Web Services Conceptual Architecture (WSCA 1.0)

May 2001

By Heather Kreger
IBM Software Group

Notice

The authors have utilized their professional expertise in preparing this report. However, neither International Business Machines Corporation nor the authors make any representation or warranties with respect to the contents of this report. Rather, this report is provided on an AS IS basis, without warranty, express or implied, INCLUDING A FULL DISCLAIMER OF THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Contents

Figures

Preface

Abstract

This paper describes the architecture for Web Services from the point of view of components, interactions and application development patterns. This architecture is the blueprint for an IBM instantiation of the Web Services approach. It is a framework for the building and deployment of Web Services applications.

The architecture presented in this paper includes high-level descriptions of the components and functions required for Web Services, and requirements on the tools and middleware to implement these components and functions. Some functionality exists today in products such as the IBM XML and Web Services Development Environment, the IBM Web Services Toolkit and IBM WebSphere® Application Server. These and other products will implement additional functions in the future. However, the presence of a component, function or requirement in this paper does not guarantee that it will be implemented in future IBM products.

Target Audience

- Early adopters and implementers of Web Services.
- External technical reviewers evaluating the IBM Web Services approach. Reviewers should read the *Introducing Dynamic e-business: Concepts and Value* paper that explains the value of Web Services.

Comments

Please send any feedback, technical or editorial, to Web Services at wbservcs@us.ibm.com or webservices/Raleigh/IBM@IBMUS.

Web Services Overview

This section briefly reviews Web Services as an application integration technology, defines the term *web service* and describes the Web Services model.

Web Services: The Next Horizon for e-business

What the Web did for program-to-user interactions, Web Services are poised to do for program-to-program interactions. Web Services allow companies to reduce the cost of doing e-business, to deploy solutions faster and to open up new opportunities. The key to reaching this new horizon is a common program-to-program communications model, built on existing and emerging standards such as HTTP, Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI).

Web Services allow applications to be integrated more rapidly, easily and less expensively than ever before. Integration occurs at a higher level in the protocol stack, based on messages centered more on service semantics and less on network protocol semantics, thus enabling loose integration of business functions. These characteristics are ideal for connecting business functions across the Web—both between enterprises and within enterprises. They provide a unifying programming model so that application integration inside and outside the enterprise can be done with a common approach, leveraging a common infrastructure. The integration and application of Web Services can be done in an incremental manner, using existing languages and platforms and by adopting existing legacy applications. Moreover, Web Services compliment Java™ 2 Platform, Enterprise Edition (J2EE), Common Object Request Broker Architecture (CORBA) and other standards for integration with more tightly coupled distributed and nondistributed applications. Web Services are a technology for *deploying and providing access* to business functions over the Web; J2EE, CORBA and other standards are technologies for *implementing* Web Services.

Although early use of Web Services is peer-wise and *ad hoc*, it still addresses the complete problem of program-to-program communications including describing, publishing and finding interfaces. And, as the use of Web Services grows and the industry matures, more dynamic models of application integration will develop. Eventually systems integration through Web Services will happen dynamically at runtime. Just-in-time integration will herald a new era of business-to-business integration over the Internet.

Definition of Web Services

A *Web service* is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion, called its *service description*. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web Services fulfill a specific task or a set of tasks. They can be used alone or with other Web Services to carry out a complex aggregation or a business transaction.

The Web Services Model

The Web Services architecture is based upon the interactions between three roles: service provider, service registry and service requestor. The interactions involve the publish, find and bind operations. Together, these roles and operations act upon the Web Services artifacts: the Web service software module and its description. In a typical scenario, a service provider hosts a network-accessible software module (an implementation of a Web service). The service provider defines a service description for the Web service and publishes it to a service requestor or service registry. The service requestor uses a find operation to retrieve the service description locally or from the service registry and uses the service description to bind with the service provider and invoke or interact with the Web service implementation. Service provider and service requestor roles are logical constructs and a service can exhibit characteristics of both. Figure 1 illustrates these operations, the components providing them and their interactions.

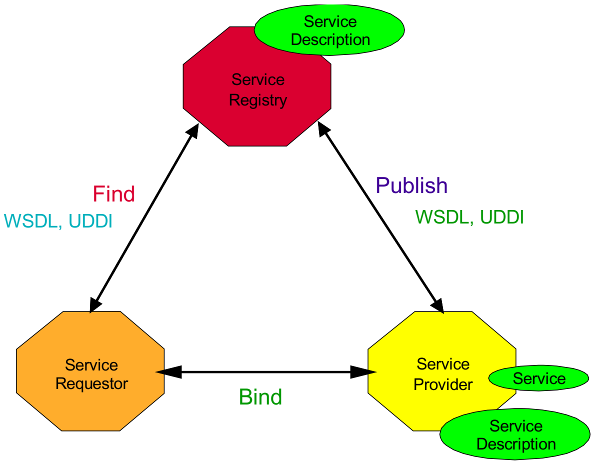

Figure 1. Web Services roles, operations and artifacts

Roles in a Web Services Architecture

- **Service provider.** From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that hosts access to the service.
- **Service requestor.** From a business perspective, this is the business that requires certain functions to be satisfied. From an architectural perspective, this is the application that is looking for and invoking or initiating an interaction with a service. The service requestor role can be played by a browser driven by a person or a program without a user interface, for example another Web service.

- **Service registry.** This is a searchable registry of service descriptions where service providers publish their service descriptions. Service requestors find services and obtain binding information (in the service descriptions) for services during development for static binding or during execution for dynamic binding. For statically bound service requestors, the service registry is an optional role in the architecture, because a service provider can send the description directly to service requestors. Likewise, service requestors can obtain a service description from other sources besides a service registry, such as a local file, FTP site, Web site, Advertisement and Discovery of Services (ADS) or Discovery of Web Services (DISCO).

Operations in a Web Service Architecture

For an application to take advantage of Web Services, three behaviors must take place: publication of service descriptions, lookup or finding of service descriptions, and binding or invoking of services based on the service description. These behaviors can occur singly or iteratively. In detail, these operations are:

- **Publish.** To be accessible, a service description needs to be published so that the service requestor can find it. Where it is published can vary depending upon the requirements of the application (see “Service Publication” for more details).
- **Find.** In the find operation, the service requestor retrieves a service description directly or queries the service registry for the type of service required (see “Service Discovery” for more details). The find operation can be involved in two different lifecycle phases for the service requestor: at design time to retrieve the service’s interface description for program development, and at runtime to retrieve the service’s binding and location description for invocation.
- **Bind.** Eventually, a service needs to be invoked. In the bind operation the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact and invoke the service.

Artifacts of a Web Service

- **Service.** Where a Web service is an interface described by a service description, its implementation is the service. A service is a software module deployed on network-accessible platforms provided by the service provider. It exists to be invoked by or to interact with a service requestor. It can also function as a requestor, using other Web Services in its implementation.
- **Service Description.** The service description contains the details of the interface and implementation of the service. This includes its data types, operations, binding information and network location. It could also include categorization and other meta-data to facilitate discovery and utilization by service requestors. The service description might be published to a service requestor or to a service registry.

The Web Services architecture explains how to instantiate the elements and implement the operations in an interoperable manner.

Web Services Development Lifecycle

The Web Services development lifecycle includes the design, deployment, and runtime requirements for each of the roles: service registry, service provider and service requestor. Each role has specific requirements for each element of the development lifecycle. The development and deployment of a service registry is outside the scope of this paper.

The development lifecycle can have four phases:

1. Build

   The build phase of the lifecycle includes development and testing of the Web service implementation, the definition of the service interface description and the definition of the service implementation description. Web service implementations can be provided by creating new Web Services, transforming existing applications into Web Services, and composing new Web Services from other Web Services and applications.

2. Deploy

   The deploy phase includes the publication of the service interface and service implementation definition to a service requestor or service registry and deployment of the executables for the Web service into an execution environment (typically, a Web application server).

3. Run

   During the run phase, the Web service is available for invocation. At this point, the Web service is fully deployed, operational and network-accessible from the service provider. Now the service requestor can perform the find and bind operations.

4. Manage

   The manage phase covers ongoing management and administration of the Web service application. Security, availability, performance, quality of service and business processes must all be addressed.

The details of each of these lifecycle phases are discussed in the *Web Services Development Concepts* paper.

Architecture Overview

We can examine the IBM Web Services architecture in several layers. First, we will look at a conceptual stack for Web Services and the stack details. Then we will discuss the criteria for choosing the network protocol. We will also review basic XML-based messaging distributed computing. We extend basic XML messaging with service description, which is explained in terms of a service description stack. Following this, we discuss the role of service description in the Web Services architecture, illustrating the range of service publication techniques supporting static and dynamic Web Services applications. Related to service publication, we discuss the role of service discovery. Finally, we describe extensions of the basic Web Services architecture required to make Web Services viable for e-business.

