

Introduction to Service Oriented Architecture

Karl Ray

<http://anengineersperspective.com>

© 2010

Table of Contents

Table of Contents	i
Overview	1
Introduction to Service-Oriented Architecture	2
Evolution of Service-Oriented Architecture	2
Standardization of SOA	7
XML	12
Web services	15
Web services Invocation	17
Web services Taxonomy	21
Primary SOA Building Blocks: SOAP, UDDI, and WSDL	24
SOAP	24
UDDI	27
WSDL	31
Web services Extensions	35
SOA Platforms	41
Java 2 Platform Enterprise Edition	41
Microsoft .Net Platform	44
SOA as the Enterprise Architecture	46
Designing for SOA	52
Reusing Legacy Applications in an SOA	55
Conclusion	56
References	57

Overview

The rapidly-changing business environment and the ubiquity of the Internet and the World-Wide Web have led to the emergence of platform-independent, web-based technologies as the standard building blocks for enterprise integration. These technologies are called "Service Oriented Architecture" (SOA). Fundamental to SOA are the concepts of Web services and the Enterprise Service Bus (ESB). But SOA is also the enterprise Information Technology (IT) infrastructure – Web portals, networks, common software services, web-enabled legacy applications, and databases that support delivery of Web applications. This book explores the evolution and concepts of SOA in both contexts, the enabling technologies and as an enterprise IT infrastructure.

Introduction to Service-Oriented Architecture

Service Oriented Architecture (SOA) is a design paradigm that seeks to combine reusable, coarse-grained, business processes, packaged as automated Web services, into new applications. SOA defines these Web services and related components and the IT infrastructure that allows applications to be composed from services written in different languages and running on different, perhaps distributed, platforms. SOA services collaborate by passing messages from one service to another or a service may participate in an application by coordinating activities amongst other services. SOA concepts rely upon and evolved from distributed and object-oriented programming concepts.

Evolution of Service-Oriented Architecture

As enterprises grow, integrating the variety of legacy, custom, and commercial package applications involved has become increasingly complex. Over time, a number of integration patterns have evolved. The earliest pattern was simple sharing of data through files. Later, sharing of data through enterprise databases became a commonly used pattern. Socket programming answered the need for real-time passing of data between applications. Remote Procedure Call (RPC) run-time libraries abstracted the socket interface enabling increased programmer productivity. RPC, however, has a number of shortcomings. These include:

- There is little room for code reuse because the code for marshaling and unmarshaling, and the code for network communication is buried in the client and server applications.
- RPC isn't language independent; the client and the server must employ the same programming language.
- The integration of the client and server is point-to-point and is not suitable when a number of applications need to be integrated.

- RPC is also not suitable if a large number of remote calls are involved. This is because the synchronous nature of the call does not allow the client to proceed before the server completes its work.¹

Distributed objects, through the use of Object Request Brokers (ORBs), and Asynchronous Messaging, typically through the use of commercial Message-Oriented Middleware (MOM) evolved to address the limitations of using RPCs for integration. There are three major distributed object implementations. These are platform-dependent, language-dependent, and platform and language independent. The Java Remote Method Invocation (RMI) is an example of a language-dependent technology. Microsoft Distributed Component Object Model (DCOM) and the IBM System Object Model (SOM) are examples of platform-dependent technologies. Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG). It can be used independently of language or platform. Products from different vendors can typically be integrated using CORBA. Important concepts in CORBA are the message broker, which allows remote objects to be located by name, i.e. by a Uniform Resource Locator (URL), and the use of an Interface Description Language (IDL), which allows language and platform independence. These concepts are carried over into SOA as the registry component and the reusable marshaling, unmarshaling, and communication component. Despite the advantages of CORBA-based integration, there are disadvantages. These include:

¹ Roshen, W. *Services-Based Enterprise Integration Patterns Made Easy*, Retrieved 26 May 2008 from: http://www.ibm.com/developerworks/webservices/library/ws-intpatterns/index.html?S_TACT=105AGX04&S_CMP=EDU

- An ORB-based solution is not scalable to high volumes, due to the synchronous nature of the interaction, which blocks the client application from proceeding further with its work until it receives the response from the server.
- The interaction between the client object and the server object is fine-grained and results in many trips across the network. Network bandwidth is not used efficiently, further limiting the scalability of the solution.
- ORB-based communication is not reliable, and there is no guarantee that the messages and return values will be delivered to the intended targets. Thus, the client application might experience a hang up under certain circumstances, such as a break in the network connection.
- Although CORBA, in principle, is language independent, most of the commercial products that employ ORBs are language specific, such as Java or Java 2 Platform, Enterprise Edition (J2EE). This is because the CORBA open standards proved to be difficult to implement in their most general form. This limits the integration capabilities of ORBs to the applications that are written in those specific languages.²

To deal with these problems, asynchronous messaging through the use of MOM was developed. The evolution of MOM implementations led to the notion of the Enterprise Service Bus (ESB) in the SOA. MOM solutions include the open source Apache Software Foundation ActiveMQ, IBM WebSphere MQ, Talarian SmartSockets, Microsoft MSMQ, Oracle Advanced

² Roshen, W. *Services-Based Enterprise Integration Patterns Made Easy*, Retrieved 26 May 2008 from:

http://www.ibm.com/developerworks/webservices/library/ws-intpatterns/index.html?S_TACT=105AGX04&S_CMP=EDU

Queuing, and BEA Systems MessageQ.³ In asynchronous messaging, applications do not establish a dedicated communication link and do not communicate directly with each other. Instead, they communicate through a queue. Messaging can be point-to-point, if there is a dedicated receiving queue for each application. For one-to-many messaging, a publish and subscribe pattern is employed. For more complex messaging, for example distributing messages amongst servers for load-balancing or routing of messages depending on content or size, the middleware provides a message router or message broker, see Figure 1.

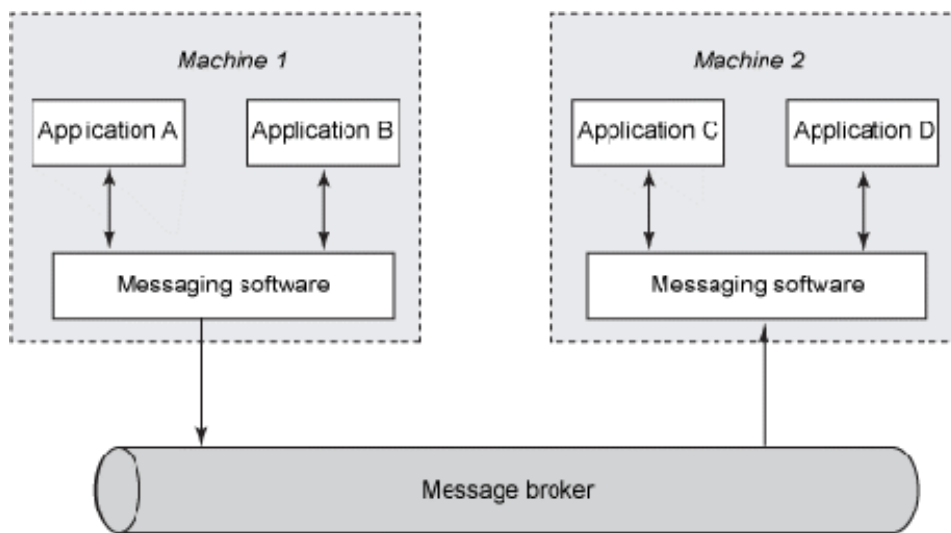


Figure 1 Message Broker, from Roshen, W. (2008), *Services-Based Enterprise Integration Patterns Made Easy*, Retrieved 26 May 2008 from:

<http://www.ibm.com/developerworks/webservices/library/ws-intpatterns/index.html>

Although asynchronous messaging is inherently one-way communication, two-way communication can be simulated by using a request queue on both the client and server side and by invoking behavior, for example, requiring acknowledgment, in the receiver. This is depicted

³ Defining Technology Incorporated. *Basic MOM*, retrieved 26 May 2008 from:

<http://www.middleware.org/mom/basicmom.html>, 2008

in Figure 2. By persisting the message on both sides of the interface and requiring acknowledgement, communication can be guaranteed. Another advantage of using MOM, is that large sets of data can be transported across the network in each exchange, resulting in more efficient use of network bandwidth.⁴

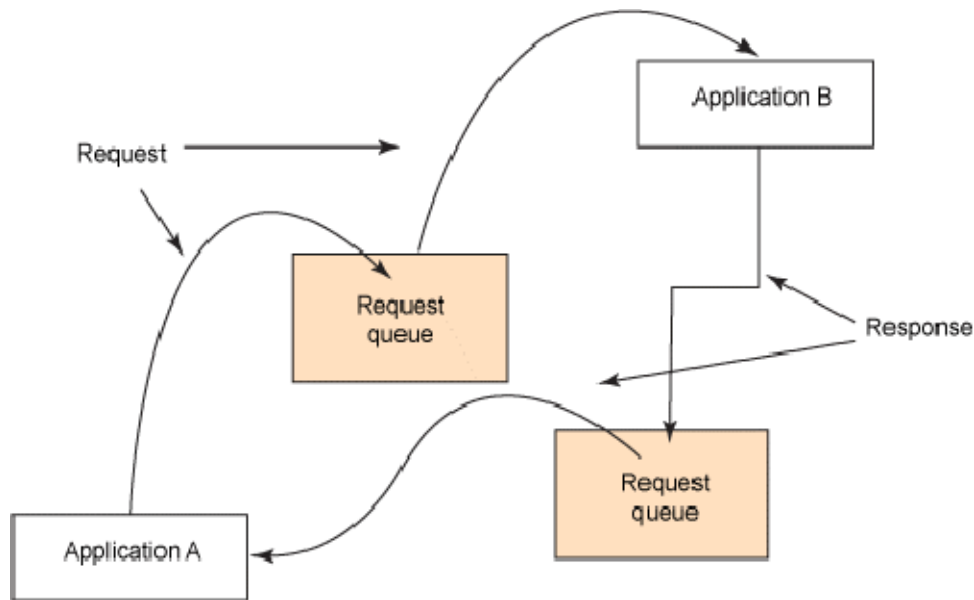


Figure 2 Simulated Synchronous Messaging Using Messaging Software, from Roshen, W. (2008). *Services-Based Enterprise Integration Patterns Made Easy*, Retrieved 26 May 2008 from: <http://www.ibm.com/developerworks/webservices/library/ws-intpatterns/index.html>

Contemporary SOA builds upon the concepts of the message broker, introduced by CORBA, and integration using coarse-grained messages, introduced by MOM. These concepts allow development of robust, distributed, scalable systems. However, to enable enterprise-scale

⁴ Roshen, W., *Services-Based Enterprise Integration Patterns Made Easy*, Retrieved 26 May 2008 from: <http://www.ibm.com/developerworks/webservices/library/ws-intpatterns/index.html>, 2008

integration, standard implementations of these concepts were needed. In subsequent sections, the development of these needed standards and their implementations will be discussed.

