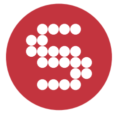

systinet

The Web Services Infrastructure Company

Introduction to Web Services Architecture

Introduction to Web Services Architecture

Table of Contents

Executive Summary 3

What are Web Services? 4

Distilling Common Themes 5

Dissecting the Name 5

Service Oriented Architecture 6

SOA Functional Architecture Components 6

SOA Systems 7

Vertical Technologies 7

Internet Middleware 8

Web Services Architecture 9

Transport 9

Description 10

Discovery 11

Alternate Discovery Mechanisms 13

Extending the Basic Web Services Architecture 13

WSA Invocation Mechanisms 16

Implementing Web Services Architecture 17

Systinet WASP 20

About Systinet 22

Copyright 2002, Systinet Corp.

Executive Summary

Web services appear to be the latest "new thing". But what are Web services? If you ask five people to define Web services, you'll probably get at least six answers. Even so, most people will agree that a Web service represents an information resource or business process that can be accessed over the Web by another application, and that it communicates using standard Internet protocols.

What distinguishes Web services from other types of Web-based applications is that Web services are designed to support application-to-application communication. Other Web applications support human-to-human communication (email and instant messaging) or human-to-application communication (browsers). Web services are designed to allow applications to communicate without human assistance or intervention.

Even though Web services are new, from an architectural perspective, they are based on established middleware design principles for application-to-application communication. These design principles are known as the service-oriented architecture (SOA). Previous SOA systems include RPC, RMI, DCOM, and CORBA. The Web services architecture (WSA) represents the convergence of SOA and "the Web".
The Web makes WSA completely platform- and language-independent. Web services can be developed using any language, and they can be deployed on any platform: from the tiniest device to the largest super computer.

Another way to think about Web services is as Internet middleware. Today most Web services are implemented using a core set of Internet middleware technologies including:

- XML, which provides a platform-neutral mechanism to represent data
- SOAP, which defines the data communication protocol for Web services
- WSDL, which describes a Web Service
- UDDI, which provides a means to advertise and discover Web services

A Web services platform is a set of products that implement these Internet middleware technologies. When selecting a Web services platform, you should keep in mind a number of factors, including:

- Language and platform support
- Performance and scalability
- Security
- Manageability

Copyright 2002, Systinet Corp.

Systinet Web Applications and Services Platform (WASP) provides the fastest, most scalable, most secure, most advanced Web services solution in the industry. WASP goes far beyond supplying just a basic Web services platform. WASP is designed to support the rigorous requirements of production-class application systems. WASP performs 5 to 10 times faster than other Web services solutions. WASP's optimized resource management services also ensure excellent scalability. WASP is the only Web services platform to support an integrated, application-level, end-to-end security framework. And WASP provides a comprehensive management infrastructure to coordinate your entire Web services system.

WASP, which supports Java and C++, is designed to fit into your existing configuration. WASP is portable across all leading operating systems, application servers, and databases.

What are Web Services?

If you ask five people to define Web services, you'll probably get at least six answers. In February 2002, the W3C Web Services Architecture Working Group (consisting of 75 members) exchanged nearly 400 emails over two weeks trying to define the term. They eventually abandoned the effort to achieve consensus.

For what it's worth, the last definition that the team produced was, "A Web service is a software application or component identified by a URI, whose interfaces and binding are capable of being described by standard formats and supports direct interactions with other software applications or components via Internet-based protocols".[1]

Here are a few more definitions from various industry sources:

"Web services are loosely coupled software components delivered over standard Internet technologies."

- Daryl Plummer, Gartner[2]

[Web services are] "Loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols."

- Brent Sleeper and Bill Robins, Stencil Group[3]

"A Web service is any piece of software that makes itself available over the Internet and uses a standardized XML messaging system."

- Ethan Cerami, author of *Web Services Essentials*[4]

[1] From the W3C WS-Arch discussion list: http://lists.w3.org/Archives/Public/www-ws-arch/2002Mar/0028.html

[2] UDDI Advisory Group presentation, June 2000.

[3] "Defining Web Services": http://www.stencilgroup.com/ideas_scope_200106wsdefined.html

[4] Web Services FAQ: http://www.oreillynet.com/pub/a/webservices/2002/02/12/webservicefaqs.html

Copyright 2002, Systinet Corp.

"From a technical point of view, Web services can be seen as an application API that can be invoked by an Uniform Resource Locator (URL)."

- Urban Bettag, Reuters[5]

"I think of a "Web service" as an arbitrarily grouped set of resources intended for machine (not human) manipulation, as a "Web site" is an arbitrarily grouped set of resources intended for human manipulation."

- Paul Prescod, co-author of *The XML Handbook* and W3C participant[6]

Distilling Common Themes

These definitions share a number of common themes. Everyone seems to agree that Web services are accessed via Internet protocols. There appears to be general consensus that a Web service represents a piece of software or perhaps a software component, although one definition asserts that the Web service is the interface rather than the actual software. Another recurring theme is that Web services are intended to be consumed by software rather than by humans. A couple of other points appear more than once: Web services are loosely coupled, and Web services are identified by a URI and accessed by a URL. It's interesting to note that only one of these definitions makes an association between Web services and XML, and no one seems willing to tie the *definition* of Web services to any specific Web services-related technology, such as SOAP. Web services represent an architecture – not a particular set of technologies.

Dissecting the Name

Another way to look for a definition is to analyze the constituent parts of the name "Web service".

- The *Web* refers to an immense information space that enables access to *Web resources*. A Web resource is some type of electronic construct, such as a file, network, processor, application, or service. Every Web resource is identified by its URI and accessed via Web protocols.

- A *service* is a resource that exposes its functionality through a programmatic interface, which means that it's designed to be consumed by software rather than by humans. The method of invocation and the possible results of that invocation are described by a contract.

- A *Web service*, therefore, is a service that is identified by a URI and can be accessed by applications via Web protocols in accordance with the contract that describes its programmatic interface.