The Web Services Stack

To perform the three operations of publish, find and bind in an interoperable manner, there must be a Web Services stack that embraces standards at each level. Figure 2 shows a conceptual Web Services stack. The upper layers build upon the capabilities provided by the lower layers. The vertical towers represent requirements that must be addressed at every level of the stack. The text on the left represents standard technologies that apply at that layer of the stack.

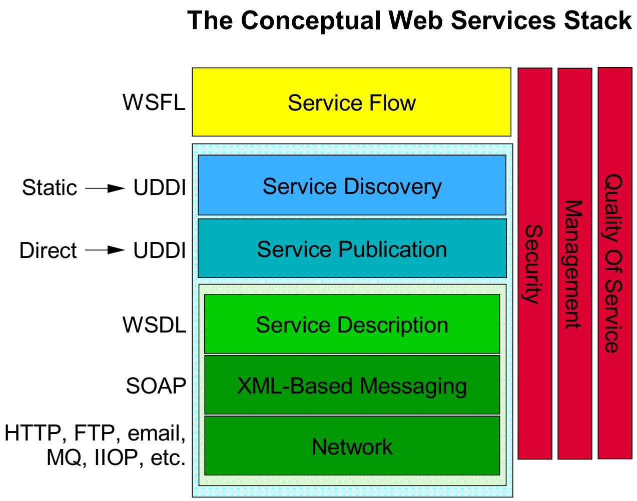

Figure 2. Web Services conceptual stack

The foundation of the Web Services stack is the network. Web Services must be network-accessible to be invoked by a service requestor. Web Services that are publicly available on the Internet use commonly deployed network protocols. Because of its ubiquity, HTTP is the *de facto* standard network protocol for Internet-available Web Services. Other Internet protocols can be supported, including SMTP and FTP. Intranet domains can use reliable messaging and call infrastructures like MQSeries®, CORBA, and so on. The section “The Network” describes this layer in more detail.

The next layer, XML-based messaging, represents the use of XML as the basis for the messaging protocol. SOAP is the chosen XML messaging protocol for many reasons:

- It is the standardized enveloping mechanism for communicating document-centric messages and remote procedure calls using XML.
- It is simple; it is basically an HTTP POST with an XML envelope as payload.
- It is preferred over simple HTTP POST of XML because it defines a standard mechanism to incorporate orthogonal extensions to the message using SOAP headers and a standard encoding of *operation* or *function*.
- SOAP messages support the publish, find and bind operations in the Web Services architecture. The section “XML Messaging-Based Distributed Computing” describes this layer in more detail.

The service description layer is actually a stack of description documents. First, WSDL is the *de facto* standard for XML-based service description. This is the minimum standard service description necessary to support interoperable Web Services. WSDL defines the interface and mechanics of service interaction. Additional description is necessary to specify the business context, qualities of service and service-to-service relationships. The WSDL document can be complemented by other service description documents to describe these higher level aspects of the Web service. For example, business context is described using UDDI data structures in addition to the WSDL document. Service composition and flow are described in a Web Services Flow Language (WSFL) document. The section “Service Description: From XML Messaging to Web Services” describes this layer in more detail.

Because a Web service is defined as being network-accessible via SOAP and represented by a service description, the first three layers of this stack are required to provide or use any Web service. The simplest stack would consist of HTTP for the network layer, the SOAP protocol for the XML messaging layer and WSDL for the service description layer. This is the interoperable base stack that all inter-enterprise, or public, Web Services should support. Web Services, especially intra-enterprise, or private, Web Services, can support other network protocols and distributed computing technologies. Figure 3 depicts the interoperable base stack.

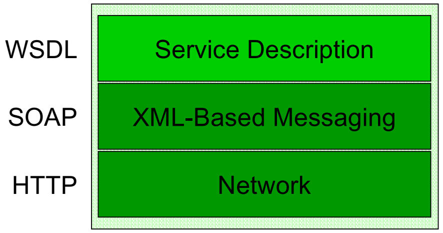

Figure 3. Interoperable base Web Services stack

The stack depicted in Figure 3 provides for interoperability and enables Web Services to leverage the existing Internet infrastructure. This creates a *low cost of entry* to a ubiquitous

environment. Flexibility is not compromised by the interoperability requirement, because additional support can be provided for alternative and value-add technologies. For example, SOAP over HTTP must be supported, but SOAP over MQ can be supported as well.

While the bottom three layers of the stack identify technologies for compliance and interoperability, the next two layers—service publication and service discovery—can be implemented with a range of solutions.

Any action that makes a WSDL document available to a service requestor, at any stage of the service requestor’s lifecycle, qualifies as *service publication*. The simplest, most static example at this layer is the service provider sending a WSDL document directly to a service requestor. This is called *direct publication*. E-mail is one vehicle for direct publication. Direct publication is useful for statically bound applications. Alternatively, the service provider can publish the WSDL document describing the service to a host local WSDL registry, private UDDI registry or the UDDI operator node. The variety of service publication mechanisms is discussed in more detail in the section “Service Publication.”

Because a Web service cannot be discovered if it has not been published, service discovery depends upon service publication. The variety of discovery mechanisms at this layer parallels the set of publication mechanisms. Any mechanism that allows the service requestor to gain access to the service description and make it available to the application at runtime qualifies as service discovery. The simplest, most static example of discovery is static discovery wherein the service requestor retrieves a WSDL document from a local file. This is usually the WSDL document obtained through a direct publish or the results of a previous find operation. Alternatively, the service can be discovered at design time or runtime using a local WSDL registry, a private UDDI registry or the UDDI operator node. The variety of service discovery mechanisms is discussed in more detail in the section “Service Discovery.”

Because a Web service’s implementation is a software module, it is natural to produce Web Services by *composing* Web Services. A composition of Web Services could play one of several roles. Intra-enterprise Web Services might collaborate to present a single Web service interface to the public, or the Web Services from different enterprises might collaborate to perform machine-to-machine, business-to-business transactions. Alternatively, a workflow manager might call each Web service as it participates in a business process. The topmost layer, service flow, describes how service-to-service communications, collaborations, and flows are performed. WSFL is used to describe these interactions. The topic of Web Services flows is covered in its own section “Business Processes, Workflows and Web Services.”

For a Web Services application to meet the stringent demands of today’s e-businesses, enterprise-class infrastructure must be supplied, including security, management and quality of service. These vertical towers must be addressed at each layer of the stack. The solutions at each layer can be independent of each other. More of these vertical towers will emerge as the Web Services paradigm is adopted and evolved. We discuss these vertical towers in more detail in the section “Web Services for Real e-business.”

The bottom layers of this stack, representing the base Web Services stack, are relatively mature and more standardized than the layers higher in the stack. The maturation and adoption of Web Services will drive the development and standardization of the higher levels of the stack and the vertical towers.

The Network

At the base of the Web Services stack is the network. This layer can represent any number of network protocols: HTTP, FTP, SMTP, Message Queuing (MQ), Remote Method Invocation (RMI) over Internet Inter ORB Protocol (IIOP), e-mail, and so on. The network protocol used in any given situation depends on application requirements.

For Web Services accessible from the Internet, the network technology choices will favor ubiquitously deployed protocols such as HTTP. For Web Services being provided and consumed within an Intranet, there is the opportunity to agree upon the use of alternative network technologies. The network technology can be chosen based on other requirements, including security, availability, performance and reliability. This allows Web Services to capitalize on existing higher-function networking infrastructures and message-oriented middleware, such as MQSeries. Within an enterprise with multiple types of network infrastructures, HTTP can be used to bridge between them.

One of the benefits of Web Services is that it provides a unified programming model for the development and usage of private Intranet and public Internet services. As a result, the choice of network technology will be transparent to the developer of the service.

XML Messaging-Based Distributed Computing

The most fundamental underpinnings of the IBM Web Services architecture is XML messaging. The current industry standard for XML messaging is SOAP. IBM, Microsoft® and others submitted SOAP to the W3C as the basis of the XML Protocol Working Group. The XML protocol will replace SOAP as the industry-standard XML messaging protocol. When the W3C has released a draft standard for the XML protocol, the IBM Web Services architecture will migrate from SOAP to the XML protocol.

SOAP is a simple and lightweight XML-based mechanism for exchanging structured data between network applications. SOAP consists of three parts: an envelope that defines a framework for describing what is in a message, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls (RPCs) and responses. SOAP can be used in combination with or re-enveloped by a variety of network protocols such as HTTP, SMTP, FTP, RMI over IIOP or MQ.

While it is important to understand this foundation, most Web service developers will not have to deal with this infrastructure directly. Most Web Services will use optimized programming language-specific bindings generated from WSDL. This optimization can be especially valuable when a service provider and requestor are both executing in similar environments.

Figure 4 shows how XML messaging (that is, SOAP) and network protocols forms the basis of the IBM Web Services architecture.