Standardization of SOA

Enterprises are changing rapidly through acquisitions, divestments, and mergers. This change leads to daunting integration challenges. Enterprises demand more and more that vendors provide standardized, cross-platform, cross-language integration capabilities. The ubiquity of the Internet and the World-Wide Web led enterprises and vendors to standardize on web-based technologies as the basic building blocks for integration. Standardization of these capabilities was through the efforts of the World Wide Web Consortium (W3C). The foundations of SOA as it is implemented today are Simple Object Access Protocol (SOAP); Web services Description Language (WSDL); and Universal Description, Discovery, and Integration (UDDI) Protocol. Extensible Markup Language (XML) is a simple, very flexible text format that defines the format and structure of messages passing through the services.⁵ Other organizations active in establishing standards for SOA are the Web services Interoperability (WS-I), the Internet Engineering Task Force (IETF), and the Java Community Process (JCP).

SOAP was first submitted to the W3C in 2000. Version 1.2 became a W3C Recommendation on June 24, 2003.⁶ The specification was designed to standardize Remote Procedure Call (RPC) communication. With SOAP, parameters passed between components are serialized into XML, transported between components, and then deserialized into the

⁵ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006

⁶SOAP, retrieved June 20, 2008 from <http://en.wikipedia.org/wiki/SOAP>

component's native format.⁷ Schema definition and schema translation allow mapping of the sending component's data representation to the receiving component's.

WSDL provides the public interface of an SOA component. In this sense, it is analogous to the header file in C++ or the Specification in Ada. WSDL was first recommended by W3C in 2001.

The remaining foundation, UDDI, is promoted by Organization for the Advancement of Structured Information Standards (OASIS). It allows software services to advertise themselves either as free or on a fee-per-use basis. UDDI is the broker component of Web services that allows service providers and requesters to locate each other.

Web services continue to evolve. In addition to the four foundations, a second generation of Web services specifications is being developed. These specifications are commonly referred to collectively as "WS-*" because the majority of the titles of the specifications begin with "WS-".⁸

The primary building blocks of SOA are SOAP, WSDL, and UDDI. XML is the standard language of communication between all of these elements. The relationships between these components and service requesters and service providers are shown in Figure 3. The relationships amongst the various standards are shown in Figure 4.

⁷ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006

⁸ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006, p 157

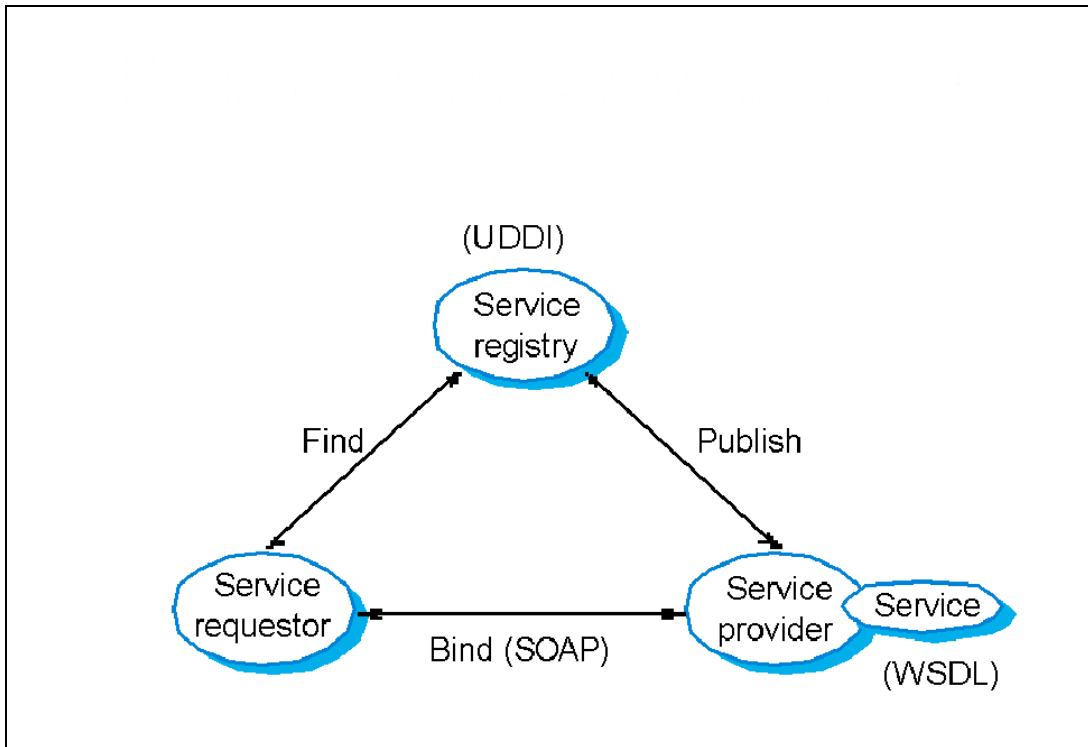


Figure 3 Primary SOA Components, Source: Sommerville, I. (2006). *Software Engineering, 8th edition*. Addison-Wesley

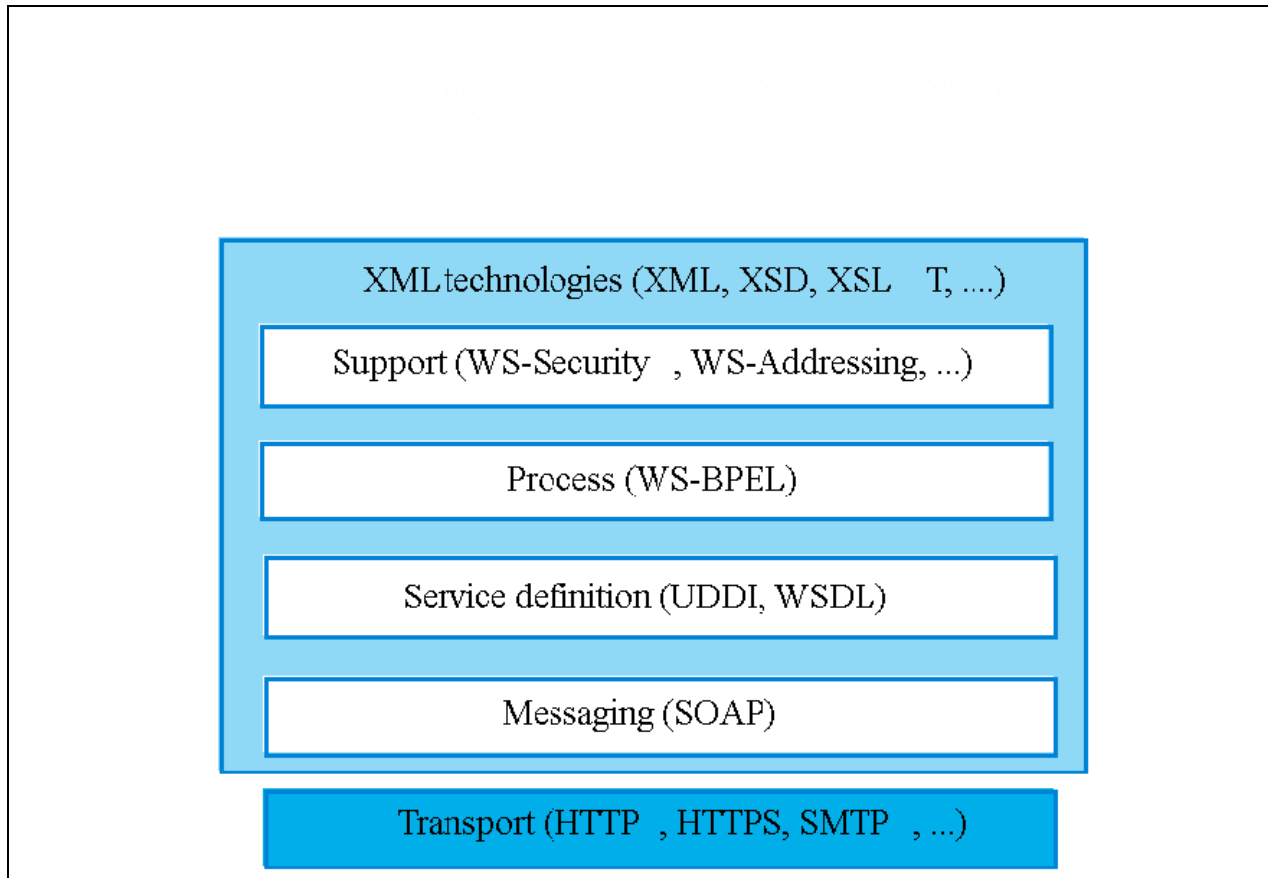


Figure 4 Web Service Standards. Source: Sommerville, I. (2006). *Software Engineering, 8th edition*. Addison-Wesley

Like object-oriented architecture and reuse, SOA also depends on a good understanding of reuse and on the building of reusable components. However, SOA extends the idea of reuse to business services.⁹ Examples of business services are obtaining a stock quote, posting a credit card transaction, or providing a weather report for a specific location. SOA services are typically of much coarser granularity, that is, they provide much more information in one invocation, than

⁹ Hurwitz, J., Bloor, R., Baroudi, C., and Kaufman, M., *Service Oriented Architecture for Dummies*, John Wiley & Sons, 2007

object-oriented methods. This enables better bandwidth utilization in distributed systems.

According to Hurwitz, et. al, characteristics of SOA are:

- SOA is for building business applications.
- SOA is a black-box component architecture.
- SOA components are loosely coupled.
- SOA components are orchestrated to link together through business processes to deliver a well-defined level of service¹⁰

The primary driver for SOA is the need to rapidly deploy and integrate new capabilities.

