ‘simple’ technologies to increase its range of capabilities. As REST architectures grow more complex, SOAP approaches are being simplified (through profiling) and so the two will converge in range of functional capability and implementation complexity;

- Support tools for WSDLv2.0 – code generation tools for WSDSLv2.0 are not yet available. Early tools are now available, such as the W3Cs WSDLv1.1 to WSDLv2.0 conversion. Mainstream support for WSDLv2.0 in the J2EE and .NET World’s should occur in the next 12-18 months. Some work by IBM has shown how WSDLv2.0 can be used to describe RESTful services and work is underway to develop corresponding code generation tools;
- Semantic Web – there are many difficulties in finding information across the Web, because each relationship must be explicitly defined. New approaches are required to allow relationships between information to be derived from the content itself and the way that content is represented. This is the aim of the semantic web and its core technologies of the Resource Description Framework Schema (RDFS), and the Web Ontology Language (OWL). At present, there are too few commercially robust tools for these to be mainstream but this will change in the next few years;
- Web Service enabled network devices – companies such as Cisco are working on new network devices that incorporate

Web Service capability e.g. SOAP message processing, support for service registries, etc. If the router infrastructure includes Web Services functionality then access to the end services is simplified and the network itself can support key functionality e.g. WS-Security;

- Cloud Computing – this is the provision of dynamically scalable and virtualised services through the Web. Cloud computing is a combination of Software-, Platform- and Infrastructure-as-a-Service. The Gartner ‘Hype Cycle’ released in 2008 showed that Cloud Computing was 2-5 years away from accepted deployment and had not yet entered the ‘Trough of Disillusionment’. Even so, Amazon already provides Cloud Computing solutions and so now is an ideal time to consider how such a technology can be exploited;
- Model Driven Architecture (MDA) – MDA and its set of related approaches (model driven design, model driven testing, etc.) are based upon on the principle that it is possible to derive real solutions for system from models of the system. In the case of a Web-based service this approach requires the creation of a functional model of the service (there could be more than one such model) followed by transformation of this model (this could have several stages) into an implementation for the target platform e.g. C# for a .NET based solution. The aim is to make is easier to create and deploy correct Web Services by minimising the time spent on the traditional development, coding and testing activities.

14. IN CONCLUSION

A lot of material has been covered in the previous pages. Do not get overwhelmed by the complexity of the technology. When adopting Web technologies it is important to take an incremental approach. The starting point is the Core Web technologies.

The Core Web technologies are the minimal set needed to provide service-oriented computing. The core defines the mechanism for passing messages between a client (service consumer) and a service (service provider), and the way to describe services (the operations/messages they support). The Core consists of the use of XML, SOAP and WSDL such that:

- The data is described as XML with XML Schema used to describe the structure, semantics and constraints of XML documents. Most Web Services specifications are defined by languages and documents expressed in XML Schema;
- For Web Services the data messaging uses SOAP. SOAP is a protocol for the exchange of messages described in XML over a distributed network e.g. the Internet. Several versions of SOAP are deployed but SOAPv1.1 has widest adoption;
- WSDL is used to describe Web Services in terms of the message exchange patterns. A WSDL service description defines the operations, messages and end points for the service. WSDLv1.1 is the most commonly used version with WSDLv2.0 in early deployment.

While the core Web Service specifications are sufficient to achieve basic service interoperability, they leave many details unspecified. Furthermore, the core specifications are specific to only limited technologies. The broader collection of Web Services specifications, denoted WS-*, define a more comprehensive service architecture and further detail solutions to different interoperability problems. They also uncouple the different behaviours and representations, providing a more comprehensive model for abstracting capabilities from underlying representations. Therefore many of the WS-* specifications are designed to be combined with other specifications to define a complete Web Services or SOA model.

There are an almost endless number of ways in which Web Services can be implemented. Therefore, best practice guidance is essential. The Web Services Interoperability Organization (WS-I) is an open industry consortium chartered to establish Best Practices for Web Services interoperability, for selected groups of Web Services standards, across platforms, operating systems and programming languages.

WS-I committees and working groups create Profiles and supporting Testing Tools based on best practices for selected sets of Web Services standards. The Profiles and Testing Tools are available for use by the Web Services community to aid in developing and deploying interoperable Web Services. The primary deliverables from WS-I are Profiles that provide implementation guidelines for how related Web Services specifications should be used together to achieve interoperability.

The core deliverable from WS-I is the Basic Profile. The latest version of the WS-I Basic Profile v1.1 recommends the use of SOAPv1.1 messaging, over HTTPv1.1 to exchange XML data for Web Services described using WSDLv1.1 with UDDI v2.0 used for service publication and discovery.