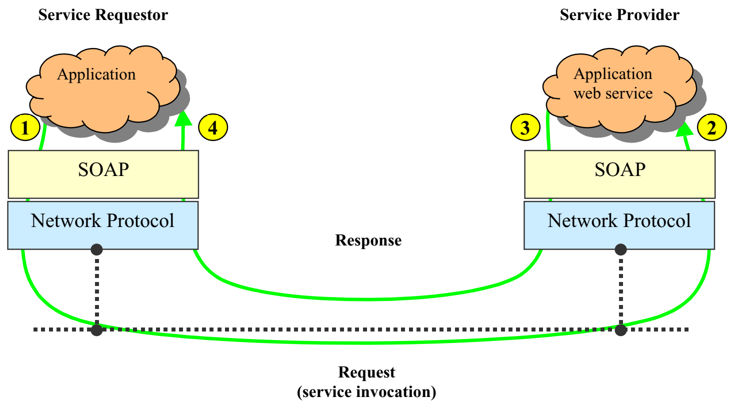

Figure 4. XML messaging using SOAP

The basic requirements for a network node to play the role of requestor or provider in XML messaging-based distributed computing are the ability to build, parse a SOAP message, or both, and the ability to communicate over a network (receive, send messages, or both).

Typically, a SOAP server running in a Web application server performs these functions. Alternatively, a programming language-specific runtime library can be used that encapsulates these functions within an API. Application integration with SOAP can be achieved by using four basic steps:

1. In Figure 4, at (1) a service requestor's application creates a SOAP message. This SOAP message is the request that invokes the Web service operation provided by the service provider. The XML document in the body of the message can be a SOAP RPC request or a document-centric message as indicated in the service description. The service requestor presents this message together with the network address of the service provider to the SOAP infrastructure (for example, a SOAP client runtime). The SOAP client runtime interacts with an underlying network protocol (for example HTTP) to send the SOAP message out over the network.

2. At (2) the network infrastructure delivers the message to the service provider's SOAP runtime (for example a SOAP server). The SOAP server routes the request message to the service provider's Web service. The SOAP runtime is responsible for converting the XML message into programming language-specific objects if required by the application. This conversion is governed by the encoding schemes found within the message.

3. The Web service is responsible for processing the request message and formulating a response. The response is also a SOAP message. At (3) the response SOAP message is

presented to the SOAP runtime with the service requestor as the destination. In the case of synchronous request/response over HTTP, the underlying request/response nature of the networking protocol is used to implement the request/response nature of the messaging. The SOAP runtime sends the SOAP message response to the service requestor over the network.

4. At (4) the response message is received by the networking infrastructure on the service requestor's node. The message is routed through the SOAP infrastructure; potentially converting the XML message into objects in a target programming language. The response message is then presented to the application.

This example uses the request/response transmission primitive that is quite common in most distributed computing environments. The request/response exchange can be synchronous or asynchronous. Other transmission primitives such as one-way messaging (no response), notification (push-style response), publish/subscribe are possible using SOAP.

Now, how does the service requestor know what format the request message should use? This question is addressed this in the next section.

Service Description: From XML Messaging to Web Services

It is through the service description that the service provider communicates all the specifications for invoking the Web service to the service requestor. The service description is key to making the Web Services architecture loosely coupled and reducing the amount of *required* shared understanding and custom programming and integration between the service provider and the service requestor. For example, neither the requestor nor the provider must be aware of the other's underlying platform, programming language, or distributed object model (if any). The service description combined with the underlying SOAP infrastructure sufficiently encapsulates this detail away from the service requestor's application and the service provider's Web service.

The Basic Service Description

The IBM Web Services architecture uses WSDL for base-level service description. WSDL has been submitted to the W3C for standardization. WSDL is an XML document for describing Web Services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented (RPC) messages. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints or services. WSDL is extensible to allow description of endpoints and their messages, regardless of what message formats or network protocols are used to communicate. However, the only currently described bindings are for SOAP 1.1, HTTP POST, and Multipurpose Internet Mail Extensions (MIME).

The use of WSDL in the IBM Web Services architecture conventionally divides the basic service description into two parts: the service interface and the service implementation. This enables each part to be defined separately and independently, and reused by other parts.

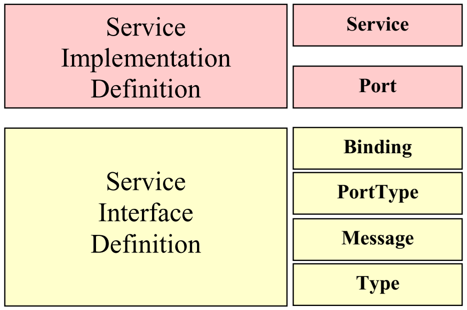
Service Implementation Definition
Service
Port
Service Interface Definition
Binding
PortType
Message
Type

Figure 5. Basic service description

A *service interface definition* is an abstract or reusable service definition that can be instantiated and referenced by multiple service implementation definitions. Think of a service interface definition as an Interface Definition Language (IDL), Java interface or Web service type. This allows common industry-standard service types to be defined and implemented by multiple service implementers. This is analogous to defining an abstract interface in a programming language and having multiple concrete implementations. Service interfaces can be defined by industry standards organizations such as RosettaNet, or HL7 for the health industry.

The service interface contains WSDL elements that comprise the *reusable* portion of the service description: WSDL:*binding*, WSDL:*portType*, WSDL:*message* and WSDL:t*ype* elements as depicted in Figure 5. In the WSDL:*portType* element, the operations of the Web service are defined. The operations define what XML messages can appear in the input and output data flows. Think of an operation as a method signature in a programming language. The WSDL:*message* element specifies which XML data types constitute various parts of a message. WSDL:*message* element is used to define the input and output parameters of an operation. The use of complex data types within the message is described in the WSDL:*types* element. The WSDL:*binding* element describes the protocol, data format, security and other attributes for a particular service interface (WSDL:*portType*).

The *service implementation definition* is a WSDL document that describes how a particular service interface is implemented by a given service provider. A Web service is modeled as a WSDL:*service* element. A service element contains a collection (usually one) of WSDL:*port* elements. A port associates an endpoint (for example, a network address location or URL) with a WSDL:*binding* element from a service interface definition.

To illustrate the allocation of responsibility, the Open Applications Group (OAG) might define a service interface definition for the Open Applications Group Integration Specification (OAGIS) purchase-order standard. This service interface definition would define WSDL:*type*, WSDL:*message*, WSDL:*portType* and WSDL:*binding*. This specification would be published at some well-known site (for example http://www.openapplications.org/).

A service provider can choose to develop a Web service that implements the OAGIS purchase order service interface. The service provider would develop a service implementation definition document that describes the WSDL service, port and address location elements that describe the network address of the provider's Web service and other implementation-specific details.

The service interface definition together with the service implementation definition makes up a complete WSDL definition of the service. This pair contains sufficient information to describe to the service requestor *how* to invoke and interact with the Web service. The service requestor can require other information about the service provider's endpoint. This information is provided by the complete Web service description of the service.

The Complete Web Service Description

The complete Web service description builds on the basic WSDL definition of the service. The complete Web service description addresses questions such as: What business is hosting this service and what kind of business is it? What products are associated with this service? With what categories in various company and product taxonomies is this business or its Web service associated? Are there other aspects of the service (such as Quality of Service) that can influence whether a requestor would choose to invoke the service? What keywords can be provided so that it is easier to find this service?

A complete Web service description is depicted in Figure 6.

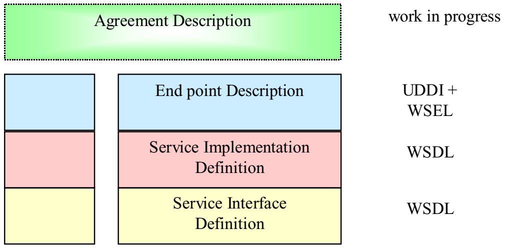

Figure 6. Complete Web Services description stack

UDDI provides a mechanism for holding descriptions of Web Services. Although UDDI is often thought of as a directory mechanism, it also defines a data structure standard for representing service description information in XML. There are four fundamental data structures in a UDDI entry, as depicted in Figure 7.

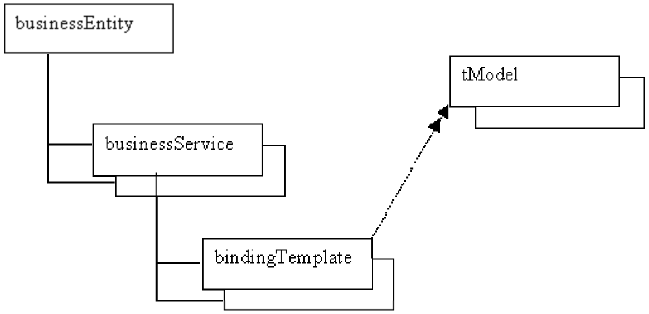

Figure 7. Basic UDDI data structures

A UDDI entry starts with a *businessEntity*. A *businessEntity* element models information about a business, including basic business information (for example, What is the business name and contact information?), categorization information (for example, What kind of business is this?), and identifier information (that is, What is the Dunn and Bradstreet number?). A *businessEntity* contains a collection of *businessService* elements, one for each Web service the business wishes to publish. Each *businessService* element contains technical and descriptive information about a *businessEntity* element's Web service. A *businessService* contains a collection of *bindingTemplate* elements. A *bindingTemplate* describes the access information (for example, the endpoint address) and describes how the *businessService* uses various technical specifications. A technical specification is modeled as a *tModel*. A *tModel* can model many different concepts such as: a type of service, a platform technology such as HTTPS or a taxonomy. The collection of *bindingTemplate* elements associated with a *businessService* represents a fingerprint of the technologies used by the *businessService*.