[5] W3C Web Services Workshop Positioning Paper: http://www.w3.org/2001/03/WSWS-popa/paper03

[6] From the W3C XMLP discussion list: http://lists.w3.org/Archives/Public/xml-dist-app/2002Jun/0038.html

Copyright 2002, Systinet Corp.

Service Oriented Architecture

The concept of a *service* is key to understanding Web services. A Web services environment conforms to a Service Oriented Architecture (SOA). Figure 1 depicts the conceptual roles and operations of an SOA. The three basic roles are the service *provider*, the service *consumer*, and a service *broker*. A service provider makes the service available and publishes the contract that describes its interface. It then *registers* the service with a service broker. A service consumer queries the service broker and *finds* a compatible service. The service broker gives the service consumer directions on where to find the service and its service contract. The service consumer uses the contract to *bind* the client to the service.

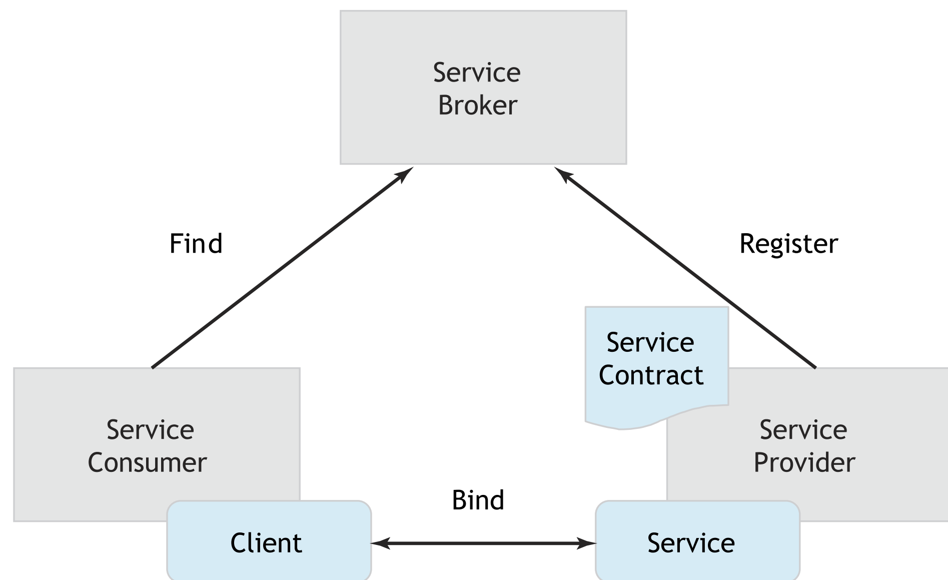

Figure 1. The three conceptual roles and operations of a service oriented architecture.

SOA Functional Architecture Components

In order for the three conceptual roles to accomplish the three conceptual operations, an SOA system must supply three core functional architecture components:

- **Transport.** The transport component represents the formats and protocols used to communicate with a service. The data format specifies the data types and byte stream formats used to encode data within messages. The wire protocol specifies the mechanism used to package the encoded data into messages. The transfer protocol specifies the application semantics that control a message transfer. The transport protocol performs the actual message transfer.

- **Description:** The description component represents the languages used to describe a service. The description provides the information needed to bind to a service. At a minimum, a description language provides the means to specify the service contract, including the operations that it performs and the parameters or message formats that it exchanges. A description language is

Copyright 2002, Systinet Corp.

a machine-readable format that can be processed by a compiler to produce communication code, such as client proxies, server skeletons, stubs, and ties. These generated code fragments automate the connection between the application code and the communications process, insulating the application from the complexities of the underlying middleware.

- **Discovery:** The discovery component represents the mechanisms used to register or advertise a service and to find a service and its description. Discovery mechanisms may be used at compile time or at runtime. They may support static or dynamic binding.

SOA Systems

Although Web services are new, the concepts behind service-oriented systems have been around for quite a while. Most standard distributed computing middleware systems implement a SOA. Examples of earlier SOA systems are:

- **Java RMI**[7]: Java Remote Method Invocation
- **CORBA**[8]: The Object Management Group Common Object Request Broker Architecture
- **DCE**[9]: The Open Group Distributed Computing Environment
- **DCOM**[10]: Microsoft Distributed Component Object Model

Each SOA system defines a set of formats and protocols that implement the core SOA functions. An SOA system often also defines an invocation mechanism, which includes an application programming interface and a set of language bindings. Table 1 shows the different formats and protocols used by various SOA systems.

Vertical Technologies

Although it's beyond the scope of this paper to examine Java RMI, CORBA, DCE, and DCOM in detail, you'll notice that each of these middleware technologies has defined its own vertical set of formats and protocols to implement the core SOA functions. This approach ensures consistency among applications that share the same middleware, but prevents interoperability with applications that use different middleware. It also requires that every service producer and service consumer that engages in a conversation must have the appropriate middleware loaded on its machine.

[7] The Java RMI specification: http://java.sun.com/j2se/1.4/docs/guide/rmi/spec/rmiTOC.html

[8] The OMG CORBA/IIOP specification: http://www.omg.org/technology/documents/formal/corba_iiop.htm

[9] The Open Group DCE RPC specification: http://www.opengroup.org/onlinepubs/9629399/.

[10] DCOM is based on DCE RPC. DCOM information: http://www.microsoft.com/com/tech/dcom.asp

Copyright 2002, Systinet Corp.

Internet Middleware

You can think of Web services as a new form of middleware – let's call it Internet middleware. But unlike previous SOA systems, Internet middleware does not require an entirely new set of protocols. The most basic Web services protocol is the industry standard Extensible Markup Language (XML), which is used as the message data format, and is also used as the foundation for all other Web services protocols. Today most Internet middleware systems are implemented using a core set of technologies, including SOAP, WSDL, and UDDI. These technologies define the transport, description, and discovery mechanisms, respectively. We'll delve deeper into these technologies in the next section of this paper. Each of these technologies are defined and implemented in XML. One important ramification of the use of XML is that any application, written in any language, running on any platform, can interpret Web services messages, descriptions, and discovery mechanisms. No specific middleware technology needs to be available to converse using Web services. Any application can interpret a SOAP message using standard XML processing tools.