In many cases, new, web-based applications must interface with legacy applications. These interfaces can be provided by enabling the legacy applications to communicate through the web via adapters. In addition to the Web services components, the adapters, messaging middleware, and integration middleware that enable this integration are also considered to be part of an enterprise's SOA. Other typical components include Web portals, a user authentication service (single sign-on), and commercial software packages. From this perspective, SOA refers not only to the building blocks, WDSL, SOAP, UDDI, and XML, but to the entire Information Technology (IT) infrastructure. The SOA Consortium has proposed an SOA Reference Architecture, Figure 5, in an attempt to promote a standard view of SOA from this enterprise infrastructure perspective.¹¹

¹⁰ Hurwitz, J., Bloor, R., Baroudi, C., and Kaufman, M., *Service Oriented Architecture for Dummies*, John Wiley & Sons, 2007

¹¹ Pai, Y., *SOA Practitioners' Guide*, Retrieved June 2, 2008, from:
<http://dev2dev.bea.com/2006/09/SOAPGPart1.pdf>, 2006

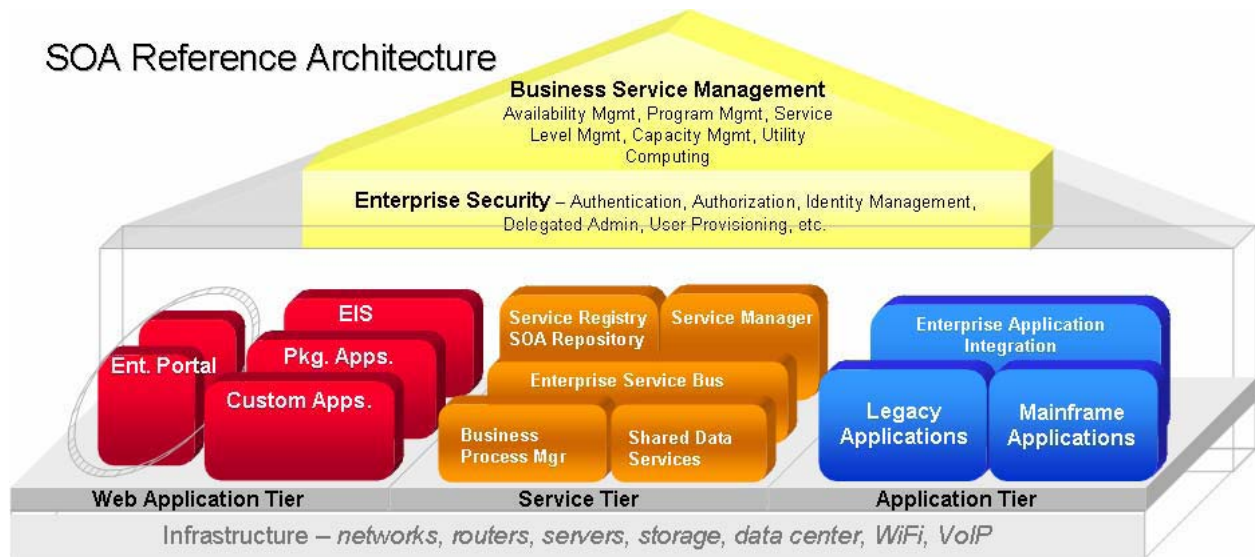


Figure 5 SOA Reference Architecture, from Pai, Y., (2006). *SOA Practitioners' Guide*, Retrieved June 2, 2008, from: <http://dev2dev.bea.com/2006/09/SOAPGPart1.pdf>

Having discussed SOA at a high level, the following sections will provide details on SOA components and implementations. The discussion will begin with an explanation of XML and Web services.

XML

XML was derived from Standardized General Markup Language (SGML) (ISO 8879). SGML was, in turn, derived from IBM's General Markup Language (GML), which was developed in the 1960s and heavily used from the 1970s through the 1990s as a markup language for developing and maintaining documents on IBM mainframes. Text files containing GML formatted documents were converted for display or printing using IBM's Script/VS. HTML, the language of web pages, is also derived from SGML. XML 1.0 was adopted as a standard by the

W3C on February 10, 1998.¹² XML can be used to represent data in a standardized manner.

Through XML, meaning and context can be attached to information transmitted across the Internet. XML forms the foundation layer upon which all SOA is built.

Unlike SGML or HTML, XML is a specification for creating custom markup languages rather than a markup language in itself. Tags are not predefined in XML. For example, the tag, `<p>`, which means to start a new paragraph in HTML has no predefined meaning in XML. In XML, the author defines the meaning of the tags. Consider the example of XML that appears in Figure 6. This example illustrates both the strengths and weaknesses of XML. The note is easily read and interpreted by a human. The syntax, `<tag>` tagged information `</tag>` is familiar and intuitive. The tags used, `<note>`, `<to>`, `<from>`, `<heading>`, and `<body>`, make sense in the context of the information being conveyed. Flexibility, relevance to the context, readability, and cross-platform portability are XML's strengths. But this XML document consists only of information wrapped in tags. Someone must write software to interpret the tags before the information can be sent, received, or displayed. The tags used in the example, though useful for our application, are not defined in any XML standard. These tags are invented by the author of the XML document. The XML language has no predefined tags. The meaning of these tags must be provided by the software processing the XML or by another, external standard.¹³

¹² W3C, *XML*, retrieved June 20, 2008 from: <http://www.w3.org/XML/>, 1998

¹³ W3Schools, *What is XML?*, retrieved June 20, 2008 from http://www.w3schools.com/xml/xml_what.asp

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Figure 6 Example of XML, retrieved June 10, 2008 from:

http://www.w3schools.com/xml/xml_what_is.asp

While XML's lack of predefined tags might seem to limit its practical application, thousands of domain specific application languages have been implemented using XML. Some familiar examples, including some already discussed as part of an SOA, are:

- WSDL
- SOAP
- Open Document Format (ODF) for creating transportable documents
- Extensible Hypertext Markup Language (XHTML) the newest version of HTML
- Wireless Markup Language (WML) used to create web content to be displayed by "micro browsers" using Wireless Application Protocol (WAP) on handheld devices
- Really Simple Syndication (RSS) a language for displaying news feeds
- Synchronized Multimedia Integration Language (SMIL) for describing and presenting multimedia presentations from the web
- MathML
- GraphML
- Scalable Vector Graphics
- MusicXML

A particular advantage to passing XML documents in an interface between components is its extensibility. Suppose that the "note" XML in the previous example is used in several different messaging applications and that one of these applications has a new requirement, the addition of a "priority" attribute. To meet this requirement we only need add a new, optional <priority>this priority</priority> tag to the XML vocabulary. The XML parsers in existing applications will just ignore the priority attribute. The additional attribute is information, but for applications that do not know about "priority", its existence is unimportant. The essential XML structure and values do not change, so the interface is unaffected.¹⁴

Given XML's flexibility, developing standard XML vocabularies to enable integration across an industry or enterprise or vertically within an industry or enterprise can be challenging. OASIS is active in coordinating and standardizing these efforts. Examples of standard XML vocabularies and other XML artifacts are hosted on OASIS' website at <http://xml.coverpages.org/>.

Web services

The W3C defines a Web Service as "a software system designed to support interoperable Machine to Machine interaction over a network."¹⁵ Web services are a distributed computing technology that enables creation of client/server applications. A Web Service publishes information on the Internet or an intranet. Like a web page, a Web Service is accessed through a Uniform Resource Locator (URL). Web services rely on existing web technologies such as Hyper Text Transfer Protocol (HTTP). Unlike a web page, information which is available

¹⁴ Chappell, D., *Enterprise Service Bus*, Cambridge, MA, O'Reilly, 2004

¹⁵ Haas, H., *Web Services Glossary*, retrieved May 29, 2008 from: <http://www.w3.org/TR/ws-gloss/>, 2004

through a Web Service is accessed by software, not by a human using a browser. Web services have advantages over other technologies:

- Web services use standard XML languages to communicate. Thus they are platform-independent and language-independent. A client program, programmed in C++ and running under Windows can talk using XML to a Web Service programmed in Java and running under Linux.
- Web services typically use HTTP for transmitting messages (such as the service request and response) allowing these to pass through proxies and firewalls.
- Web services enable loosely coupled systems.

Disadvantages of Web services are:

- Transmitting data in XML is much less efficient than using proprietary binary code. This limits their use in real-time applications.
- Web services lack versatility. They are limited to HTTP request operations, HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, and CONNECT. Supporting services, such as persistency, notifications, lifecycle management, security, transactions, and so on, are not natively available in HTTP. However, additional functionality can be provided by the SOAP header. The limitations of Web services are being addressed by the emerging, second-generation WS-* specifications.¹⁶

¹⁶ Sotomayor, B., *Globus Toolkit 4 Programmer's Tutorial*, retrieved May 29, 2008 from: <http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html>, 2005

Web services Invocation

A system built from Web services may be distributed across a network with different services running on a variety of platforms. These services need a way to find one another, need to know how to invoke each other, and need a mechanism for exchanging information. These needs are satisfied by UDDI, WSDL, and SOAP. A typical Web Service invocation using these elements is shown in Figure 7.

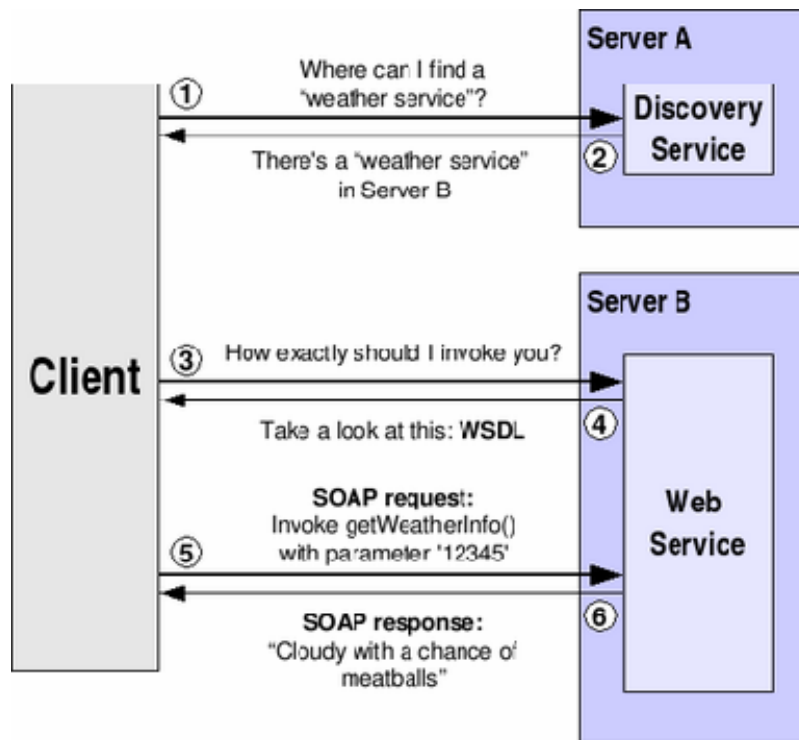


Figure 7 Web Service Invocation, Sotomayor, B. (2005), *Globus Toolkit 4 Programmer's Tutorial*, retrieved May 29, 2008 from: <http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html>

The steps in the invocation seen above are:

1. A client may not have foreknowledge of the Web Service it is going to invoke. The first step is to discover a Web Service that meets its requirements. For this example, it contacts a discovery service (which is itself a Web Service).

2. The discovery service replies, typically with a response containing a URL, telling the client that a weather service is available on Server B.
3. The client sends a message to the weather service to determine how to actually invoke it. Per Sotomayor, "we know it can give me the forecast for a US city, but how do we perform the actual service invocation? The method I have to invoke might be called 'string getCityForecast(int CityPostalCode)', but it could also be called 'string getUSCityWeather(string cityName, bool isFahrenheit)'. We have to ask the Web Service to describe itself (i.e. tell us how exactly we should invoke it)."
4. The Web Service replies using WSDL.
5. The client invokes the web service using a SOAP request.
6. The Web Service replies with a SOAP response which includes the forecast or an error message if the SOAP request was incorrect.¹⁷

Following is an example, provided by W3Schools, of a simple web service created using Microsoft's ASP.NET. It accepts a temperature in Fahrenheit or Celsius and returns the temperature in the other unit. This web service is an application written in VBScript. It would typically be hosted and run under Microsoft's Internet Information Services (IIS), which, in turn, would run under Windows Server 2003 or Windows Server 2008. More information on configuring an open source, Apache web server, or Microsoft IIS, to host Web services can be found later in this paper.