In the IBM Web Services architecture, the complete Web service description includes a layer for an *endpoint description* that uses the UDDI entry to add business and implementation context to the service description. The endpoint description follows the convention proposed by IBM for using UDDI in combination with WSDL. The endpoint description uses UDDI to provide standard representation of business information and taxonomies. The UDDI-WSDL convention outlines how the WSDL description of the service interface definition and the service implementation definition can be derived from the UDDI entries associated with the Web service. This convention is critical to using UDDI as a service registry for WSDL-based services in the IBM Web Services architecture.

The endpoint description adds additional semantics to the service description that apply to a particular implementation of the service. Security attributes can define the policy governing access to the Web service. Quality of Service attributes will specify performance-oriented capabilities, for example, the service's ability to respond within a certain period of time or the level of reliable messaging supported. Cost of Service attributes will describe the service's resource requirements. What conversation semantics are supported could also be defined.

The final layer in the service description stack is the *agreement description*. An agreement description reflects a simple choreography of Web service invocations between two business partners to complete a multistep business interaction. For example, an agreement definition defines roles such as buyer and seller within a purchasing protocol. The agreement definition outlines the requirements that each role must fulfill. For example, the seller must have Web Services that receive request for quote (RFQ) messages, purchase order (PO) messages and payment messages. The buyer role must have Web Services that receive quotes (RFQ response messages), invoice messages and account summary messages. This simple choreography of Web Services into business roles is critical for establishing multistep, service-oriented interactions between business partners. A given service requestor or service provider might be able to play the buyer or seller role in a number of different business agreement standards. By explicitly modeling business agreements and each node's ability to play roles in the business agreement, the requestor can choose which business agreement to engage in with various provider business partners.

This area is rich with innovation. Currently there is no single standard for business protocol definition. The ebXML Collaboration-Protocol Profile and Agreement Specification describes these concepts, but is not based on the Web Services techniques described as part of this architecture. The Web Services flow description and Web Services endpoint description layers are being developed to provide this level of service description.

Publication and Discovery of Service Descriptions

Service Publication

The publication of Web Services includes the production of the service descriptions and the subsequent publishing. Publishing can use a variety of mechanisms.

Producing Service Descriptions

The service description can be generated, hand-coded, or pieced together based on existing service interface definitions. Developers can hand-code the entire service description, including the UDDI entry. Tools exist to generate parts of the WSDL and potentially parts of the UDDI entry from meta-data artifacts from the programming model and the deployment of the Web service executable. Parts of the service description can already exist (for example the Web service can be based on an industry-standard service interface definition) such that very little needs to be further generated.

Publishing Service Descriptions

A service description can be published using a variety of mechanisms. These various mechanisms provide different capabilities depending on how dynamic the application using the service is intended to be. The service description can be published to multiple service registries using several different mechanisms.

The simplest case is a *direct publish*. A direct publish means the service provider sends the service description directly to the service requestor. This can be accomplished using an e-mail attachment, an FTP site or even a CD-ROM distribution. Direct publish can occur after two business partners have agreed on terms of doing e-business over the Web, or after fees have been paid by the service requestor for access to the service. In this case, the service requestor can maintain a local copy of the service description.

Slightly more dynamic publication uses DISCO or ADS. Both DISCO and ADS define a simple HTTP GET mechanism to retrieve Web Services descriptions from a given URL. An enhanced service description repository would provide a local cache of service descriptions, but with additional search capabilities.

For service description repositories that span hosts within an enterprise, a service provider would publish to a private UDDI node. There are several types of private UDDI nodes that can be used depending on the scope of the domain of Web Services published to it.

- Internal Enterprise Application UDDI node: Web Services for use within a company for internal enterprise applications integration should be published to a UDDI node of this kind. The scope of this UDDI node can be single application, departmental or corporate. These UDDI nodes sit behind the firewall and allow the service publishers more control over their service registry and its accessibility, availability and publication requirements.

- Portal UDDI node: Web Services published by a company for external partners to find and use can use a portal UDDI node. A portal UDDI node runs outside the service provider's firewall or between firewalls. This kind of private UDDI node contains only those service descriptions that a company wishes to provide to service requestors from external partners. This allows companies to retain control of their service descriptions, access to the UDDI node and quality of service for the UDDI nodes. Moreover, by using the role-based visibility inherent in portals, the enterprise limits visibility of service descriptions to the partners authorized to see their existence.

- Partner Catalog UDDI node: Web Services to be used by a particular company can be published to a partner catalog UDDI node. A partner catalog UDDI node sits behind the firewall. This kind of private UDDI node contains only approved, tested and valid Web service descriptions from legitimate business partners. The business context and meta-data for these Web Services can be targeted to the specific requestor.

- E-Marketplace UDDI node: For Web Services that the service provider intends to compete for requestors' business with other Web Services, the service description should be published to an e-marketplace UDDI node or the UDDI operator node. E-marketplace UDDI nodes are hosted by an industry standards organization or consortium and contain service descriptions from businesses in a particular industry. These services can be required to support specific standards, searchable meta-data, interfaces, or data types. E-marketplace UDDI nodes will generally provide some filtering of illegitimate entries and some guaranteed qualities of service.

UDDI Operator node: Web Services might also wish to publish to the UDDI Operator node in hopes of being discovered by new potential business partners or service users. The UDDI operator node is supported, replicated and hosted by IBM, Microsoft and Ariba. When publishing the UDDI operator node, complete business context and well-thought-out taxonomies are essential if the service is to be found by potential service requestors.

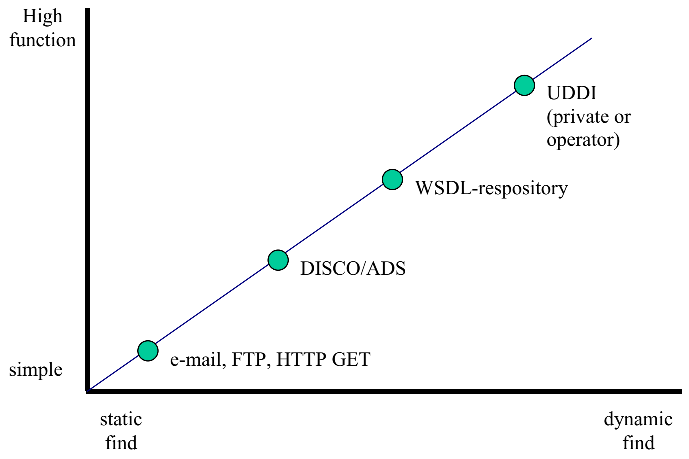

Figure 8. Service discovery continuum

Figure 8 shows the continuum from the most static, easiest technologies for publish and discovery to the most dynamic, more complex technologies. Users or implementers of Web Services might not follow this progression in strict sequence.

Service Discovery

The discovery of Web Services includes the acquiring of the service descriptions and the consuming of the descriptions. Acquiring can use a variety of mechanisms

Acquiring Service Descriptions

Like publishing Web service descriptions, acquiring Web service descriptions will vary depending on how the service description is published and how dynamic the Web service application is meant to be. Service requestors will find Web Services during two different phases of an application lifecycle—design time and runtime. At design time, service requestors search for Web service descriptions by the type of interface they support. At runtime, service requestors search for a Web service based on how they communicate or qualities of service advertised.

With the direct publish approach, the service requestor caches the service description at design time for use at runtime. The service description can be statically represented in the program logic, stored in a file or in a simple, local service description repository.

Service requestors can retrieve a service description at design time or runtime from a service description repository, a simple service registry or a UDDI node. The look-up mechanism needs to support a query mechanism that provides find by type of interface (based on a WSDL template), the binding information (that is, protocols), properties (such as QOS parameters), the types of intermediaries required, the taxonomy of the service, business information, and so on.

The various types of UDDI nodes have implications on the number of runtime binding Web Services to choose from, the policy for choosing one among many, or the amount of prescreening that must be done by the requestor before invoking the service.

Internal enterprise application UDDI nodes and partner catalog UDDI nodes will require no prescreening to establish trust of the service. Service selection can be based on binding support, historical performance, quality of service classification, proximity, or load balancing.

E-marketplace UDDI nodes will have more runtime services to choose from. Some prescreening must be done to verify that the Web service provider is a worthy partner. A service can be chosen based on price promises, cost, presence on approved partners list, as well as binding support, historical performance, quality of service classifications and proximity.

If service requestors query the UDDI operator node for Web service providers, they will have to exercise the most caution and diligence when prescreening potential service providers. An efficient and accurate mechanism should be in place to filter out garbage service descriptions and unworthy service providers.