	Java RMI[11]	CORBA[12]	DCE[13]	Web Services
Invocation Mechanism	Java RMI	CORBA RMI	RPC	JAX-RPC, .NET, …
Data Format	Serialized Java	CDR	NDR	XML[14]
Wire Format	Stream	GIOP	PDU	SOAP[15]
Transfer Protocol	JRMP	IIOP	RPC CO	HTTP, SMTP, …
Interface Description	Java Interface	CORBA IDL	DCE IDL	WSDL[16]
Discovery Mechanism	Java Registry	COS naming	CDS	UDDI[17]

Table 1. A comparison of SOA formats and protocols.

[11] JRMP = Java Remote Method Protocol.

[12] ORB = Object Request Broker. CDR = Common Data Representation. GIOP = General Inter-ORB Protocol. IIOP= Internet Inter-ORB Protocol. IDL = Interface Definition Language. COS = CORBA Object Services.

[13] RPC = Remote Procedure Call. NDR = Network Data Representation. PDU = Protocol Data Units. RPC CO = RPC Connect-Oriented protocol. IDL = Interface Definition Language. CDS = Cell Directory Service.

[14] Extensible Markup Language 1.0, W3C Recommendation: http://www.w3.org/TR/REC-xml

[15] SOAP 1.2 Part 1, W3C Working Draft: http://www.w3.org/TR/soap12-part1/

[16] Web Services Description Language 1.1, W3C Note: http://www.w3.org/TR/wsdl

[17] Universal Description, Discovery and Integration specifications: http://www.uddi.org/specification.html

Copyright 2002, Systinet Corp.

Web Services Architecture

In most Internet middleware configurations, the three core functional components *(transport, description, and discovery)* in the Web Services Architecture (WSA) are implemented using SOAP, WSDL, and UDDI, respectively. Figure 2 shows the conceptual SOA architecture using these technologies. A UDDI registry plays the role of service *broker*. The *register* and *find* operations are implemented using the UDDI Inquiry and UDDI Publish APIs. A WSDL document describes the service contract, and is used to *bind* the client to the service. All transport functions are performed using SOAP. Let's take a closer look at the WSA transport, description, and discovery functions.

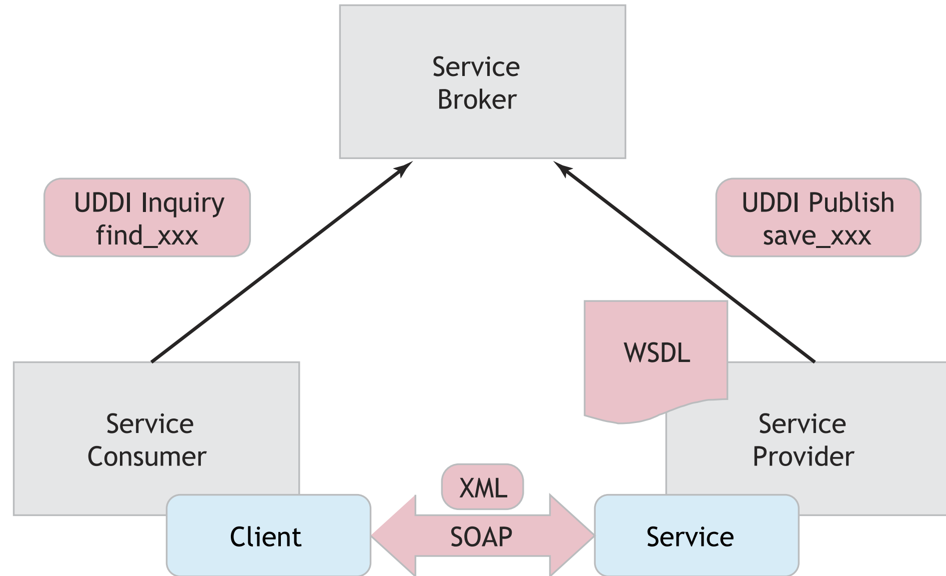

Figure 2. The SOA conceptual architecture with SOAP, WSDL, and UDDI.

Transport

The transport functional component defines the formats and protocols used to communicate between clients and services. The WSA formats and protocols are defined by SOAP, a lightweight, extensible XML protocol. SOAP provides a simple messaging framework that allows one application to send an XML message to another application.

- Data format. The SOAP data format is XML. The mechanism by which the data are encoded is totally extensible, and in fact can be specified within each SOAP message. The data format can represent an RPC invocation, in which case the message body is composed as a structure containing the RPC input parameters or return value. The name of the structure indicates the method to be invoked. When using the RPC representation, the data are normally encoded using an XML-based encoding style. Alternately, the data format can be in the form of an XML document, in which case the data are normally encoded using a specific XML Schema.

- Wire format. The SOAP wire format is an XML document called a SOAP envelope. The envelope contains an optional SOAP header and a mandatory

Copyright 2002, Systinet Corp.

SOAP body. The SOAP body contains the message payload, encoded in XML.

- **Transfer protocol**. SOAP defines an abstract binding framework that allows SOAP messages to be transferred using a variety of underlying transfer protocols. The SOAP specification defines a protocol binding for HTTP. Bindings have also been defined for HTTPS, SMTP, POP3, IMAP, JMS, and other protocols.

- **Extensions**. Additional information can be included with a SOAP message within a SOAP header. The SOAP header can provide directive or control information to the service, such as security information, transaction context, message correlation information, session indicators, or management information.

Description

The description functional component defines the language used to describe a service. The service consumer uses the description to bind the client to the service. The WSA description language is the Web Services Description Language (**WSDL**), a set of definitions expressed in XML. A WSDL document describes *what* functionality a Web service offers, *how* it communicates, and *where* to find it. The various parts of a Web service description can be separated into multiple documents to provide more flexibility and to increase reusability. Figure 3 maps the three parts of a WSDL description to the specific WSDL definition elements. A WSDL implementation document can be compiled to generate a client proxy that can call the Web service using SOAP.