In this example we use ASP.NET to create a simple Web Service.

¹⁷ Sotomayor, B., *Globus Toolkit 4 Programmer's Tutorial*, retrieved May 29, 2008 from: <http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html>, 2005

```

<%@ WebService Language="VBScript" Class="TempConvert" %>
Imports System
Imports System.Web.Services
Public Class TempConvert :Inherits WebService
<WebMethod()> Public Function FahrenheitToCelsius
(ByVal Fahrenheit As String) As String
dim fahr
fahr=trim(replace(Fahrenheit,",", "."))
if fahr="" or IsNumeric(fahr)=false then return "Error"
return (((fahr) - 32) / 9) * 5)
end function
<WebMethod()> Public Function CelsiusToFahrenheit
(ByVal Celsius As String) As String
dim cel
cel=trim(replace(Celsius,",", "."))
if cel="" or IsNumeric(cel)=false then return "Error"
return (((cel) * 9) / 5) + 32)
end function
end class

```

This document is an .asmx file. This is the ASP.NET file extension for XML Web services.¹⁸

With the TempConvert coded and hosted on a web server, it can now be invoked from a client.

Recall that Web services are not invoked directly from a browser. However, a Web service can be invoked from html code, for example using a form, within a web page. An example of such a form as displayed in a browser and the html code to display the form and invoke the Web Service using HTTP POST when the user selects "Submit", follows.¹⁹

Use a form to access a Web Service.

Using a form and HTTP POST, you can put the web service on your site, like this:

Fahrenheit to Celsius:

Celsius to Fahrenheit:

¹⁸ W3 Schools, *Web Service Example*, retrieved June 20, 2008 from http://www.w3schools.com/webservices/ws_example.asp

¹⁹ W3 Schools, *Web Service Example*, retrieved June 20, 2008 from http://www.w3schools.com/webservices/ws_example.asp

Use a form to access the Web service.

Here is the code to add the Web service to a web page:

```
<form target="_blank" action='http://www.example.com
/webServices/tempconvert.asmx/FahrenheitToCelsius '
method="POST">
<table>
  <tr>
    <td>Fahrenheit to Celsius:</td>
    <td><input class="frmInput" type="text"
      size="30" name="Fahrenheit"></td>
  </tr>
  <tr>
    <td></td>
    <td align="right"> <input type="submit"
      value="Submit" class="button"></td>
  </tr>
</table>
</form>

<form target="_blank" action='http://www.example.com
/webServices/tempconvert.asmx/CelsiusToFahrenheit '
method="POST">
<table>
  <tr>
    <td>Celsius to Fahrenheit:</td>
    <td><input class="frmInput" type="text"
      size="30" name="Celsius"></td>
  </tr>
  <tr>
    <td></td>
    <td align="right"> <input type="submit"
      value="Submit" class="button"></td>
  </tr>
</table>
</form>
```

The example above assumes that the TempConvert Web service is hosted at example.com. The web page invoking the service need not be hosted at example.com. It could be hosted anywhere on the web. This illustrates the power of distributed applications using Web services.

Web services Taxonomy

Web services can provide a variety of capabilities. Erl notes that services can be categorized depending on:

- The type of logic they encapsulate
- The extent of reuse potential
- The relationship of the logic to domains (for example, sales, accounting, manufacturing) within the enterprise

Based on this categorization, three classifications or service models emerge:

- Entity Services
- Task Services
- Utility Services²⁰

These models are further explained in the following paragraphs.

An entity service encapsulates logic that defines an entity within an enterprise. Examples of business entities include customer, employee, and invoice. These services typically offer create, read, update, and delete (CRUD) methods. They are highly reusable. For example, a service that provides and maintains information about a customer could be used in sales, customer service, order processing, delivery, and accounting applications. Entity services are in many ways analogous to object-oriented architecture "objects". Figure 8 depicts an entity service that exposes typical CRUD methods.²¹

²⁰ Erl, T., *SOA, Principles of Service Design*, Prentice Hall, Boston, 2008

²¹ Erl, T., *SOA, Principles of Service Design*, Prentice Hall, Boston, 2008

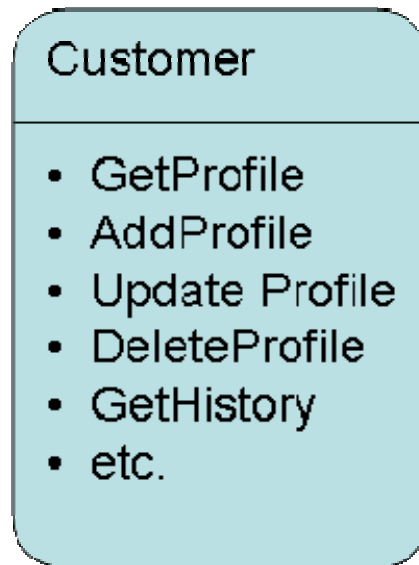


Figure 8 An Entity Service

A task service, in contrast, typically encapsulates logic associated with a business process. These services orchestrate, compose, or coordinate the efforts of entity services to perform a business process. An example of a task service is a billing consolidation application that retrieves many invoice and purchase order records, performs various calculations, and validates the results against historical client billing records. Figure 9 depicts a task service that performs revenue analysis and exposes a single, "submit" method.²²

²² Erl, T., *SOA, Principles of Service Design*, Prentice Hall, Boston, 2008

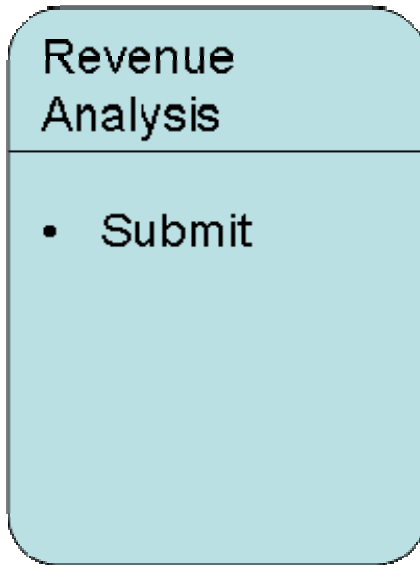


Figure 9 A Task Service

The final service classification is the utility service. This type of service typically does not encapsulate logic to represent a business entity or business process. Rather, it typically implements non-business technological functionality. Utility services are also known as application services, infrastructure services, or technology services.²³ A utility service, for example, might convert Microsoft Word documents to Portable Document Format (PDF). Figure 10 depicts a Transform Document utility service that exposes methods to import documents in Microsoft Word, PowerPoint, or Project format and a method to export documents in PDF.

²³ Erl, T., *SOA, Principles of Service Design*, Prentice Hall, Boston, 2008

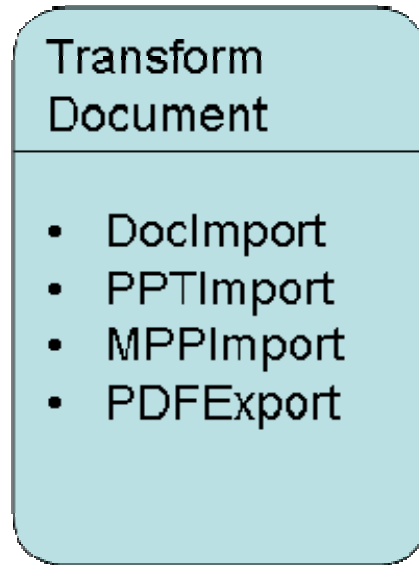


Figure 10 A Utility Service

Different types of web services: end-points, routers, load balancers, etc. Give enough background to enable understanding of ESB concept.

Primary SOA Building Blocks: SOAP, UDDI, and WSDL

SOAP

The SOAP specification defines the standard message format used in most SOA implementations. SOAP originally stood for "Simple Object Access Protocol." As of version 1.2, "SOAP" is just a standalone term. A SOAP message consists of three parts. SOAP messages are packaged into "envelopes." Each message may optionally contain a header holding meta information. Though optional, header blocks are vital to most service-oriented solutions. Most of the WS-* extensions are implemented using header blocks. Examples of these features are:

- Processing instructions that may be executed by intermediaries or the receiver

- Routing or workflow information
- Security measures, including encryption and electronic signatures, implemented in the message
- Reliability rules associated with the delivery of the message, for example, by using the WS-ReliableMessaging protocol
- Context and transaction management information
- Correlation information, i.e. an identifier used to associate a response message to a specific request message.²⁴

The body of the SOAP message contains the "payload." The original SOAP specification was designed to replace proprietary Remote Procedure Call (RPC) protocols to allow calls between components to be serialized as XML documents. Much of the original specification was developed to accommodate RPC data. This RPC-style message is not favored for SOA. Instead, a document-style message is preferred. This style enables larger payloads and coarser interfaces and allowing the same level of interaction with fewer messages being passed between services.²⁵ Figure 11 and Figure 12 provide examples of RPC-Style and Document-Style SOAP invocations.

²⁴ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006, pp 143-145

²⁵ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006, pp 146-147

```

POST /RPC2 HTTP/1.0
Host: betty.userland.com
User-Agent: Frontier/5.1.2 (WinNT)
Content-Type: text/xml
Content-length: 181
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>