Consuming Service Descriptions

After a service description is acquired, the service requestor needs to process it to invoke the service. The service requestor uses the service description to generate SOAP requests or programming language-specific proxies to the Web service. This generation can be done at design time or at runtime to format an invocation to the Web service. Various tools can be used at design time or runtime to generate programming language bindings from WSDL documents. These bindings present an API to the application program and encapsulate the details of the XML messaging from the application.

Web Services for Real e-business

While SOAP and HTTP are sufficient for interoperable XML messaging, and WSDL is sufficient to communicate what messages are required between service requestor and service provider, more is needed to cover the full range of requirements for e-business. To fully support e-business, extensions are needed for security, reliable messaging, quality of service, and management for each layer of the Web Services stack. Additional requirements for a Web Services infrastructure include support for service context, conversations and activities, intermediaries, portal integration and service flow management.

The interdependence of these mechanisms to provide reliability and security makes it clear that the two concerns should be part of an integrated strategy.

Security

Is a Web Services security layer really required? The industry already has a set of existing and widely accepted transport-layer security mechanisms for message-based architectures such as Secure Sockets Layer (SSL) and Internet Protocol Security (IPSec), why add another? To answer that question we will examine the requirements and explore several scenarios in which the security provided by the various existing transport layer security mechanisms alone does not provide adequate security in a Web Services model.

In general, there are four basic security requirements that the Web Services security layer must provide:

- **Confidentiality** is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes, and guarantees that the contents of the message are not disclosed to unauthorized individuals.

- **Authorization** is the granting of authority, which includes the granting of access based on access rights and guarantees that the sender is authorized to send a message.

- **Data integrity** is the property that data has not been undetectably altered or destroyed in an unauthorized manner or by unauthorized users thereby insuring that the message was not modified accidentally or deliberately in transit.

- **Proof of origin** is evidence identifying the originator of a message or data. It asserts that the message was transmitted by a properly identified sender and is not a replay of a previously transmitted message. This requirement implies data integrity.

The need to manage different styles of resource access in a dynamic world of Web Services based on XML messaging and workflow necessitates a reevaluation of the relationship between policy, trust and risk assessment. Existing access control models based on an individual identity are evolving into a role-based trust domain relationship in which an individual is acting under the authority granted to it by a trusted authority to perform a particular task. The IBM Web Services architecture defines agents that want information (service requestors) and agents that provide information (service providers), and sometimes agents that provide information about information (service brokers, meta information providers or service registries). The service broker gets a lot of requests for its information and it needs to be able to decide who wants what and whether or not they are granted access. Infrastructures and relationships change quickly and the policies governing them need to be flexible in allowing or denying access.

And, while XML holds the promise of providing a common interface to such services, it does not provide the entire infrastructure needed to implement such a vision. Further, XML might not be appropriate for building the entire Web Services security layer. The goal is to identify where it is important to provide information in an XML format to allow for common data exchange, and where it is important to utilize the existing security mechanism that exist on platforms today.

The SOAP envelope is defined in XML and enables you to add a large variety of meta-information to the message, such as transaction IDs, message routing information and message security. The SOAP envelope consists of two parts: header and body. The header is a generic mechanism for adding features to a SOAP message. All immediate child elements of the SOAP header element are called header entries. The body is a container for application data such as RPC intended for the ultimate recipient of the message. Thus, SOAP can be considered to introduce another layer between the transport layer (for example, HTTP) and the application layer (for example, business data), which is a convenient place for conveying message meta-information.

The SOAP header provides an extensible mechanism for extending a SOAP message for many uses. The SOAP header is the most rational place to add security features to messages, but the SOAP specification itself does not specify such header elements.

Let's take a closer look at why the various existent transport layer security mechanisms are not sufficient in a Web Services model, why there is a requirement for a Web Services security layer and what that layer looks like initially.

End-to-End. Secure transport protocols such as SSL and IPSec can provide the integrity and confidentiality of the message during transmission, but they do so only for point-to-point. However, because SOAP messages are received and processed by intermediaries, secure end-to-end communication is not possible if there is no trust association among all the intermediaries, even though the communication links between them are trusted. End-to-end security is also compromised if one of the communication links is not secured. In looking at the Web Services topologies, secure transports are not sufficient for end-to-end security of the SOAP message.

Middleware Independence. Ultimately, the only way to provide end-to-end security is at the application or middleware level. If there is any point between the communicating parties at which messages are in plain text, it can be a potential point of attack. However, it is not an easy or desirable task to integrate cryptographic functionality into a new or existing application without introducing additional security vulnerabilities and increasing risk. It is thus desirable, in most cases, to have security functionality as close to the application as possible but not built in the application itself.

Transport Independence. One of the intended uses of SOAP intermediaries is to forward messages to different networks, often using different transport protocols. Even if all the communication links are secured and the intermediaries can be trusted, security information such as the authenticity of the originator of the message needs to be translated to the next security domain of the transport protocol along the message path, which could be tedious and complex, and could lead to integrity flaws.

Asynchronous Multihop Messages. Transport layer security secures the data when traveling on communication links. It has nothing to do with data stored on any intermediaries. After a transmission is received and decrypted, transport layer security does not help much in protecting the data from unauthorized accesses and potential alterations. In situations where messages are stored and then forwarded (persistent message queues), message layer protection is necessary.

Because we have seen that secure transport mechanisms are not sufficient to meet the requirements of a Web Services development approach and usage scenarios, the task is to create a conceptual Web Security layer that consists of the following components:

1. For network security:
   a. Support for secure transport mechanisms such as SSL and HTTPS that provide confidentiality and integrity.
2. For XML messages:
   a. If communication does not have intermediate hops, the sender can rely on SSL or HTTPS for user ID and password confidentiality.
   b. Support for the XML Digital Signature work is being standardized by the W3C. It defines a standard SOAP header and algorithms for producing a message digest and signing the message with the sender's private key. Thus a recipient can verify the identity of the originator of the message.
   c. Support for in-network, trusted third party authentication services (for example Kerberos).

The conceptual XML messaging model must also support end-to-end protection of the message and its subelements. To do this comprehensively, the process and flow capability needs to be extended to include security characteristics of message exchanges. There should be a way to define a multisegment message and to protect the segments with the public keys of the intended recipients. Some of the topics that need to be explored are:

- The endpoint is responsible for implementing authentication and authorization. There should be support in any description of agreements for exchanging information between enterprises to define which employees can use which services. Intermediaries are responsible for audit and proof of origin. Intermediaries might also need to perform authentication, authorization, and digital signature verification and validation

- Security-oriented meta-data to support the security issues documented above needs to be defined in the service description layers for a service endpoint. These security descriptions will define Web service-level access control, by principal or by role. The service description will describe if and how digital signing, encryption, authentication and authorization have been supported.
- Requestors will use the security elements of a service description to find service endpoints that meet their policy requirements and match their security methodologies.

Standards groups are investigating the following topics and technologies. As these standards solidify, they will be incorporated into the Web Services security architecture.

- The W3C has a working group for XML Encryption, which will help provide confidentiality of data elements, so that an authentication exchange can be possible.
- The W3C has published a note on XML Key Management Services (XKMS) which will help distribute and manage the keys necessary for endpoints to communicate securely.
- OASIS has established a technical committee to define Authorization and Authentication assertions (SAML). This will help endpoints accept and assert access control rights.
- OASIS has established a technical committee to standardize on the expression of access control rights (XACML). This will help endpoints be able to parse the SAML assertions in a consistent manner.

Web Services security architecture is evolving as we continually examine all the threats and countermeasures that are encountered in the Web Services model.

Quality of Service and Reliable Messaging

The Quality of Service vertical tower provides for the specification of information relevant to each of the layers of the Web Services conceptual stack. For the network layer, this would imply being able to use networks of various levels of quality of service.

The need for reliable messaging over the network will drive the choice of network technologies based on their ability to deliver a high quality of service in this area. Reliable messaging refers to the ability of an infrastructure to deliver a message once, and only once, to its intended target or to provide a definite event, possibly to the source, if the delivery cannot be accomplished. The combination of the network layer and XML messaging will need to support four levels of messaging qualities of service:

1. **Best-Effort:** The service requestor sends the message, and the service requestor and infrastructure do not attempt retransmissions.
2. **At-Least-Once:** The service requestor makes a request and retries the request until it receives an acknowledgement. Duplicate message processing by the service provider is not a problem, for example, simple query processing. In implementation this might mean that each message contains a unique ID. The service requestor retransmits unacknowledged messages at an interval that it decides. A service provider sends an acknowledge message, response messages for an RPC and a *cannot process message exception* if it cannot process the request.
3. **At-Most-Once:** This builds on the at-least-once scenario. The service requestor tries the request until it gets a reply. A mechanism like existing universal unique identifiers (UUIDs) allows the service provider to suppress any duplicate requests, insuring the request is not executed multiple times. For example, a request to take an item from a number in an inventory.
4. **Exactly-Once:** The service requestor makes a request and is guaranteed by the reply that the request has been executed. The exactly-once mode of interchange eliminates the need to retransmit requests and accommodates failure scenarios.