- **Abstract interface**. The *what* part of a WSDL document describes the abstract interface of the Web service. It essentially describes a service *type*. Any number of service providers can implement the same service type. The WSDL *what* part defines a logical interface consisting of the set of operations that the service performs. For each operation it defines the input and/or output messages that are exchanged, the format of each message, and the data type of each element in the message.

- **Concrete binding**. The *how* part of a WSDL document describes a binding of the abstract interface to a concrete set of protocols. The binding indicates whether the message is structured as an RPC or as a document; it specifies which encoding style or XML Schema should be used to encode the data; it specifies which XML protocol should be used to construct the envelope; it indicates what header blocks should be included in the message; and it indicates which transfer protocol should be used. The *how* part includes or imports the associated WSDL *what* part.

- **Implementation**. The *where* part of a WSDL document describes a service implementation. A service implementation is a collection of one or more related ports. Each port implements a specific concrete binding of an abstract interface. The port specifies the access point of the service endpoint. A business might offer multiple access points to a particular service, each

Copyright 2002, Systinet Corp.

implementing a different binding. The *where* part includes or imports the associated WSDL *how* part. A service producer should always publish the *where* WSDL part with the Web service.

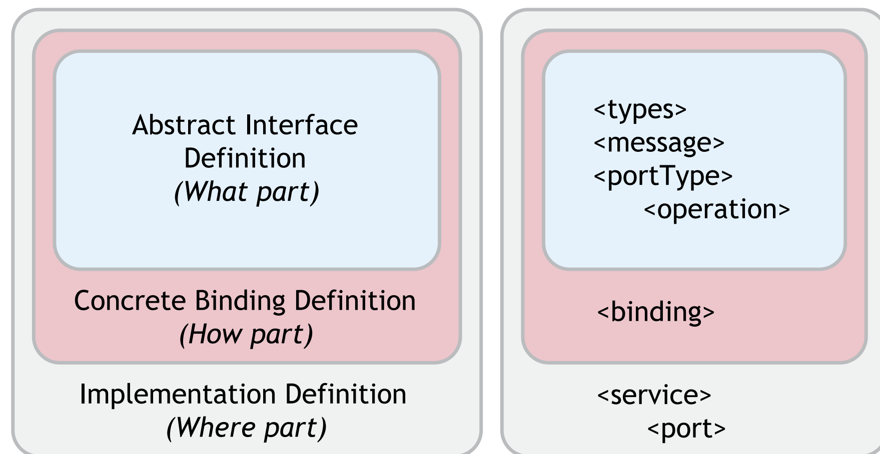

Figure 3. The three different parts of a WSDL description

Discovery

The discovery functional component provides a mechanism to register and find services. Some discovery functions are used at development time, while others are used at runtime. The WSA discovery mechanism is implemented using a Universal Description, Discovery and Integration (UDDI) registry service. For the most part, UDDI is used at development time, although it can also be used at runtime. UDDI is itself a Web service, and users communicate with UDDI using SOAP messages. UDDI manages information about service types and service providers, and it provides mechanisms to categorize, find, and bind to services.

- **Service types**. A service type, defined by a construct called a tModel, defines an abstract service. Multiple businesses can offer the same type of service, all supporting the same service interface. The tModel provides a pointer to the WSDL document that describes the abstract interface (the *what* part).

- **Service providers**. A service provider registers its business and all the services it offers. For each service offered, the service provider supplies the binding information needed to allow a consumer to bind to the service. The bindingTemplate construct provides a pointer to the WSDL document that describes the service binding (the *how* part). It also specifies the access point of the service implementation. The WSDL *where* part is usually co-resident with the access point.

- **Categorization**. When a service provider registers a service or service type, he can categorize the entity (business, service, or tModel) using a variety of taxonomies. The UDDI specification defines a core set of taxonomies, such as geographic location, product codes, and industry codes. Additional taxonomies can be added to the registry to support more focused or customized

Copyright 2002, Systinet Corp.

categorization and search.

- **Find**. When looking for a Web service, a service consumer queries the UDDI registry, searching for a business that offers the type of service that he wants. Users can search the registry for services by service type or service provider, and queries can be qualified using the taxonomies. From the tModel entry for the service type, the consumer can obtain the WSDL description describing the abstract interface. The consumer can compile this *what* part description to create a client interface for the abstract service. This abstract interface could be used to access multiple implementations of the service type. From the bindingTemplate entry for a specific service, the consumer can obtain the access point of the service and the WSDL description of the service binding.

- **Static binding**. Developers can bind clients to services either at compile time or at runtime. Using the WSDL *how* part, a developer can compile a concrete SOAP client interface or stub that implements the binding required to communicate with a specific Web service implementation. This pre-compiled stub can be included in a client application. The access point can be specified at runtime.

- **Dynamic binding**. Because a WSDL document is machine-readable, WSA also supports dynamic binding. Using just the WSDL *what* part at compile time, a developer can generate an abstract client interface that can work with any implementation of a specific service type. At runtime the client application can dynamically compile the WSDL *where* part (containing the *how* part) and generate a *dynamic proxy* that implements the binding.

- **Dynamic discovery**. Since UDDI is itself a Web service, an application can query the registry at runtime, dynamically discover a service, locate its access point, retrieve its WSDL, and bind to it, all at runtime. The client interprets the WSDL to dynamically construct the SOAP calls.

Figure 4 shows the relationship between UDDI constructs and WSDL parts. In this example, we have an industry standard specification for the XYZ service type. The XYZ service type is registered in UDDI using a tModel construct. The tModel points to a WSDL document that describes the abstract interface of this service type (a WSDL *what* part). There are two service providers (Company A and Company B) that implement the XYZ service type. Each service provider registers its business and its implementation of the XYZ service using the Business Entity, Business Service, and Binding Template constructs. The Binding Template points to the service implementation and to binding information for the service (a WSDL *how* part). Normally the WSDL *where* part is co-located with the service implementation and can be accessed by performing an HTTP GET on the access point URL or the URL appended with "?wsdl" or "/wsdl".