Flexible service implementation requires the definition of a clear interface and messaging protocol. Full interoperability requires agreement on both the interface and the protocol. The interface is a reflection of the business application representation of the service whereas the protocol is the messaging interpretation of the service. A Service Adapter provides the glue between the Interface and the Protocol. The Interface hides the details of the Protocol from the application and the Protocol ensures that the application-specific functionality is supported using a common network infrastructure. Application programming interfaces, interfaces and protocols are essential. Any service implementation must address each of these otherwise it will not be able to realise the intention of the overarching SOA requirements.

Finally, of all the material covered in the six sessions, this is the content that is most susceptible to change. There are a lot of relevant standards being developed and in many cases these are undergoing significant revision as we gain more implementation experience. Likewise, the tool-sets that make it possible to create a Web Service solution are also being changed and improved. Therefore, it is important to monitor developments closely.

APPENDIX A – BIBLIOGRAPHY

- [BPEL, 07] *Web Services Business Process Execution Language Version 2.0*, OASIS Standard, 11th April 2007. [<http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>].
- [Fielding, 00] *Architectural Styles and the Design of Network-based Software Architectures*, Roy Thomas Fielding, Ph.D. Dissertation, University of California, Irvine, 2000. <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
- [SOAP, 00] *Simple Object Access Protocol (SOAP) 1.1*, W3C Note, 8th May 2000. [<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>].
- [SOAP, 07] SOAP Version 1.2 Part 0: Primer (Second Edition), W3C Recommendation, 27th April 2007. [<http://www.w3.org/TR/soap12-part0/>].
- [WADL, 06] *Web Application Description Language*, Marc Hadley, SUN Microsystems, 2006. [<https://wadl.dev.java.net/wadl20061109.pdf>].
- [WSA, 06] Web Services Addressing 1.0 – Core, W3C Recommendation, 9th May 2006. <http://www.w3.org/TR/ws-addr-core>.
- [WSDL, 01] *Web Services Description Language (WSDL) 1.1*, W3C Note, 15th March 2001.

- [WSDL, 07] *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, W3C Recommendation, 26th June 2007. [<http://www.w3.org/TR/wsdl20-primer>].
- [WSI, 06] *Web Services Interoperability Basic Profile Version 1.1*, 2nd Edition, Eds K.Ballinger, D.Ehnebuske, C.Ferris, M.Gudgin, C.K.Liu, M.Nottingham and P.Yendluri, Web Services-Interoperability Organization, April 2006.
- [XML, 04] *XML Schema Part 0: Primer*, Second Edition, W3C Recommendation, 28th October 2004. [<http://www.w3.org/TR/xmlschema-0/>].

APPENDIX B – ACRONYMS

API	Application Programming Interface
BPEL	Business Process Execution Language
CORBA	Common Object Request Broker Architecture
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
HTTPS	Hypertext Transport Protocol (Secure)
ISO/IEC	International Standards Organization/International Electrotechnical Commission
J2EE	Java 2 Enterprise Edition
JCA	Java Connector Architecture
JMS	Java Messaging Service
MDA	Model Driven Architecture
MEP	Message End Point
MIME	Multipurpose Internet Mail Extensions
MOM	Message Oriented Middleware

MPEG	Motion Picture Experts Group
MTOM	Message Transmission Optimization Mechanism
OASIS	Organization for the Advancement of Structured Information Standards
ORB	Object Resource Broker
OWL	Web Ontology Language
PKI	Public Key Infrastructure
POX	Plain Old XML
RDA	Rich Desktop Application
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
REST	Representational State Transfer
RIA	Rich Internet Application
RMA	Rich Mobile Application
RMI	Remote Method Invocation
RPC	Remote Procedural Call
SAAS	Software As A Service
SAML	Security Assertion Markup Language
SMTP	Simple Mail Transfer Protocol

SOA	Service Oriented Architecture
SQL	Structured Query Language
UDDI	Universal Description, Discovery and Integration
UK	United Kingdom
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WSDL	Web Services Description Language
WS-I	Web Services Interoperability Organisation
XML	Extensible Markup Language
XOP	XML-binary Optimized Packaging
XSD	XML Schema Definition

INDEX

A

API 33—37, 39, 55

Application Programming Interface.

See API

Asynchronous 35

B

BPEL 29, 32, 35, 40, 41, 45, 53, 55

Business Process Execution Language. *See* BPEL

C

Common Object Request Broker Architecture. *See* CORBA

CORBA 45, 55

Core Web Technologies 8—12

E

Enterprise Service Bus. *See* ESB

ESB 42—46, 47, 55

Extensible Markup Language 57

F

Fire-and-forget 34

FTP 25, 55

H

HTML 8, 9, 25, 55

HTTP 8, 10, 11, 12, 13, 15, 25, 28, 30, 44, 55

HTTPS 48, 55

Hypertext Markup Language.