Reliable messaging is usually delivered through a standard design pattern in which a piece of infrastructure, sometimes called an endpoint manager, is employed at each end of the communication to coordinate the message delivery. In this pattern, the sender delivers the message to the endpoint manager via a synchronous request. Once delivered to the endpoint manager, the sender can be assured that the message will be delivered or a definite event will be raised (for example, a time out). The endpoint manager participates in local transactions with other resource managers so that *queuing* the message with the endpoint manager and possibly recording the business process step in a database can be done in one transaction. The application should delegate the responsibility for delivering messages, or detecting the failure to do so, to the endpoint manager, which can function at the network transport level or at the XML messaging level.

The technology and goals of reliable, one-time message delivery are not in debate. However, an important question has been raised as to how to support it in the context of SOAP and XML. The key question is: Should the necessary protocols and message formats be defined at the XML message level, thus allowing the reliable message delivery to be the responsibility of the two end applications, or can the protocols and message formats be defined at a lower, transport, level?

Where a transport that supports reliable messaging is not available, that is, the Internet, the XML messaging layer will need to support these qualities of service over the nonreliable infrastructures. The endpoint managers will need to modify the message, rather than just its transport envelope, to fulfill their role. The applications and business process definitions would have to be concerned with all possible outcomes, such as the rejection of a message, or the inability to deliver it in an acceptable amount of time. However, they would also need to be concerned with the intermediate states that can occur in the delivery process. To expose these states to the business process would greatly complicate its definition in ways that would not be meaningful to the business analysts defining the process. To use XML messaging to deliver reliable messaging formats would make the use of these existing transports very inefficient in some cases. It would be preferable for a reliable HTTP standard to be developed for use in the Internet.

Where a transport that supports reliable messaging is available, that is, within an enterprise, it could be used to deliver reliable messaging instead of the XML messaging layer (which could default to a null implementation). The endpoint managers would not modify the XML message, only the transport envelope. Using a reliable transport isolates the applications and business process definitions from needing to be aware of or handle intermediate states of message delivery.

In the future, several enhancements will be necessary:

- HTTP for the Internet will need to be improved to provide simple reliable messaging for use between enterprises. This will provide the added benefit that reliable messaging will be available to many message types and not just SOAP. This will reduce the need for the XML messaging layer to handle reliable messaging and promote application development independently of network choices.

- The XML messaging layer over HTTP will also need to address publish and subscribe, message ordering, delivery time constraints, priorities, multicast, and so on.

What quality and implementation of reliable messaging a service provider supports will be defined in the binding information of the service description.

Quality of Service information is also relevant in the service descriptions at the service implementation level (for example, bind via transactional or secure SOAP) and others at the interface level (for example, maximum duration after the requestor expects the provider to respond.). It is anticipated that WSDL extensions or new service description layers will be developed to allow specification of other qualities of service and capabilities.

Quality of Service at the Web service level would be used in service composition and service flow. Expected execution time, timeout values and historical execution averages could all be input in service selection for a flow or signaling a flow manager that it is time to initiate recovery or alternative flows. The endpoint description and workflow description layers of the service description stack will need to provide this information.

The Quality of Service issues and solutions for Web Services are still emerging.

Systems and Application Management

As Web Services become critical to business operations, management of them will be required. Management in this case means that a management application, either custom built for the application or purchased from a vendor, can discover the existence, availability and health of the Web Services infrastructure, Web Services, service registries and Web service applications. Optimally, the management system should also be able to control and configure the infrastructure and components.

It must be possible to manage Web Services at all levels of the conceptual Web Services stack and the Web Services model components. The need for management can be broken down into two focused areas. The first is the manageability of the infrastructure used to implement Web Services. The major concerns are to ensure availability and the performance of key elements that supply the service description, the messaging and the network. The infrastructure providers of Web Services should provide this level of systems management.

Enterprises will have full autonomy over their own infrastructure and management. However, when enterprises interact with each other on a peer basis, they should provide basic reporting and recovery for the network layer, XML messaging layer, service registries, and Web service implementations. Furthermore, the management interfaces that an enterprise provides to its partners should operate at the service level, and not at the relatively low level of the infrastructure. Partners should have access to an interface that reports on the status and health of operation and request processing, without having to understand the details of how an enterprise manages its requests.

For the network layer, existing network management products support nearly all current network infrastructures. These products should be used to manage the network infrastructure for Web Services within the enterprise. When enterprises interact with each other, they should provide basic reporting of Web Services infrastructure availability to their partners. Network availability that impacts the Web Services infrastructure availability should be factored into this reporting.

At the XML messaging layer, the protocol should be managed within the enterprise by existing infrastructure management tools. Where enterprises interact with each other, it will be necessary for each site to provide basic reporting and recovery for the protocol. For example, if site A supports conversations, it should provide an interface that site B can use to query for active conversations and force rollback. A normal channel and protocol, and a peer-wise control interface will be necessary for the protocol level.

The second aspect of management is the manageability of the Web Services themselves. Some of the major concerns are performance, availability, events, and usage metrics that provide the service provider market with the necessary information to charge for the use of the services they offer.

Service descriptions can be used to advertise manageability characteristics and management requirements. Conventions for this are under development.

Any implementation of a service registry, be it for private or public consumption requires that the infrastructure be available, deliver the promised quality of service and be able to report on usage. These elements of systems management are fundamental to the successful adoption of UDDI.

For the components of a Web service application, supporting a management environment can add significant complexity to an application. Because Web Services must be easy to develop, this complexity must be hidden, as much as possible, from the developer. The Web Services approach to management is to supply infrastructure to provide metrics automatically, audit logs, start and stop processing, event notification, and other management functions as part of a Web Services runtime (that is, at least the SOAP Server). Because not all information can be gleaned by the infrastructure through observing the behavior of the components it hosts, a Web service implementation might need to provide basic health and monitoring information to its hosting server.

The Web Services infrastructure should provide an easy way for services to participate in management and to leverage management infrastructure. A manageable service WSDL document will be defined that a Web service can implement to provide access to a Web service's management information by management systems. This interface would include the ability to get configuration and metric data, update configurations, and receive events from a manageable Web service.

The platform independence of the Web Services architecture makes it inappropriate to impose any single Web Services management standard. Therefore a Web service-based approach for allowing Web Services to communicate with a management system is needed. To this end, a management service described by a WSDL document that can receive events and metric updates from manageable Web Services should also be defined and available. The implementation technology for the management service would be irrelevant to the Web service. However, for Java technology-based environments, Java Management Extensions (JMX) would be a logical, vendor agnostic choice. By using an open standard such as JMX, it should be easy for existing systems management providers to extend their current offerings to include the management of Web Services key elements.

The management architecture for Web Services is still evolving.

Service Context

Intelligent Web Services are generally Web Services whose behavior is consistent with their environment's current state. Optimally, this behavior should be self-configuring. While it is the responsibility of the Web service provider to code an intelligent application, it is difficult to do so without information about the runtime environment's current state and context. The XML messaging layer needs to define support for propagating context information on calls to Web Services. Some examples of context information are

- The type of wireless device on which the application executes and its Global Positioning System (GPS) coordinates.
- A reference to one or more *user profiles* that define the user's preferences.
- Current time and condition information from the service requestor's perspective.

The XML messaging layer *per se* is not responsible for defining the details or schema of preferences and contexts. Rather, the model requires an approach for flowing and propagating contexts in an extensible way, which allows specific industries, coalitions and so on to define their context schema and semantics.

Conversations and Activities

In the loose sense, *transactions* are one of the most fundamental concepts in application processing. For many reasons, the Web Services model requires a more flexible mechanism for controlling requests and outcomes than typically offered by traditional distributed and database transaction models. Some specific differences between the Web Services model and transaction processing monitor programming models are:

- Multienterprise Web service collaborations that often rely on asynchronous messaging. While this model also occurs in traditional, intra-enterprise applications, distributed transactions do not occur between messaging systems. Instead, these applications assume a chained, multiple-transaction model.
- Cross-business applications inherently have multiple spans of control. During the processing of a traditional transaction, the infrastructure dedicates resources to the executing transaction (database lock, threads, and so on). Enterprises rely on sophisticated monitoring and management tools to ensure that transaction processing does not introduce stall and service-denial failures into their infrastructure. Providing similar support to a multiple enterprise, dynamic binding environment is theoretically possible, but impractical.
- Online transaction processing and database systems optimize execution of high-volume, short-duration transactions and conversations. Multiple-enterprise Web service collaborations will typically have far longer duration than transaction and database applications.

Instead of simply extending existing transaction models to the Web, an incremental approach to transactional quality of service is required:

1. A core **Activity Service**, which includes a generalized capability for specifying the operational context of a request (or series of requests), controlling the duration of the activity and defining the participants engaged in an outcome decision, is needed. An example of such a service is typified in the OMG Extended Structuring Mechanism, which is also being adopted within Java under the Activity Service JSR.
2. A **Conversation Service** that provides the fundamental conversation styles or behaviors that need to be constructed for Web Services using the activity service is needed. These fundamental styles are:
   - **Under Request Atomicity** is a single operation on a Web service occurs either completely or not at all. This is a capability that the endpoint manager publishes to users. The endpoint manager can implement this by an internal transaction on its infrastructure or by some other mechanism.