Copyright 2002, Systinet Corp.

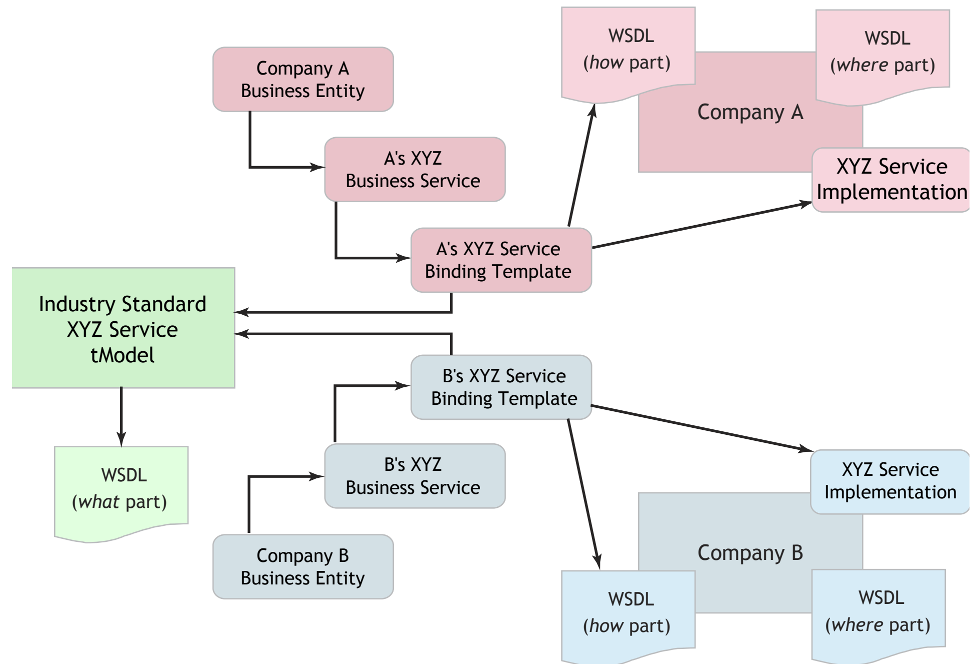

Figure 4. The relationship between UDDI constructs (tModel, businessEntity, businessService, and bindingTemplate) and WSDL parts (what, how, and where).

Alternate Discovery Mechanisms

UDDI is particularly useful if the service consumer doesn't know which type of Web service it wants to use, which service providers provide the service, or where to go to find the services. If a service consumer already knows this information, then the consumer can use a simpler, more direct form of discovery, such as the Web Services Inspection Language (WS-Inspection[18]). WS-Inspection is an XML format that can be used to inspect a Web site for available Web services. Assuming that the service consumer knows the home page URL for the service provider, and knows which service type it's looking for, the consumer can use WS-Inspection to find information (including WSDL *where* part descriptions) about all of the Web services offered by that service provider. Given a WSDL *where* part description, a service consumer has everything it needs to bind to the service.

Extending the Basic Web Services Architecture

As mentioned earlier, SOAP provides a built-in extension mechanism via SOAP headers. A SOAP header can be used to pass directive or control information between client and service to implement extended middleware functions, such as routing, intermediate caching, reliable delivery, asynchronous communications, stateful conversations, transactions, security, and management.

[18] The WS-Inspection specification: http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html

Copyright 2002, Systinet Corp.

To illustrate this extension mechanism, let's take a look at how you can use SOAP headers to support security. Security is a rather expansive topic. There are four different functions that fall under this topic:

- **Authentication and Proof of Identity**. Authentication is the process used to verify an entity's identity. There are a number of mechanisms that can be used to authenticate an entity, such as HTTP Basic and HTTP Digest authentication, a PKI certificate authority, and a Kerberos login. Once an authentication authority has verified your identity, you may receive an authentication token that you can use in future interactions as proof of identity. Such an authentication token could take the form of an X.509 certificate, a Kerberos ticket, or a SAML[19] authentication assertion.

- **Authorization and Access Control**. Authorization is the process used to determine if an authenticated entity has permission to perform a particular action or function. You may want to define access control policies for all services at a given location, for individual services, or for specific operations in a service.

- **Confidentiality and Integrity**. Encryption protects the confidentiality and integrity of message communication. Confidentiality prevents unauthorized access to the contents of the message. Integrity prevents unauthorized modification of the message.

- **Proof of Origin**. A digital signature provides proof that the signed data was sent from a specific authenticated identity. All or part of the message may be signed.

The WSA architecture supports security by allowing you to specify and exchange security information in a SOAP header. Figure 5 conceptually shows how an authentication token and a digital signature could be specified in a SOAP header. This diagram is based on the WS-Security[20] specification, which defines a set of SOAP header constructs that can be used to pass security information in SOAP messages. The WS-Security specification supports the following features:

- A way to pass authentication tokens.
- A mechanism to sign message content using XML Signature
- A way to pass signature and key information to allow the receiver to interpret the signed data

[19] The OASIS SAML (Security Assertions Markup Language) specification: http://www.oasis-open.org/committees/security/#documents

[20] WS-Security is being standardized at OASIS. http://www-106.ibm.com/developerworks/library/ws-secure/

Copyright 2002, Systinet Corp.

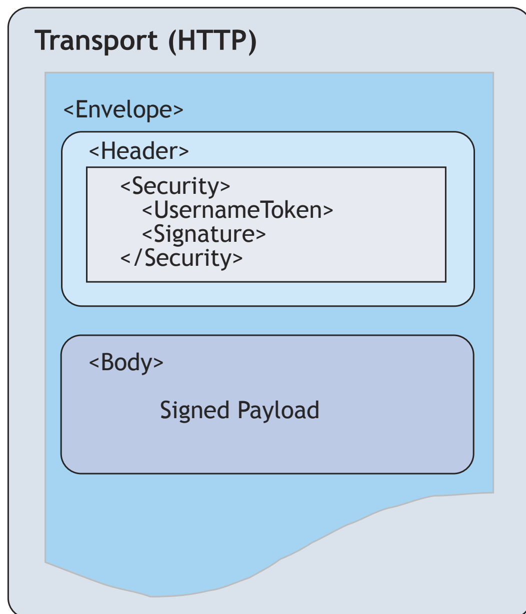

Figure 5. Using a SOAP Header to pass security information.