See HTML

I

Interfaces 33—37

Interoperability 4, 7, 20, 38—39, 51, 54, 57, 60

ISO/IEC 22, 55

J

J2EE 6, 45, 48, 55

Java Connector Architecture.

See JCA

Java Messaging Service. *See* JMS

JCA 44, 55

JMS 44, 55

M

MDA 49, 55

MEP 55

Message Transmission Optimization
Mechanism. *See* MTOM

MIME 12, 27, 30, 55

Model Driven Architecture.
See MDA

MOM 44, 55

MTOM 23, 27, 28, 31, 56

O

OASIS 30, 53, 56

ORB 56

Organization for the Advancement
of Structured Information
Standards. *See* OASIS

OWL 48, 56

P

PKI 29, 56

Plain Old XML. *See* POX

Polled 35

POX 11, 12, 56

Protocol

FTP 25, 55

HTTP 8, 10, 11, 12, 13, 15, 25, 28, 30,

44, 55

HTTPS 48, 55

Protocols 33—37

Publish & Subscribe 35

R

RDA 56

RDF 16, 56

RDFS 48, 56

Remote Procedural Call. *See* RPC

Representational State Transfer.
See REST

Resource Description Framework.
See RDF

Resource Description Framework
Schema. *See* RDFS

REST 8, 11, 12, 47, 48, 56

RIA 56

Rich Desktop Application. *See* RDA

Rich Internet Application. *See* RIA

Rich Mobile Application. *See* RMA

RMA 56

RMI 56

RPC 13, 44, 56

S

SAAS 38, 56

SAML 29, 56

Semantic Web 48

Service Adapter 33—37

Service Oriented Architecture 1, 57

Simple Mail Transfer Protocol.
See SMTP

SMTP 25, 56

SOAP 4, 8, 9, 10, 13, 14, 15, 17,
18, 19, 20, 22, 23, 25, 27,
28, 29, 30, 31, 36, 40, 42,
44, 47, 48, 49, 50, 53

SOAP Messages 13—15

SOAPv1.1 10, 22, 23, 50, 52

SOAPv1.2 22, 23, 27

Software As A Service. *See* SAAS

SQL 45, 57

Synchronous 34

U

UDDI 10, 22, 23, 25, 29, 32, 42,
52, 57

Uniform Resource Identifier.
See URI

Universal Description, Discovery
and Integration. *See* UDDI

URI 11, 30, 57

W

W3C 30, 53, 54, 57

WADL 11, 12, 53, 57

Web Application Description Lan-
guage. *See* WADL

Web Ontology Language. *See* OWL

Web Services 3, 4, 5, 6, 8, 9, 10, 12,
13, 16, 19, 20, 22, 24, 25,
26, 27, 28, 29, 30, 31, 34,
35, 36, 40, 42, 44, 45, 47,
49, 50, 51, 52, 53, 54, 57

Web Services Addressing. *See* WS-
Addressing

Web Services Description Lan-
guage. *See* WSDL

Web Services Interoperability.
See WS-I

Web Services Policy. *See* WS-Policy

Web Services Security. *See* WS-
Security

Web Services Stack 30—32

World Wide Web Consortium.
See W3C

WS-Addressing 13, 23, 28, 31

WS-Atomic Transaction 31

WS-Coordination 32

WS-Discovery 11, 32

WSDL 10, 12, 16, 17, 18, 19, 24,
25, 26, 28, 29, 31, 34, 35,
38, 40, 42, 50, 53, 54, 57

WSDL Document 16—19

WSDLv1.1 10, 16, 19, 22, 41, 48,
50, 52

WSDLv2.0 10, 16, 19, 22, 48, 50

WS-I 4, 20, 22, 23, 25, 27, 51, 52,
57

WS-I Organisation 20—23

WS-Policy 29, 32

WS-Reliable Messaging 31, 42

WS-Resource Framework 31

WS-Security 29, 32, 42, 49

WS-* Standards 27—29

WS-Trust 32

X

XML-binary Optimized Packaging.
See XOP

XML Schema Definition. *See* XSD

XOP 23, 27, 28, 57

XSD 9, 16, 18, 57

This work was funded by Digital 20/20. Further information is available at <http://www.digital2020.org.uk>.

This booklet was written and produced by Colin and Christine Smythe of Dunelm Services Ltd (<http://www.dunelm.com>), at 34 Acorn Hill, Stannington, Sheffield, S6 6AW, Tel: 0114-2334009, E-mail: colin@dunelm.com.