```

Figure 11 RPC-Style SOAP Invocation, from: Hollinger, D., (2004) *Lecture Notes for CSCI-4220, Network Programming*, retrieved from: <http://www.cs.rpi.edu/~hollingd/netprog/> June 3, 2008

```

POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Figure 12 Document-Style SOAP Invocation, Hollinger, D., (2004) *Lecture Notes for CSCI-4220, Network Programming*, retrieved from: <http://www.cs.rpi.edu/~hollingd/netprog/> June 3, 2008

UDDI

UDDI allows software services to advertise themselves either as free or on a fee-per-use basis. UDDI is the broker component of Web services that allows service providers and requesters to locate each other. UDDI is implemented as a distributed database with interconnected servers. UDDI offers three search options.

1. White Pages allow users to search for Web services by name
2. Yellow pages allow users to search for Web services by type
3. Green Pages specify the interfaces on which a Web Service will respond, the method calls it will accept, properties that it can send or return, and payment methods.²⁶

The operation of UDDI is depicted in Figure 13.

²⁶ Panko, R., *Business Data Networks and Telecommunications*, 6th Edition, pg 520.

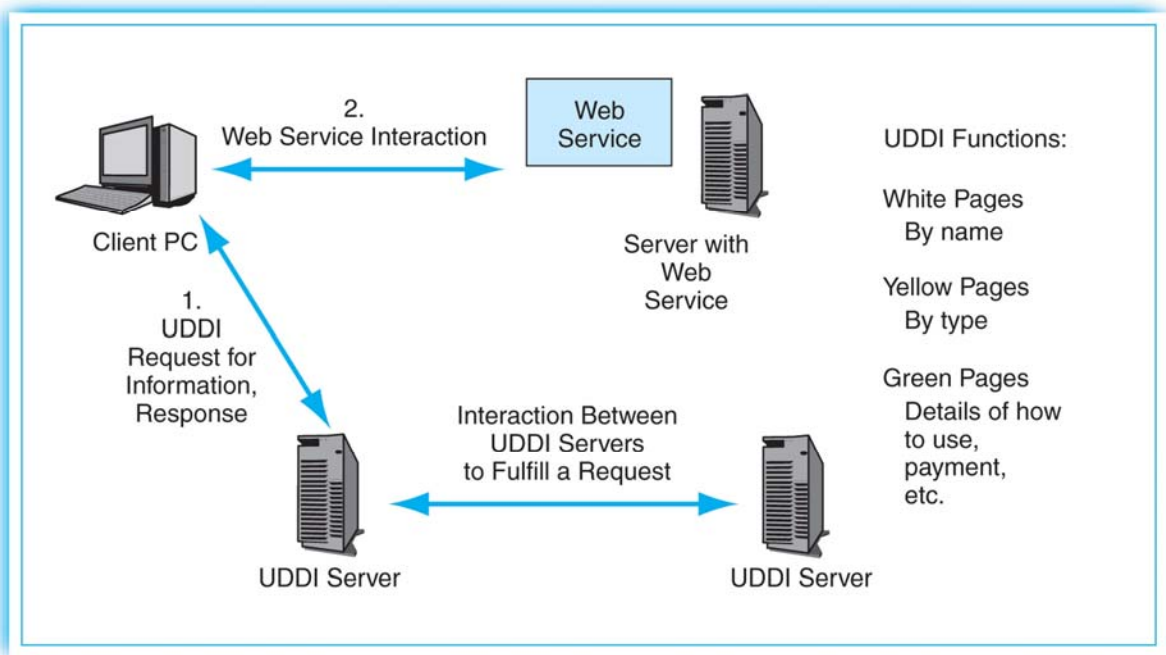


Figure 13 UDDI Services for Web services, Source: Panko, R., *Business Data Networks and Telecommunications, 6th Edition*, pg 520.

The information in a UDDI registry is contained in five types of data structures. These are:

1. `businessEntity`, which contains information about the entity who provides a family of services
2. `businessService`, which contains information about a particular service
3. `bindingTemplate`, which provides information about the service entry point, typically a URL, and technical details for accessing the entry point
4. `tModel`, which provides a reference to the specifications for the services. This is an abstract concept that will be further explained.

5. publisherAssertion, which provides information about a relationship between two parties.²⁷

The relationships between these structures are depicted in Figure 14.

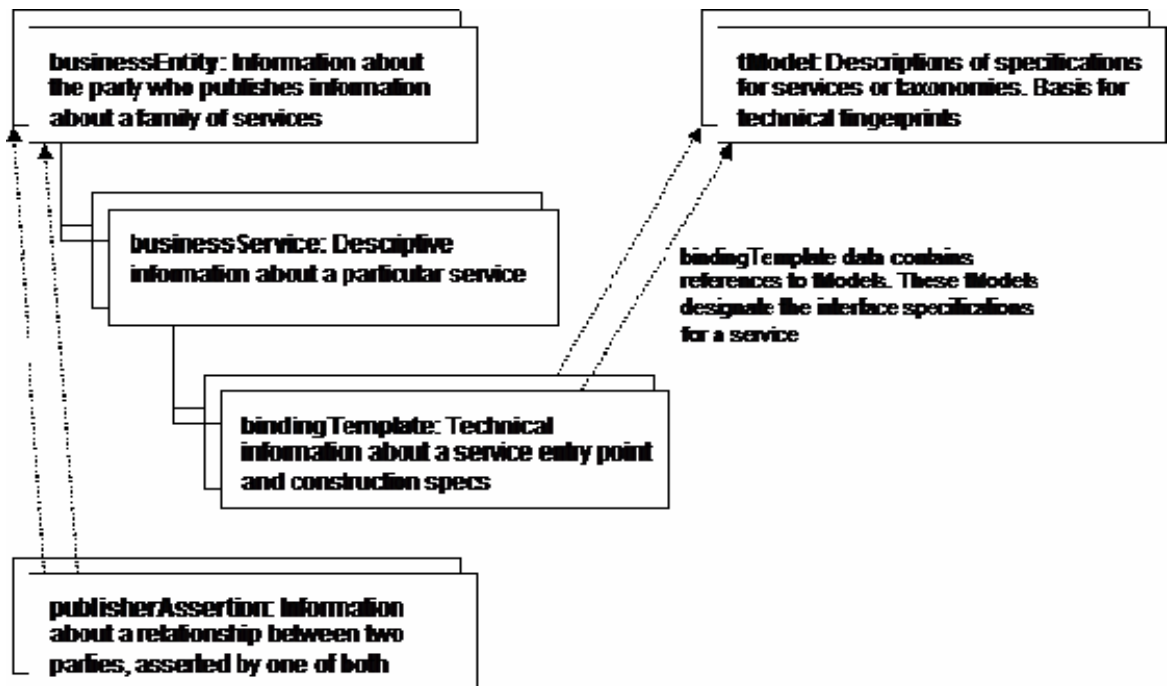


Figure 14 Information Contained in a UDDI Registry, from OASIS, (2003), *UDDI Version 2.03 Data Structure Reference*, Retrieved July 3, 2008 from: <http://www.uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm>

The intent of tModel is to allow automated discovery of services. This is a difficult problem because much may depend on the context. The tModel may refer to a shared namespace. This would be appropriate for vertical integration of a supply chain. The

²⁷ OASIS, (2003), *UDDI Version 2.03 Data Structure Reference*, Retrieved July 3, 2008 from:

<http://www.uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm>

participants in the implementation would likely be able to agree and standardize a nomenclature for the taxonomy of services to be provided by suppliers. In other applications, other specifications for naming of services might be referenced. According to OASIS:

Being able to describe a Web service and then make the description meaningful enough to be useful during searches is an important UDDI goal. Another goal is to provide a facility to make these descriptions useful enough to learn about how to interact with a service that you don't know much about. In order to do this, there needs to be a way to mark a description with information that designates how it behaves, what conventions it follows, or what specifications or standards the service is compliant with. Providing the ability to describe compliance with a specification, concept, or even a shared design is one of the roles that the tModel structure fills.

The tModel structure takes the form of keyed metadata (data about data). In a general sense, the purpose of a tModel within the UDDI registry is to provide a reference system based on abstraction. Thus, the kind of data that a tModel represents is pretty nebulous. In other words, a tModel registration can define just about anything, but in the current revision, two conventions have been applied for using tModels: as sources for determining compatibility and as keyed namespace references.

The information that makes up a tModel is quite simple. There's a key, a name, an optional description, and then a URL that points somewhere – presumably somewhere where the curious can go to find out more about the actual concept represented by the metadata in the tModel itself.²⁸

UDDI was originally intended to function on the web with a public implementation much like Domain Name Service (DNS). Microsoft, IBM, and SAP sponsored the UDDI Business Registry (UBR) project towards this end. The project was discontinued in 2005 when the

²⁸ OASIS, (2003), *UDDI Version 2.03 Data Structure Reference*, Retrieved July 3, 2008 from:

<http://www.uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm>

participants decided that the original vision was no longer valid. The original vision for UDDI was as a standard that would enable companies to conduct business with each other automatically over the web. Companies could publish how they wanted to interact and other companies could find that information and use it to establish a relationship. Typically, however, companies form relationships through human interactions. So, the automated UBR saw little use. However, the UBR did serve as an interoperability reference implementation. Through its evolution UDDI evolved to become the metadata management standard for SOA."²⁹

WSDL

WSDL enables the communication between service requestors and service providers. WSDL is an XML document that describes the endpoints provided by a service. WSDL describes both the physical address and the endpoint interface. The endpoint interface description provides the network protocol and message format. Figure 15 shows an example of a WSDL definition for a service that provides stock quotes. The service provides a single operation, `GetLastTradePrice`, which can be accessed using SOAP 1.1 over HTTP. To use the interface, the requestor provides a ticker symbol of type string. The service returns the price as a float.

²⁹ Krill, P. (2005). Microsoft, IBM, SAP discontinue UDDI registry effort, *InfoWorld*, Retrieved May 8, 2008, from http://www.infoworld.com/article/05/12/16/HNuddishut_1.html

```

<?xml version="1.0"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd1="http://example.com/stockquote.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
        <complexType>
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="TradePrice">
        <complexType>
          <all>
            <element name="price" type="float"/>
          </all>
        </complexType>
      </element>
    </schema>
  </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>

  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>

  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
      <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="StockQuoteService">

```

Figure 15 Stock Quote WSDL, Christensen, E., Web Services Description Language (WSDL)

1.1, retrieved June 15, 2008 from: <http://www.w3.org/TR/wsdl>

Note that the data types, abstract service, and concrete service bindings are specified in the preceding example. A more flexible approach is to define each of these separately and use the "import" statement to pull in the definitions. This approach improves readability and promotes flexibility and reuse. Figure 16 shows the same interface defined through the use of three different files. Note particularly the use of an XML Schema Definition (XSD) or .xsd file.

```

http://example.com/stockquote/stockquote.xsd

<?xml version="1.0"?>
<schema targetNamespace="http://example.com/stockquote/schemas"
  xmlns="http://www.w3.org/2000/10/XMLSchema">

  <element name="TradePriceRequest">
    <complexType>
      <all>
        <element name="tickerSymbol" type="string"/>
      </all>
    </complexType>
  </element>
  <element name="TradePrice">
    <complexType>
      <all>
        <element name="price" type="float"/>
      </all>
    </complexType>
  </element>
</schema>

http://example.com/stockquote/stockquote.wsdl

<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote/definitions"
  xmlns:tns="http://example.com/stockquote/definitions"
  xmlns:xsd1="http://example.com/stockquote/schemas"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <import namespace="http://example.com/stockquote/schemas"
    location="http://example.com/stockquote/stockquote.xsd"/>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>