Other Middleware Capabilities

Many other middleware capabilities can be added to the WSA using SOAP headers, such as transactions, conversations, and reliable message delivery. Figure 6 shows a summary of the WSA functional architecture, indicating the capabilities that SOAP, WSDL, and UDDI provide for transport, description, and discovery. The transport delivery, context, security, and management functions are provided through SOAP extensions.

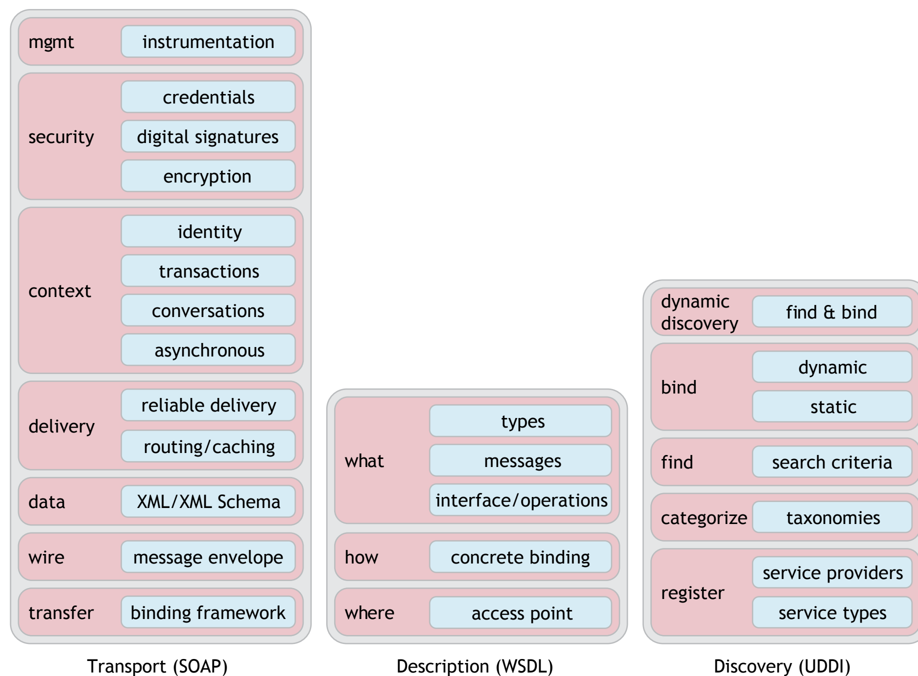

Figure 6. Summary of the WSA Functional Architecture

Copyright 2002, Systinet Corp.

WSA Invocation Mechanisms

Unlike other SOA systems, Internet middleware does not define a specific invocation mechanism. The core Internet middleware specifications simply define the communication protocols. The specifics of how applications interact with SOAP and WSDL have been left as an exercise to the application community. Any application language can parse and process XML, so at a minimum, applications can simply construct XML messages directly, package them in a SOAP envelope and exchange messages. Such manual processing isn't particularly conducive to developer productivity, and it doesn't exploit the fact that WSDL is machine-readable and can be compiled into application code. Hence the application community has produced a set of standard invocation mechanisms. Microsoft has defined a set of standard programming interfaces and class libraries for the Visual Studio .NET languages within the .NET framework, and the Microsoft SOAP Toolkit provides support for COM-based applications written in Visual Basic and Visual C++. The Java Community Process has just defined a set of standard programming interfaces for Java applications.

- JWSDL. The Java API for WSDL (JWSDL) provides an API to create, inspect, and manipulate a WSDL document. JWSDL is used when compiling WSDL files into client stubs and proxies, and it's used to process a WSDL file at runtime for dynamic binding and discovery.

- JAX-RPC. The Java API for XML based RPC (JAX-RPC), working with JWSDL, provides support for static binding, dynamic binding, and dynamic discovery. JAX-RPC also defines mappings between Java data types and XML types. The JAX-RPC client APIs automatically marshal and unmarshal SOAP requests on behalf of the client application.

    - A static *stub*, generated by compiling either the *how* or the *where* part of a WSDL document, provides a local object that represents the remote service. Clients simply invoke operations on the object, and the requests are automatically converted into SOAP requests and routed to the service. It looks and feels very much like RMI.

    - Clients can also invoke services using a JAX-RPC dynamic proxy. Clients create a *service* based on the abstract interface generated from the WSDL *what* part, and at runtime use the WSDL *where* part to generate a dynamic proxy that implements the binding.

    - Clients can also interpret a WSDL *where* part description at runtime and dynamically construct a JAX-RPC *call*.

- JAXM. The Java API for XML Messaging (JAXM) provides an API to construct SOAP messages without the benefit of a WSDL document. JAXM works with *profiles* that define a template for a SOAP message structure, automatically constructing the SOAP envelope, SOAP header, and SOAP body. The client application simply adds the XML payload to the message.

Copyright 2002, Systinet Corp.

- **SAAJ**. The SOAP with Attachments API (SAAJ) provides a low level SOAP API. Client applications can use SAAJ to manually construct and process SOAP messages.

- **JAXR**. The Java API for XML Registries (JAXR) is an API that can be used to access a variety of XML registries, including UDDI registries and ebXML registries. Given that it is a generic registry API, the JAXR data model is different from the UDDI data model. There are a number of other UDDI client APIs that more closely match the UDDI data model, including two open source projects: UDDI4J and the Systinet UDDI Client API.

Implementing Web Services Architecture

Since WSA is based on standard XML, you can implement Web services using the pervasive XML processing technologies that come with most platforms. You can build applications that query UDDI, parse WSDL documents, and construct SOAP requests—all using standard XML parsers. Or you can save yourself a lot of time and effort by using a set of Internet middleware products that implement these technologies.