   - **Conversations** allow a pair of collaborating services to correlate sequences of requests within a loose unit of work. The pair of services uses architected conversation messages and headers to begin the conversation and end the conversation. They determine if the conversation ended successfully or if one or both participants want the conversation to rollback. The semantics of *rollback* is that each participant will undo the operations it has performed within the conversation.

These styles or behavior mechanisms are by no means complete. However, they allow sites participating in Web Services to supplement their existing implementations to support more complex processing and compensation models. The sites can build on their internal transaction and business logic environments to provide the increased flexibility. Additionally the activity service would allow for additional operational patterns beyond the traditional transactional model.

The sole requirement currently placed on Web Services is the ability for two endpoints to agree on the outcome. Multiparty transaction processing occurs within enterprises that use their own internal transaction and workflow models to manage operations and conversations with multiple participants.

Intermediaries

One enhancement that Web Services will provide is support for *intermediaries*. An intermediary is a component (potentially itself a Web service) that insinuates itself between a service requestor and service provider during an invocation. Literally the intermediary *intermediates* the Web service invocation. An example of an intermediary is a third-party notary site that timestamps and logs a service invocation. This intermediary can be used to support nonrepudiation.

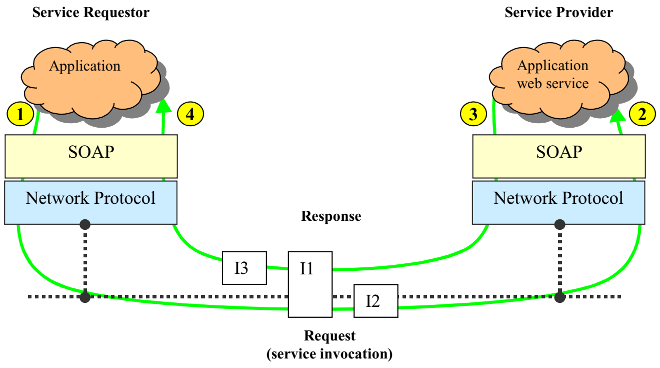

Figure 9. Intermediaries

Intermediaries can be placed in the execution path of the request and the response message flow. The intermediary, as in the case of I1 in Figure 9, can be invoked during both the request and the response. I1 can be a third-party Quality of Service (QOS) monitor or audit-trail intermediary. Certain intermediaries will only be invoked on either the request or the response message flow, as in I2 and I3 above. I2 can be a nonrepudiation intermediary and I3 can be a response-time logging intermediary. The specifications for defining the set of intermediaries on the message path are still being defined.

Intermediaries will be supported by the AXIS implementation of SOAP available from Apache. A service provider that publishes a service could specify which intermediaries it supports, the service they provide and their identity. The ability to search for and inspect service descriptions by interface type and by implementation type provides the meta-data necessary for site A to examine the intermediary chain that site B supports for a service. Some examples in the intermediary chain might be a nonrepudiation service and a billing and payment service. Intermediaries will provide qualities of service that the endpoints agree to, for example exactly-once semantics. In the billing and payment scenario, if an endpoint is ensured to be invoked exactly once, you would like to see exactly one payment position on your bill for that invocation.

Site B can specify a workflow graph that defines the ordering of required intermediaries. When site A resolves a service to site B, the workflow graph states that invoking the service on B occurs by invoking the service on an entry node in the flow, which will result in the service eventually occurring at B after the intermediaries process the message. The flow definition also specifies which data must by sealed for delivery to which intermediaries in the chain.

The actual representation of intermediaries with the XML messaging layer is not well defined in SOAP and is currently a topic of debate within the XML Protocol Working Group.

Portals and Portlets

Portals are commonly used to provide people with access to information and applications in a condensed form. Typically, portals display personalized information from various sources in a single page, thus allowing the user to efficiently access this information instead of visiting various Web sites one after the other. Depending on customizable user settings, various portlets—usually rectangular areas that display information—are included in the page.

As Web Services become the predominant method for making information and applications programmatically available on the Internet, portals and portal development tools need to allow for integration of Web Services as data sources. Figure 10 illustrates some common data flows between portals, portlet services and Web services.

There are initially two integration points for Web Services and portlets: portlets that use Web Services as a backend and portlets that are described, wrapped and published as Web Services. An example of the first, which will be the predominant scenario, is a news portlet that allows the user to configure the news categories to track and then gets the news for these categories live from a Web service whenever it is displayed. In this case, the portlet code runs locally on the portal and uses the Web service to access information. The rendering of the content is done by the portlet itself.

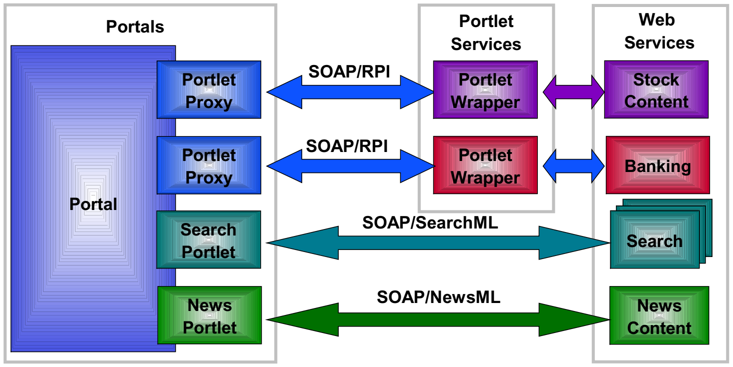

Figure 10. Portals and portlets

An example of the second use of Web Services by portals is enabling sharing of portlets with other portals. In this scenario, a remote server that is another portal publishes portlets as remote portlet Web Services in a Web Services directory. The portal can thus find the remote portlet services in the directory and bind to them. As a result, the remote portlets become available for the portal users without requiring local installation of the portlet code on the portal itself.

Other opportunities for integration of Web Services and portal technologies will emerge as these technologies mature.

Business Processes, Workflows and Web Services

Web Services are composable; this is an important characteristic of Web Services identified previously in this paper. Workflow, as a primary mechanism to compose Web Services in a nontrivial fashion, is critical for the rapid creation of new, higher-function Web Services. Workflow will provide choreography for interactions between component Web Services.

Business processes are graphs of activities that carry out some meaningful business operation. Examples are purchasing an airline ticket, managing inventory in a warehouse, and ordering furniture for a home or office. Long-running transactions, such as tracking an order to fulfillment or supporting collaborative planning, forecasting and replenishment (CPFR) are also business processes. Business processes vary in level of granularity, and the details of a business process will vary from enterprise to enterprise.

Workflows are business processes that are run in an IT environment using a tool such as IBM MQSeries Workflow. Workflow tools allow businesses to define each of their business processes as a series of activities carried out by individuals or applications, and vary the sequence through the activity series depending on the output data from each individual activity. For more information on business processes and workflows, see *Web Services and Business Process Management Technology*.

As described above, Web Services are attractive components of workflows, because they can be composed dynamically or orchestrated into workflows, and because they are widely available across the network. This section describes the fundamental architecture of workflows in the context of Web Services and in particular:

1. Illustrates the ways in which workflows can be combined and transformed into Web Services.
2. Illustrates the fractal nature of Web-service workflows—how each Web-service-based activity participating in a workflow can in turn be composed of a lower-level workflow.
3. Introduces the concept of *enabling services* to support e-business Web Services, and illustrates how such enabling services can be combined with e-business Web Services into workflows to support customer processes.
4. Introduces the concept of a business process hierarchy with activities in public workflows being implemented by private workflows, whose activities are implemented in turn by workflows implemented as Java components bound together by scripts.

This section deals primarily with workflows as a means of composing and choreographing Web Services across enterprises. However, workflow is not the only technique for composing or choreographing Web Services within an enterprise. Scripting in JavaScript™ and microflows can also be used for this purpose. For a good discussion of these alternative techniques, see *IBM Web Services Roadmap*.

A Simple Web Services Workflow

Figure 11 illustrates a simple workflow involving Web Services.

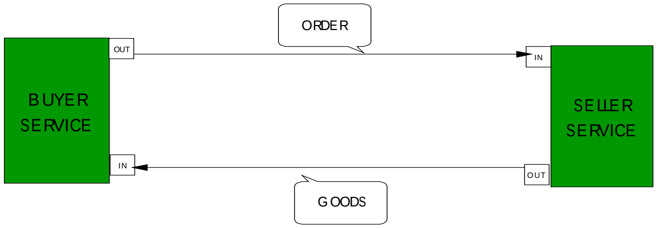

Figure 11. Simple workflow

In Figure 11, a buyer service (which might be a simple client) is ordering goods from a seller service. The seller service is a Web service whose interface is defined using WSDL. The buyer service is invoking the order method on the seller service using SOAP and the WSDL definition for the seller service. The buyer service knows what to expect in the SOAP reply message because this is defined in the WSDL definition for the seller service.