  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>
</definitions>

http://example.com/stockquote/stockquoteservice.wsdl

<?xml version="1.0"?>
<definitions name="StockQuote"

```

Figure 16 WSDL with XML Schema Definition, from Christensen, E., Web Services

Description Language (WSDL) 1.1, retrieved June 15, 2008 from: <http://www.w3.org/TR/wsdl>

Web services Extensions

Web services technology is evolving to extend the basic capabilities of HTTP. Extended Web services standards, the WS-* standards, provide the specifications for using these capabilities. This technology is developing rapidly. Existing vendor products may partially implement these standards and may include non-standard vendor-specific features.³⁰ It is important that the SOA practitioner understand the extent to which these WS-* extensions are supported when using SOA products from multiple vendors. Extended Web services standards important to SOA include proposed or adopted standards for metadata management, security, reliability, notification and transaction processing. WS-* specifications that describe these functionalities are discussed in the following paragraphs.

Metadata is data about data. A Web service has a variety of associated metadata. This metadata includes data types and structures for messages, message exchange patterns, network addresses (URIs) of endpoints that exchange messages, and requirements for extended features such as security, reliability, or transactions. Web services metadata management technologies include the previously discussed UDDI and XML Schema as well as the extensions WS-Policy, WS-Addressing, and WS-Exchange.³¹

WS-Policy allows a service provider to instruct a service requester to follow certain policies. Policies can be expressed as simple rules, for example, "Strong authentication using Kerberos is required," or as a set of rules evaluated in priority order that determine if a requester

³⁰ Newcomer, E. (2005), *Understanding SOA with Web Services*, Upper Saddle River, NJ: Addison-Wesley, page 271

³¹ *Ibid*, page 273

will be allowed to interact with the provider. Three competing specifications exist for expressing policies. These are WS-Policy Framework, developed by BEA, IBM, Microsoft, and SAP; Web Services Policy Language (WSPL), created by a subcommittee of the eXtensible Access Control Markup Language (XACML) Technical Committee at OASIS and WSDL 3.2 features and properties, created by the WSDL Working Group at W3C.

WS-Addressing was authored by Microsoft, IBM, BEA, Sun, and SAP and submitted to W3C for approval as a standard. The WS-Addressing specification provides important functionality in an SOA. WS-Addressing adds the concept of an endpoint reference. An endpoint reference identifies a specific instance of a service. Being able to identify a specific instance of a service allows scalability. For example, a service "A" sends a message to service "B". "B" replies to "A" requesting further information. If there are multiple instances of "B" deployed to provide the necessary throughput for a system, it is necessary that "A" reply to the specific instance of "B" that contains the state data from the earlier communication. WS-Addressing provides this capability. WS-Addressing is implemented through the addition of a Message Information (MI) header to a SOAP message. The MI header may contain:

- Destination: The address to which the message is being sent
- Source Endpoint: The endpoint reference to the service that generated the message
- Reply Endpoint: The address to which a reply should be sent
- Fault Endpoint: The address to which an error notification should be sent
- Message ID: A value that uniquely identifies the message (required if reply endpoint is used)

- Relationship: Depends on context, typically contains the message id of the related message to which this message is a response
- Action: A URI value that indicates the message's overall purpose. It can be used to identify the operation to be invoked upon receiving a request message

With WS-Addressing the endpoint address is contained in the SOAP message. This allows the message to be passed independently of addressing mechanism used by the transport protocol.³²

Closely related to WS-Addressing is the specification WS-ReliableMessaging. WS-ReliableMessaging is implemented through additions to the SOAP header. These additions include:

- Sequence
- MessageNumber
- LastMessage
- SequenceAcknowledgement
- AcknowledgementRange
- Nack
- AckRequested

These extensions can be used to guarantee that service providers are notified of the success or failure of message transmission and that messages that are intended to be in a specific sequence will arrive as intended or will generate a fault condition.³³

³² Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006

³³ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006

The WS-MetadataExchange specification provides a mechanism for a service requestor to issue a standardized request for some or all of the metadata related to an endpoint. The original specification allowed for three types of requests, Get WSDL, Get Schema, and Get Policy. These are still the most commonly used requests. The specification was later revised to allow just one type of request, Get Metadata. The type of metadata requested is now included in the request. This allows greater flexibility. The ability to request metadata supports loose coupling between services and improves interoperability in dynamic environments. For example, if there are multiple versions of a service deployed, a requester can evaluate the capabilities of a candidate service provider to determine if the provider can provide the needed capabilities before the requester initiates a sequence of message exchanges.³⁴

The WS-Security specification provides a standardized framework for implementing identification, authentication, authorization, confidentiality and integrity. Along with XML-Signature and XML-Encryption, WS-Security provides a container within which other security standards are implemented. WS-Security is analogous to a smorgasbord that allows from a selection of a number of possible security implementations. Important security specifications within the framework include:

- WS-SecurityPolicy
- WS-Trust
- WS-SecureConversation
- WS-Federation
- Extensible Access Control Markup Language (XACML)

³⁴ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006

- Extensible Rights Markup Language (XrML)
- XML-KeyManagement (XKMS)
- XML-Signature
- XML-Encryption
- Security Assertion Markup Language (SAML)
- Microsoft .Net Passport
- Secure Sockets Layer (SSL)
- WS-I Basic Security Profile

Detailed information on these specifications can be found at www.soaspecs.com.³⁵

Two competing specifications provide publish and subscribe message services, WS-Notification, spearheaded by IBM, and WS-Eventing, sponsored by Microsoft. WS-Notification provides:

- WS-BaseNotification: A standardized interface used by publishers and subscribers
- WS-Topics: Governs the structure and categorization of topics
- WS-BrokeredNotification: Provides a broker intermediary to send and receive messages on behalf of publishers and subscribers

WS-Eventing implements event-oriented publish and subscribe functions. Requests supported are:

- Subscribe: Create a new subscription

³⁵ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006

- Unsubscribe: Cancel an existing subscription
- Renew: Extend an existing subscription that is set to expire
- GetStatus: Retrieve the status of a subscription³⁶

The WS-AtomicTransaction specification provides cross-service ACID transaction functionality. "ACID" is an acronym that represents the four characteristics of a traditional transaction:

1. Atomic: Either all of the changes within the scope of the transaction succeed or none of them succeed. This requires supporting a rollback feature
2. Consistent: None of the changes made as a result of a transaction can violate the validity of an associated data model. If one does, the transaction must be rolled back
3. Isolated: If multiple transactions occur concurrently each transaction must be guaranteed an isolated execution environment
4. Durable: Upon completion of a successful transaction the changes made are durable. They can survive failures of subsequent transactions.³⁷

This section has discussed commonly used WS-* extensions. There are many more than can be covered here. A very useful and comprehensive list of SOA specifications and the URLs for locating them on the web appears in pages 425-429 of Newcomer's book, *Understanding SOA with Web Services*.

³⁶ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006, pp 266-276

³⁷ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006, pp 186-200

SOA Platforms

To this point, Web services have been discussed somewhat in the abstract and without regard to how they are deployed on a network. This section discusses the two primary Web services implementations, Java 2 Platform Enterprise Edition (J2EE) and the Microsoft .NET framework. Both of these implementations include a development environment, a runtime environment for hosting the developed software, run time application program interfaces (APIs) that allow the developed software to interface with the run-time platform, and an operating system on which to deploy the runtime and which interfaces with the underlying hardware.

Java 2 Platform Enterprise Edition

The J2EE Platform includes a development and a runtime environment. It can run on a variety of operating systems and has an ecosystem of vendors who provide development tools, middleware products and server runtimes. Other major Java Platform implementations are the Standard Edition (J2SE) for desktop applications, and Micro Edition (J2ME) for mobile devices. The layers of the J2EE Platform that are relevant to SOA are depicted in Figure 17. Relevant specifications are:

- Java 2 Platform Enterprise Edition Specification: Establishes the distributed J2EE component architecture and provides foundation standards that product vendors must meet in order to claim J2EE compliance
- Java API for XML-based RPC (JAX-RPC): Defines the JAX-RPC environment and APIs and the JAX-RPC service endpoint.

- Web Services for J2EE: Defines the services architecture³⁸

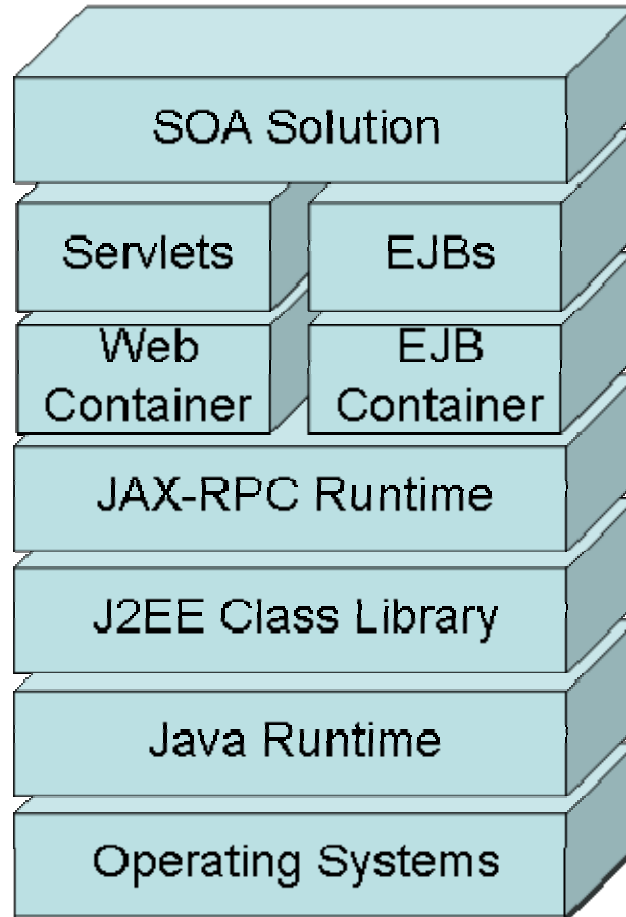


Figure 17 J2EE Platform

The following types of components can be used to build J2EE applications:

- Java Server Pages (JSPs): Dynamically generated web pages hosted by a web server, typically Apache. JSPs are text files comprised of code and html.
- Struts: An extension to J2EE that allows development of web applications with additional user-interface and navigation capabilities

³⁸ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall,

- Java Servlets: Compiled programs that reside on a web server and process HTTP requests and responses
- Enterprise Java Beans (EJBs): Business components deployed on application servers such as BEA's Weblogic, IBM's WebSphere, Red Hat's JBoss or Apache Foundation's Geronimo. Compared with web servers, application servers allow these services more flexibility to leverage features of middleware.³⁹