Internet middleware products, sometimes referred to as a Web services platform, provide a ready-made foundation for building and deploying Web services. The advantage of using a Web services platform is that developers don't need to be concerned with constructing or interpreting SOAP messages. A developer simply writes the application code that implements the service, and the Internet middleware does the rest. A Web services platform generally consists of development tools, a runtime server, and a set of management services.

- Development tools are used to create Web services, to generate WSDL descriptions that describe those services, and to generate client proxies that can be used to send messages to the service. Development tools may also provide wizards to register or discover services in a UDDI registry.

- A runtime server processes SOAP messages and provides a runtime container for Web services. The runtime server often runs within a Web or application server. The runtime server listens for SOAP requests. For each request, the runtime server processes the SOAP message, translates the XML data into the service's native language, and invokes the service. When the service completes its work, the runtime server translates the return value into XML, packages it into a SOAP response message, and sends the message back to the calling application.

- Management tools provide mechanisms to deploy, undeploy, start, stop, configure, and administer your Web services. An administrative console is obviously useful, and other potential management services include a private UDDI registry, a WSDL repository, a single sign-on (SSO) service, and runtime monitoring facilities.

Copyright 2002, Systinet Corp.

Figure 7 shows a typical Web services platform runtime environment. Web services are deployed within a Web services server (the runtime server). A Web services server can run standalone, or it can execute within a Web server or application server. In a Java environment, a Web services server normally runs as a servlet. A Web services server consists of a SOAP message processor and a Web service container. The SOAP message processor processes an incoming SOAP message, converts it from XML into programming language data (e.g., Java), and routes the request to the application that implements the service. The application usually executes within the Web service container, which manages the lifecycle of the application. The Web services server acts as a lightweight application server.

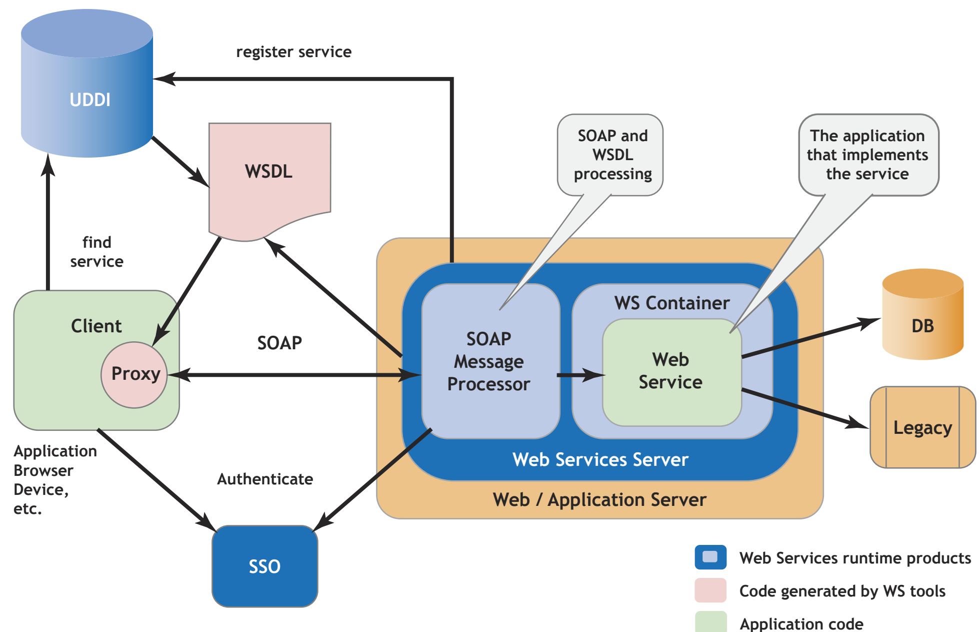

Figure 7. Internet middleware provides a ready-made Web services environment.

When you deploy a Web service, the Web services server usually generates a WSDL *where* part document that describes the Web service. (In some cases you may have to generate the WSDL document using the Web services development tools, or you may have to create one manually.) You'll want to register the Web service and its WSDL description in a UDDI registry to help users find the service. The client application uses UDDI to find the service and its description, and then uses the WSDL file to generate a client proxy. At runtime the client uses the client proxy to construct and send SOAP messages to the Web service. This illustration also shows the client and the service using a single sign-on service for authentication.

Platform Considerations

When choosing a Web services platform, you should ask yourself a number of questions.

- What language will you use to build your Web services?
- What language will you use to build your client applications?
- Do you have control over the client environment?
- What platform will you use to deploy your Web services?

Copyright 2002, Systinet Corp.

- Do you have a Web server or application server in place that you would like to use?
- What are your performance requirements?
- What are your scalability requirements?
- What level of security will you need?
- What other requirements might affect your choice?

Keep in mind these factors when evaluating a Web services platform:

- **Language support**. A Web services platform generally supports a specific language or set of languages. This factor is always your starting point. You want to select a Web services platform that supports the programming language you're using.

- **Platform support**. A Web services platform may support a limited set of operating systems or system configurations. You want to select a Web service platform that supports your preferred operating system and hardware.

- **Web/Application server support**. A Web services platform often runs within a Web or application server. Some platforms can run standalone, some platforms include a Web or application server, and others may be integrated with other application servers. Some platforms will only run in proprietary environments.

- **Transport support**. SOAP is designed to support multiple transports. Most Web services platforms support HTTP and HTTPS transfer protocols. Some platforms support other protocols, such as JMS and SMTP.

- **Interoperability**. One of the primary goals of Web services technology is to support interoperability across languages, platforms, and tools. But due to imprecision of the specifications, not all Web services platforms provide seamless interoperability with all other Web services platforms. If you intend to expose your Web services to clients outside of your realm of control, make sure that you select a Web service platform that ensures easy interoperability.

- **Performance**. Web services communicate by exchanging XML. XML is verbose, and it must be processed (converted to native languages) at both ends of the wire. One of the most important factors affecting Web service performance is the speed at which the SOAP message processor can translate XML. As messages get larger or more complex, the translation time goes up. Run some performance tests to ensure that the platform you choose meets your minimum performance requirements.

- **Scalability**. If you intend to publish a Web service to a large audience, you will need a Web services platform that can deliver consistent, predictable performance as the client volume rises. Don't forget to test for scalability.

Copyright 2002, Systinet Corp.

- **Manageability**. If you intend to use your Web services to support your business operations, they need to be up and running when you need them to be up and running. You will need the appropriate management facilities to monitor and manage your Web services environment.

- **Security**. If you intend to make your internal business processes available through a Web services interface, you'll almost certainly need to set up some form of security on those systems to protect yourself from unauthorized access or malicious use.

Systinet WASP

Systinet Web Applications and Services Platform (WASP) is the industry's most advanced Web services infrastructure solution. WASP goes far beyond supplying just a basic Web services platform. WASP provides the fastest, most scalable, most secure, most advanced Web services solution in the industry. WASP provides everything you need to create, deploy, and manage Web services.

The WASP product suite consists of two runtime servers, a set of development tools, and a private UDDI registry service.

WASP Server is a portable, high-performance Web services runtime environment. WASP Server can run standalone or it can be configured to work with a wide variety of Web servers, application servers, and database servers. WASP Server includes comprehensive command line development tools that provide developers with everything they need to build and deploy Web services. The environment provides excellent interoperability with most SOAP implementations, including Microsoft .NET and IBM Web Services Tool Kit. The environment includes an end-to-end security framework and comprehensive management services. The runtime is equipped with a number of built-in extensibility hooks that permit easy enhancement and customization of the environment. Systinet provides two versions of this product:

- WASP Server for Java
- WASP Server for C++

WASP Developer seamlessly extends the industry's most popular Java IDEs to support Web services. WASP Developer provides a point-and-click code generation experience that can turn any existing Java application into a Web service. WASP Developer fully automates the generation of all WSDL descriptions and SOAP interface code. Integrated deployment, debugging, security, and monitoring tools give the developer complete control from within the IDE. WASP Developer also includes UDDI wizards to help you locate or publish Web services. Systinet provides three versions of this product:

- WASP Developer for Sun ONE Studio
- WASP Developer for Borland JBuilder
- WASP Developer for Eclipse

Copyright 2002, Systinet Corp.

WASP UDDI is a secure UDDI registry service designed for private use. This registry service conforms to the UDDI V2 specification and supports access using most UDDI client libraries, including UDDI4J, JAXR, and Systinet's own open source UDDI client library. WASP UDDI supports enhanced security and query facilities that address the requirements of a private enterprise or community. WASP UDDI also provides support for custom taxonomies that businesses can use to categorize their services.

Systinet is constantly working to enhance and extend the Web services environment. In addition to the official product family, Systinet also provides previews of two advanced technologies:

- **WASP Secure Identity** is a preview of a SAML-based single sign-on service. Users sign on and receive a SAML authentication assertion in response. This SAML assertion can be passed in the SOAP header in all subsequent Web service invocations. The WASP client and server runtime services automatically manage and interpret the SAML SOAP headers. WASP Secure Identity also works with other Web services platforms that support SOAP header processing.

- **WASP TX** is a preview of a Web services transaction service. WASP TX implements the OASIS Business Transaction Protocol (BTP[21]) standard, which is designed to support loosely coupled transactions. WASP TX can be used to coordinate transactions across any number of Web services, implemented using any Web services platform, and it works in heterogeneous environments. WASP TX relies on SOAP headers to convey transaction identifiers.

The key features of WASP are:

- **Performance and scalability**. WASP is 5-10 times faster that most other SOAP-based solutions, and it is the only solution that offers flat, consistent performance regardless of the number of concurrent users.

- **Cross-platform support**. WASP has been designed to fit seamlessly into nearly any existing configuration. WASP supports both Java and C++ environments. WASP can be deployed standalone, or it can be deployed in a wide assortment of Web and application servers, including BEA WebLogic, IBM WebSphere, Sun ONE, Oracle, Borland, Orion, J2EE Reference Implementation, JBoss, Tomcat, Jetty, Apache Web Server, Microsoft IIS, and AOL Netscape Enterprise Server.

- **Standards support**. WASP supports all the latest Web services standards and technologies, including SOAP 1.1, SOAP with Attachments, W3C SOAP 1.2, WSDL 1.1, UDDI 1.0, UDDI 2.0, W3C XML Schema, W3C XML Signature, and OASIS SAML.

[21] The OASIS BTP specification: http://www.oasis-open.org/committees/business-transactions/#documents

Copyright 2002, Systinet Corp.

- **Extensibility**. Systinet WASP provides an extensive set of plug-in modules and interceptors that can be used to implement automatic, transparent support for additional middleware functionality at any stage during Web service processing.

- **Interoperability and Integration**. WASP offers excellent integration facilities, and enables seamless interoperability with .NET and codeless integration with J2EE.

- **Management**. WASP provides comprehensive management facilities to monitor and administer your distributed Web services environment.

- **Security**. WASP is the only Web services solution to provide an automatic, transparent, end-to-end security framework that supports authentication, authorization, data integrity, data confidentiality, and non-repudiation. The WASP security framework supports multiple authentication mechanisms and security providers.

WASP is free for evaluation, development, and testing. We encourage you to try it. Use it to learn about Web services. Use it to implement your first proof of concept. Then test it against other products. No other Web services platform can compare.

About Systinet

Systinet provides Web services infrastructure software. Our products make it easy for enterprises to build, deploy, secure and manage Web services.

The Systinet WASP suite of products is based on industry-standards such as XML, SOAP, WSDL and UDDI. WASP products are available for Java and C++, interoperate seamlessly with other Web services implementations such as .NET, and are portable across a wide variety of platforms and servers.

Systinet is a privately-held company with headquarters in Cambridge, Massachusetts.

Five Cambridge Center, 8th Floor
Cambridge, MA 02142
Phone: 617.868.2224
E-mail: sales@systinet.com

Copyright 2002, Systinet Corp.