The J2EE environment runs on a Java runtime foundation. In support of Web services it provides additional APIs, primarily through the JAX-RPC, and two component containers, an EJB container and a web container. The web container is an extension to the web server which hosts JSPs or Java servlets. The EJB container hosts EJBs and provides additional services including transaction management, concurrency management, security, and object pooling. APIs provided include:

- Java API for XML Processing (JAXP): Provides XML parsing using several available parsers
- JAX-RPC: Provides SOAP processing
- JAX API for XML Registries (JAXR): Provides access to service registries including UDDI
- Java API for XML Messaging (JAXM): Provides asynchronous SOAP message transmission
- SOAP with Attachments API for Java (SAAJ): Provides management for SOAP with attachments following the SOAP with Attachments (SwA) standard

³⁹ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006

- Java Architecture for XML Binding (JAXB): Provides a mechanism for generating Java classes from XSD schemas
- Java Message Service (JMS): A Java messaging protocol which provides for reliable delivery

The power of these APIs and J2EE's ability to run on a variety of hardware platforms and operating systems and wide vendor support makes J2EE a popular platform for implementing SOAs.⁴⁰

Microsoft .Net Platform

The second major platform for SOAs is the proprietary Microsoft .Net platform. Components of the .Net architecture include ASP.Net, Web Services Enhancements (WSE), Assemblies, the .Net Class Library, and a Common Language Runtime (CLR), see Figure 18.

⁴⁰ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006

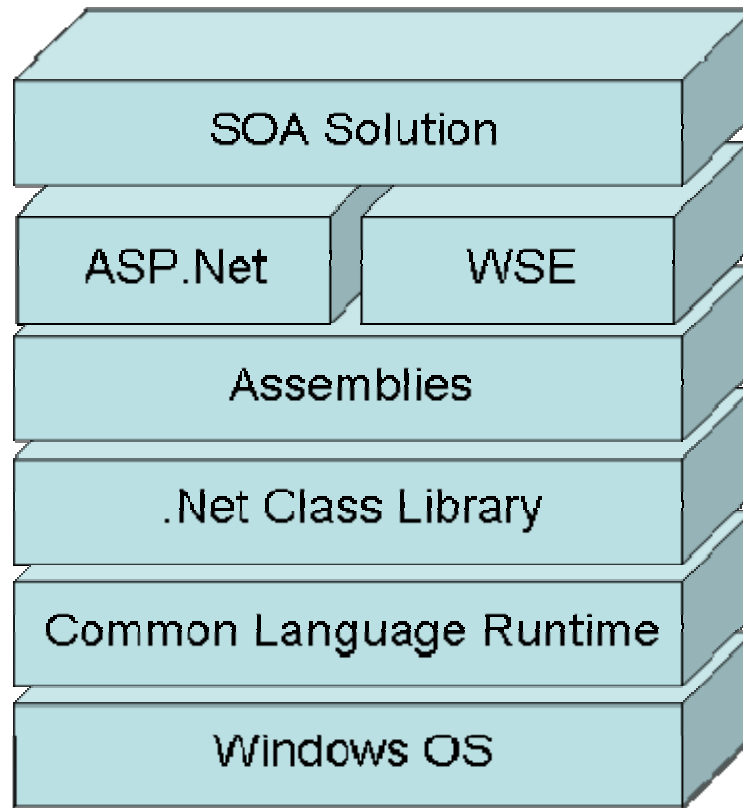


Figure 18 Microsoft .Net Platform

Components associated with SOA include:

- ASP.Net Web Forms: Dynamically built web pages that support the creation of online forms through server-side controls
- ASP.Net Web Services: An ASP.Net application designed as a service provider that runs on the web server
- Assemblies: A unit of processing logic within the .Net environment. An assembly can contain multiple classes and typically contains the application logic for a .Net web service
- CLR: Provides runtime services for managing .Net applications including cross language support, memory management, central data typing and object lifecycle management.

- Programming Languages: Unified support is provided for Visual Basic, Visual C++, Visual C# .Net. Programs written in any of these languages are converted into Microsoft Intermediate Language (MSIL). This is the code that is understood by the CLR

APIs provided by .Net include:

- System.XML: Provides parsing and processing for XML documents
- System.Web.Services: Provides a family of classes that support Web services including processing for WSDL and SOAP messages
- System.XML.Schema: Provides processing for XSD
- System.Web.Services.Discovery: Provides for programmatic discovery of Web service metadata

WSE is a toolkit that provides supplementary classes to implement WS-* functionality including WS-Addressing, WS-Policy and WS-Security. The ubiquity of the Windows operating system, the powerful Visual Studio development environment, and the tight integration with other Windows features, such as Active Directory, make ASP.Net a popular platform for developing and delivering SOA solutions.⁴¹

SOA as the Enterprise Architecture

SOA has been discussed in terms of the enabling technologies. Web services components, brokers, communications middleware, composition of services, and so on have been discussed. "SOA" can also refer to the deployment of these services in an enterprise architecture. IBM proposes a nine-layer reference model for an enterprise SOA, see Figure 19.

⁴¹ Erl, T., *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle River, NJ: Prentice Hall, 2006

⁴² It is instructive to compare IBM's proposed model with Pai's, seen previously in Figure 5. Both models share common elements but there are differences in the packaging. Pai sees the ESB, the SOA evolution of MOM, as part of the service tier. IBM sees the ESB as an all-embracing enabler of the SOA. Other similarities and differences are shown in Table 1. These similarities and differences are indicative of the state of the practice of SOA. There is a general consensus on the basic ideas and components of an SOA, but there is also a sense of competition as the various players in the industry strive to have their products and implementations "blessed" as the de facto standards.

Table 1 SOA Reference Architectures Compared

IBM Layer	Pai Tier	Comparison
No corresponding layer	Infrastructure	Pai explicitly identifies infrastructure as a component of an Enterprise SOA. Implicit in IBM architecture
Operational Layer	Application Tier	Both contain legacy applications.
Service Components, Services, and Business Process Layers	Service Tier	IBM separates Pai's Service Tier into three layers but notes that higher layers need not go through all intermediary layers to use a service in a lower level layer.

⁴² Arsanjani, A., et. al., (2007), *Design an SOA Solution Using a Reference Architecture*, retrieved June 17, 2008 from: <http://www.ibm.com/developerworks/library/ar-archtemp/>

IBM Layer	Pai Tier	Comparison
Consumers Layer	Web Application Tier	Pai emphasizes this as a presentation layer with applications being revealed to users through a web portal. IBM is vague regarding what this layer contains. Pai sees packaged applications as existing in this layer, though they are accessed through a web portal. IBM sees packaged applications being deeply buried in the Operational Layer with their functionality being accessed through web-enabling APIs. This difference may be due to IBM's large investment in legacy packages and applications.
Integration Layer	No corresponding tier	IBM sees the ESB as enabling the other layers. Pai sees it as a service in the Service Tier.
Quality of Service	Enterprise Security and Business Service Management	Pai separates security and management services into separate layers. IBM groups them together.
Information Architecture and Governance	No corresponding tier	IBM identifies two overarching layers. Not surprisingly, IBM offers products that correspond to these layers.

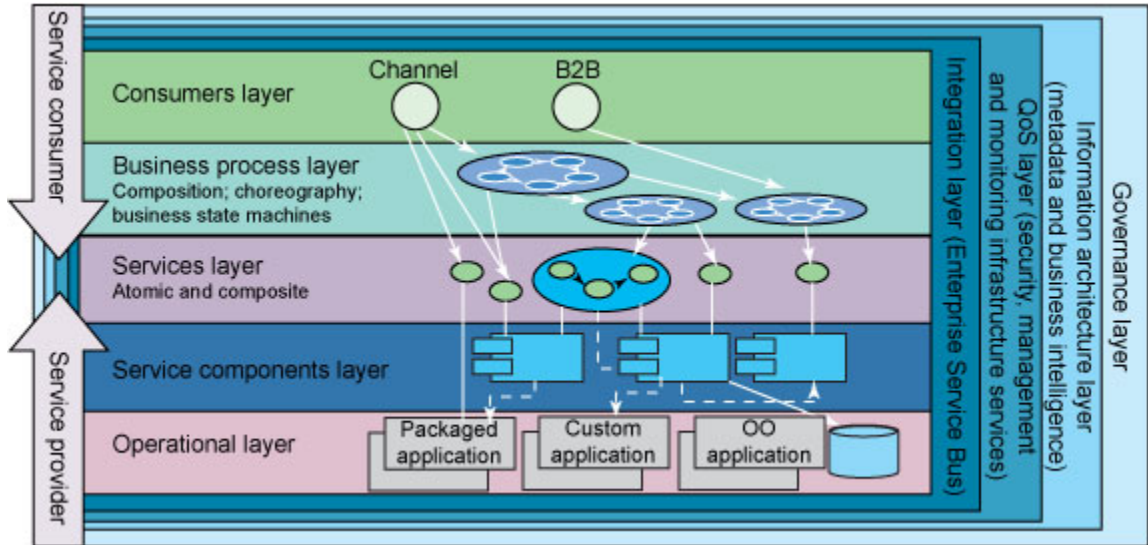


Figure 19 Layers of the Enterprise SOA, retrieved June 17, 2008 Arsanjani, A., et. al.

<http://www.ibm.com/developerworks/library/ar-archtemp/>

The layers in IBM's model are:

- Layer 1. Operational Layer: This layer includes custom and packaged applications including legacy software systems. It can be used to leverage existing IT investments in implementing an SOA solution. This reduces the cost of implementing an SOA solution.
- Layer 2. Service component layer: This layer contains software components which provide the implementation of a business process, utility process, or a task process.
- Layer 3. Services layer: This layer consists of all the services defined within the SOA. A service is an abstract specification of a collection of (one or more) business-aligned IT functions. The service's specification provides service users with sufficient detail to invoke the functions exposed by the service. The service

specification includes a description of the functionality offered by the service typically through WSDL.

- Layer 4. Business process layer: Compositions and choreographies of services in layer 3 are in this layer. Service composition through WS-BPEL is used to combine groups of services into composite applications.
- Layer 5. Consumer layer: The consumer layer is the presentation layer. It is typically implemented as a web portal. An recently adopted standard, Web Services for Remote Portlets (WSRP) Version 2.0 can be used to leverage Web services at the application interface or presentation level.
- Layer 6. Integration layer: The integration layer provides the capability to mediate, route, and transport service requests from the service requester to the appropriate service provider. This layer enables the integration of services through the introduction of a reliable set of capabilities. These include both point-to-point capabilities for tightly coupled integration and intelligent routing, protocol mediation, and other transformation mechanisms through an enterprise service bus (ESB).
- Layer 7. Quality of service layer: SOA has characteristics that raise QoS concerns. These characteristics are:
 - Increased virtualization
 - Loose coupling
 - Widespread use of XML
 - The composition of federated services
 - Heterogeneous computing infrastructures

- Decentralized SLAs
- The need to aggregate IT QoS metrics to produce business metrics

The QoS layer provides an SOA with the ability to meet Quantitative Performance Requirements (QPRs). It can capture, monitor, log, and signal noncompliance with QPRs. This layer observes the other layers and can raise alarms when a noncompliance condition is detected or anticipated.

- Layer 8. Information architecture and business intelligence layer: The information architecture and business intelligence layer enables the creation of business intelligence through data warehouses. Metadata is stored in this layer. This includes cross-industry and industry-specific data structures, XML-based metadata architectures and protocols for exchanging business data.
- Layer 9. Governance layer: The governance layer provides life-cycle management for the enterprise SOA. It provides policies for managing an SOA solution, including capacity, performance, security, and monitoring. It enables the SOA to be aligned with an enterprise's business priorities.⁴³

SOA seeks to align the IT initiatives of an enterprise with its business strategy.

Responsibility for SOA services is distributed across various functional organizations, both within and outside the enterprise. This can lead to an ill-coordinated "wild west" proliferation of services and sub-optimal use of IT resources. Governance is critical to avoiding this and achieving the benefits of the SOA vision. The need to ensure compliance with regulations such

⁴³ Arsanjani, A., et. al., (2007), *Design an SOA Solution Using a Reference Architecture*, retrieved June 17, 2008 from: <http://www.ibm.com/developerworks/library/ar-archtemp/>

as Sarbanes Oxley (SOX) and Health Insurance Portability and Accountability Act (HIPAA) also drives the need for SOA governance. SOA governance seeks to identify, specify, create and deploy services across the enterprise so as to maximize the alignment of IT initiatives with business strategies. Governance also identifies requirements across services for performance, security and reliability. These requirements may be implemented as Service Level Agreements (SLAs) or contracts between services using the WS-* extensions. The SOA governance council is responsible for establishing the standards used by the enterprise. The SOA governance council is responsible for overseeing the entire life cycle of an enterprise's service portfolio.

Designing for SOA

According to Gartner Research, an application is an SOA application if it implements all five of these principles:

1. It is modular.
2. The modules are distributable.
3. Software developers write or generate interface metadata that specifies an explicit contract so that another developer can find and use the service.
4. The interface of a service is separate from the implementation (code and data) of the service provider.
5. Services are shareable — that is, designed and deployed in a manner that enables them to be invoked successively by disparate consumers.

A good SOA application will also follow most of these guidelines:

- A service should be logically cohesive — that is, all the operations implemented in one service provider module should relate to the same purpose.

- In most cases, data used by a service provider should be encapsulated — that is, protected from direct external access.
- Most provider modules should be directly mapped to a business concept.
- In most large-scale SOA systems, service description metadata should be managed using a formal registry or repository, often using related industry standards such as XML, XSD and WSDL.
- Service metadata is generally better when it is more complete.
- Services should use standard protocols, such as XML/HTTP or SOAP/XML/HTTP, except where there are compelling performance or quality-of-service reasons why a proprietary protocol is more appropriate.
- Large, long-living or frequently changing SOA service communities should use enterprise service bus (ESB) technology to implement a communication and mediation backplane.⁴⁴

The guideline, "provider modules should ... directly map to a business concept," encapsulates the fundamental vision of SOA. Services that map directly to business processes can be composed in different ways to provide new applications as the enterprise grows and evolves. Use of a Business Process Modeling (BPM) tool allows orchestration of new service compositions to be performed using a graphical modeling tool. The output from a BPM tool is typically Web Services Business Process Execution Language (WS-BPEL). A WS-BPEL process can invoke other WS-BPEL processes and can also invoke itself recursively. WS-BPEL

⁴⁴ Schulte, R., (2006), Five Principles of SOA in Business and IT, Gartner Research, Stamford, CT

relies on WSDL and defines a basic set of tasks for creating Web Service compositions. These are:

- Invoke task: Invokes a one-way or request-response method on a Web service
- Receive task: Block while waiting for a message to arrive
- Reply task: Send a message in response to a received message
- Wait task: Wait for a specifiable period of time
- Assign task: Copy information from one place to another
- Throw task: Indicate that an error has occurred
- Terminate task: Terminate the orchestration instance

WS-BPEL structured tasks can then be used to combine basic tasks into more complex compositions. WS-BPEL structured tasks are:

- Sequence task: Defines an ordered sequence of tasks
- Switch task: Branch depending on conditional logic
- Pick task: Block and wait for a suitable message or a time-out. When one of these conditions occurs, execute an associated activity
- While task: Defines a group of tasks to be repeated while a condition is satisfied
- Flow task: Defines a collection of steps that are to be executed in parallel

WS-BPEL accesses services using the portTypes defined in WSDL documents. This allows the same, abstract business process definitions to be used in different deployments, so long as the services used in the deployments offer the same portTypes.⁴⁵

When performing SOA design, as when performing object-oriented design, it is useful to consider previously proven designs. In object-oriented programming, the "gang of four" design patterns are well known. Likewise design patterns are emerging in SOA. Thomas Erl surveyed SOA practitioners and identified nearly one hundred proven design patterns. The results of his work are available at his website, www.soapatterns.com.

Designing for SOA involves understanding and modeling business processes, defining atomic and composite services to implement the business processes, and providing the services technology infrastructure to execute and monitor the resulting composed services.

Reusing Legacy Applications in an SOA

A benefit to implementing SOA as an enterprise architecture is in that the implementation can be gradual and methodical. Legacy applications can be made to interact with new SOA components by providing a Web service gateway. A typical example of this is a Web service that accesses a mainframe IMS or CICS application. The interface to an IMS or CICS transaction is typically defined in Common Business Oriented Language (COBOL). To web-enable such an application, the COBOL is mapped to an XML schema. The gateway service is then defined using WSDL. The gateway component is then hosted on the mainframe or on a separate web server. The gateway component is responsible for handling service requests

⁴⁵ Newcomer, E. (2005), *Understanding SOA with Web Services*, Upper Saddle River, NJ: Addison-Wesley, page

in the form of incoming SOAP messages. It translates the SOAP message into the corresponding, fixed-format COBOL message then sends it to IMS. IBM provides a product, WebSphere MQ, that handles communication with CICS or IMS. The gateway component accepts the IMS or CICS response, converts it into a SOAP message, and routes it back to the service requester. Similarly, a Web service can be made to create input files, invoke a legacy batch process, and process the output files to handle a service request.

Conclusion

SOA is a rapidly evolving design paradigm that answers the needs of enterprises seeking to compete in today's rapidly changing global, Internet economy. SOA seeks to model business processes and package them as automated Web services. These services can then be composed in different ways and across different platforms to create new applications. SOA defines not only these new applications but also the IT infrastructure that allows these new applications to be hosted and deployed. SOA concepts rely upon and evolved from distributed and object-oriented programming concepts. These concepts are extended to allow integration of applications across an enterprise or amongst trading partners. The web-based technologies used in SOA are independent of platform and programming language. They can be implemented gradually and can incorporate legacy application functionality through web-enabling adapters. The SOA ecosystem is complex. Vendors are offering a variety of new products to enable SOA. These products are competing in the marketplace. As this competition unfolds, competing standards are gradually merged to become the de facto industry standard. As this evolution continues, SOA is rapidly becoming the design paradigm of choice for enterprise information technology.

References

Arsanjani, A., Zhang, L., Ellis, M., Allam, A., Channabasavaih, K., (2007), *Design an SOA*

Solution Using a Reference Architecture, retrieved June 17, 2008 from:

<http://www.ibm.com/developerworks/library/ar-archtemp/>

BEA Systems, Inc. (2007). BEA's SOA Reference Architecture - A Foundation for Business

Agility, San Jose, CA: BEA Systems, Inc.

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S. (2001), *Web Services Description*

Language (WSDL) 1.1, retrieved June 15, 2008 from: <http://www.w3.org/TR/wsdl>

Defining Technology Incorporated, (2008). *Basic MOM*, retrieved 26 May 2008 from:

<http://www.middleware.org/mom/basicmom.html>

Erl, T. (2006). *Service-Oriented Architecture, Concepts, Technology, and Design*, Upper Saddle

River, NJ: Prentice Hall

Erl, T. (2007). *SOA Specifications*, Retrieved June 18, 2008, from <http://www.soaspecs.com/>

Erl, T. (2008). *SOA, Principles of Service Design*, Upper Saddle River, NJ: Prentice Hall

Forrester Research (2008). Forrester Wave: SOA Service Life-Cycle Management, Q1 2008,

Cambridge, MA: Forrester Research

Haas, H. (2004). Web Services Glossary, retrieved May 29, 2008 from:

<http://www.w3.org/TR/ws-gloss/>, W3C

Hohpe, G., Woolf, B., Brown, K., D'Cruz, C., Fowler, M., Neville, S. Rettig, M., & Simon, J.

(2004). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging*

Solutions. Boston, MA: Addison-Wesley

Hollinger, D., (2004) *Lecture Notes for CSCI-4220, Network Programming*, retrieved June 3,

2008 from: <http://www.cs.rpi.edu/~hollingd/netprog/>

Hurwitz, J., Bloor, R., Baroudi, C., & Kaufman, M., (2007), *Service Oriented Architecture for Dummies*, John Wiley & Sons. Online from Books 24-7 through CTU subscription.

IDG Research Services Group (2007). *SOA Report 2007: Service-Oriented Architecture Graduates to the Enterprise*, Framingham, MA: IDG Research Services

International Business Machines (2008). *IBM Service Oriented Architecture (SOA)*, Retrieved June 18, 2008, from <http://www-306.ibm.com/software/solutions/soa/>

International Business Machines and Microsoft (2003). *Federation in a Web Services World*, Joint Whitepaper, Retrieved June 18, 2008, from <http://www-106.ibm.com/developerworks/library/ws-fedworld/>.

International Business Machines and Microsoft (2003). *Reliable Message Delivery in a Web services World*, Joint Whitepaper, Retrieved June 18, 2008, from <http://www-106.ibm.com/developerworks/library/ws-rmdev/>

International Business Machines and Microsoft (2002). *Security in a Web Services World*, Joint Whitepaper, Retrieved June 18, 2008, from <http://www-106.ibm.com/developerworks/library/ws-secmap/>

IONA Technologies (2007). *IONA SOA Solutions*, Retrieved June 18, 2008, from <http://www.iona.com/solutions/soa/welcome.htm>

Krill, P. (2005). Microsoft, IBM, SAP discontinue UDDI registry effort, *InfoWorld*, Retrieved May 8, 2008, from http://www.infoworld.com/article/05/12/16/HNuddishut_1.html

Larman, C. (2005). *Applying UML and Patterns, An Introduction to Object-Oriented Analysis and Design and Iterative Development*, Upper Saddle River, NJ: Prentice Hall

Newcomer, E., & Lomow, G. (2005). *Understanding SOA with Web Services*, Upper Saddle River, NJ: Addison-Wesley

- Organization for the Advancement of Structured Information Standards (2006). *OASIS Reference Model for Service Oriented Architecture V 1.0*, Retrieved June 18, 2008, from <http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf>
- Organization for the Advancement of Structured Information Standards (2004). *UDDI Executive Overview: Enabling Service-Oriented Architecture*, Retrieved May 8, 2008, from <http://uddi.xml.org/files/uddi-exec-wp.pdf>
- OASIS, (2003), *UDDI Version 2.03 Data Structure Reference*, Retrieved July 3, 2008 from: <http://www.uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm>
- Oracle Corporation (2008). *Service-Oriented Architecture Technology Center*, Retrieved June 18, 2008, from <http://www.oracle.com/technology/tech/soa/index.html>
- Pai, Y., (2006). *SOA Practitioners' Guide*, Retrieved June 2, 2008, from: <http://dev2dev.bea.com/2006/09/SOAPGPart1.pdf>
- Panko, R., (2007). *Business Data Networks and Telecommunications, Sixth Edition*, Upper Saddle River, NJ: Prentice Hall
- Pavlik, G., (2007). *Next-Generation SOA Infrastructure*, Oracle Corporation, Redwood Shores, CA.
- Roshen, W. (2008), *Services-Based Enterprise Integration Patterns Made Easy*, Retrieved 26 May 2008 from: <http://www.ibm.com/developerworks/webservices/library/ws-intpatterns/index.html>
- Schulte, R., (2006), *Five Principles of SOA in Business and IT*, Gartner Research, Stamford, CT
- Sommerville, I. (2006). *Software Engineering*, 8th edition. Addison-Wesley, Essex
- Sotomayor, B. (2005), *Globus Toolkit 4 Programmer's Tutorial*, retrieved May 29, 2008 from: <http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html>

WC3, (2008). *Extensible Markup Language (XML)*, Retrieved April 24, 2008, from
<http://www.w3.org/XML/>