Figure 11 shows in and out boxes for each Web service involved in a workflow. In general, the in and out boxes are defined using WSDL, as described previously in this paper. The tool that encapsulates code to create the buyer service derives the format of the outbox or API for the seller service by analyzing and transforming the WSDL description of the seller service, and also knows what to expect from the seller service by analyzing the WSDL description. The in and out boxes in Figure 11 are symbolic representations of this analysis activity.

e-business Services and Enabling Services

Figure 11 illustrates a simple workflow involving two Web Services (or a Web service and a service requestor application, depending on the nature of the buyer). Now consider a slightly more complex workflow, as illustrated in Figure 12.

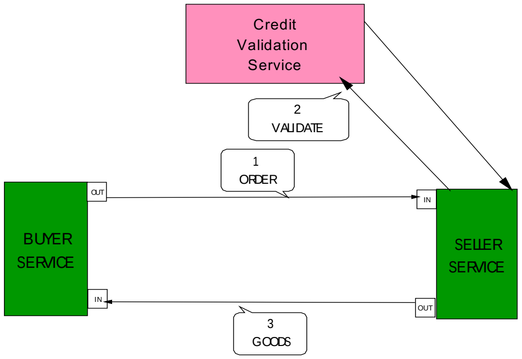

Figure 12. More complex workflow

In Figure 12, the seller service uses a credit validation service to ensure that the buyer can pay for the goods, before shipping the goods to the buyer. The credit validation service is a third-party Web service that is available to e-business services in general.

In this example, the buyer service and the seller service are *e-business services* that do activities directly associated with an enterprise's business. The credit validation service, on the other hand, is an *enabling service* that can be used by many e-business services to help them perform their business processes. Other examples of enabling services are a satisfaction tracking service, a security service, and a workflow service to host private workflows for companies without their own infrastructure for doing this.

Composed Workflows, and Public and Private Workflows

A Web service that serves as an activity in one workflow can itself consist of a series of sequenced activities (or a workflow) as illustrated in Figure 13.

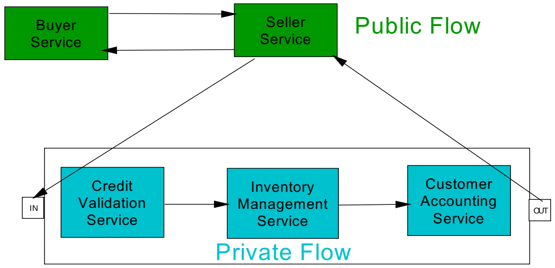
Buyer Service
Seller Service
Public Flow
IN
Credit Validation Service
Inventory Management Service
Customer Accounting Service
OUT
Private Flow

Figure 13. Composed workflow

The seller service in Figure 13 is actually a complete workflow that is encapsulated as a single Web service. The seller service consists of a credit validation activity, an inventory management activity and a customer accounting activity. The seller service presents a single interface to the buyer service using a single WSDL definition, thereby hiding the details of the lower-level workflow that it encapsulates. In Figure 13, the enterprise providing the workflow for the seller service does not expose the details of this service to public applications and services that seek to use the seller service. That is to say that the seller service participates in a *public flow*, whereas the workflow that makes up the seller service is a *private flow*.

This idea of composing an activity or Web service out of a workflow is very powerful, and can be applied at multiple levels of granularity. Figure 13 illustrates three levels of workflow granularity.

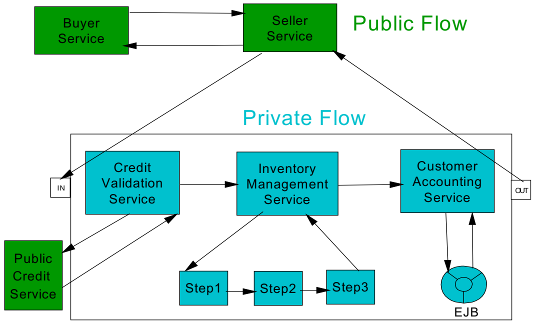

Figure 14. Further composition of workflows

In Figure 14 the inventory management service itself is shown to consist of a workflow including several steps. Note also that in Figure 14 the credit validation service that is part of the workflow for the seller service actually goes out on the public Internet and utilizes a public credit service, which it might find and bind to using information retrieved from a registry such as UDDI.

Finally, Figure 14 illustrates that underneath all Web Services are coded applications. In Figure 14, the customer accounting service is actually an encapsulated Enterprise JavaBean™ (EJB), which in turn is part of a customer application.

Business Process Hierarchy of Workflows

Figure 14 illustrates how workflows might exist at different levels. Figure 15 is an elaboration of Figure 14 that shows how workflow might exist even at the code level. Figure 15 also illustrates how workflows might map to actual middleware products.

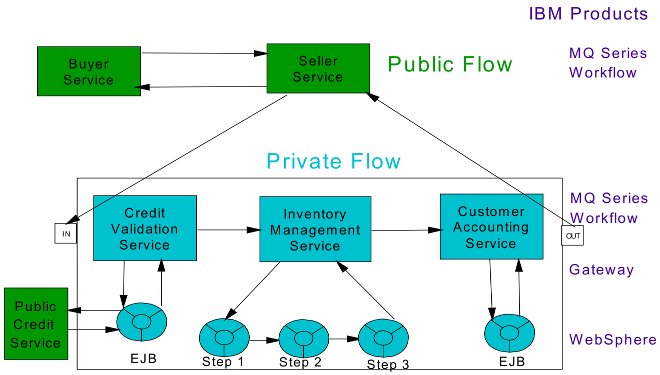

Figure 15. Business process hierarchy

In Figure 15, the activities in the inventory management service have been replaced with (EJBs) and access to the public credit service is via an EJB encapsulated to serve as a Web Services client application. Figure 15 also calls out some of the IBM middleware products that will support the different layers in the business process hierarchy.

MQSeries Workflow is a middleware product from IBM that might in the future support the WSFL that defines workflows composed of Web Services, and includes Web Services from lower-level workflows. IBM will work with its partners to standardize a language that serves these needs. WSFL is the IBM input to this standardization process. After a workflow definition language such as WSFL is standardized, IBM and other companies will likely support it. For more information on WSFL, see *Web Services Flow Language (WSFL 1.0)*

At the lowest level illustrated in Figure 15, EJBs carry out the actual services associated with workflows—either directly or by accessing existing customer applications via connectors. These EJBs run in containers provided by IBM WebSphere Application Server and can be created using IBM VisualAge® for Java and encapsulated into Web Services using IBM WebSphere Studio Technology Preview for Web Services (or other appropriate tools).

The private flow illustrated in Figure 15 might be initiated over SOAP and HTTP, or it might run over MQSeries, which also supports SOAP and provides additional security and guaranteed delivery. In general, a gateway will handle impedance mismatches between the external public flow using SOAP over HTTP and intra-enterprise internal flows that can take advantage of information buses such as MQSeries and the RMI over IIOP support provided by IBM WebSphere® Application Server. For more detail, see *IBM Web Services Roadmap*.

Finally, note that in Figure 15 the three EJBs implementing the steps of the inventory management services are actually doing a workflow. In principle, this workflow can be described using WSFL, and the sequence could be carried out as a microflow. This type of workflow involves actual code components and is obviously related to the other levels of flow illustrated in Figure 15. IBM plans to coordinate its VisualAge and Studio tools to facilitate the visual creation of workflows at all of the different levels illustrated in Figure 15, going from public workflows to private workflows to microflows. For a discussion of this topic see *IBM Web Services Roadmap*.

Hierarchical Workflows and Peer-to-Peer Workflows

The workflows we have shown thus far in this paper have been hierarchical, in that they are self-contained, with all business process activities being part of the same workflow, though in some cases the Web service representing an activity was itself a workflow. Peer-to-peer workflows involve two or more independently executing workflows having conversations to complete the overall workflow. For example, to accomplish what it needs to do, a Web service that is an activity in one workflow might send a message to a Web service that is an activity in another workflow at the same level and wait for a reply before continuing. For an example of a peer-to-peer workflow, see the ticket-order example in the companion paper *Web Services Flow Language (WSFL 1.0)*.

Web Services Workflows Today and Tomorrow

The standards efforts for Web Services workflows are not as far along as some of the other Web Services standards initiatives such as SOAP (XML Protocol) and WSDL. However, IBM and other companies are working on standards for Web service flow definition and Web Services workflow endpoint definition. These standards should be agreed upon and approved quickly, as were the standards for SOAP and WSDL. The IBM candidate for Web-service flow definition is the Web Services Flow Language.

After the standards have been approved, IBM and other middleware vendors should quickly provide tools and runtime support for Web service-oriented workflows as described in this paper. In the meantime, a great deal can be done to begin implementing workflows using existing Web Services technology such as WebSphere Technology for Developers, MQSeries Workflow, MQSeries Integrator, and WebSphere Business Integrator. To read more about creating Web service workflows now, see *Web Services and Business Process Management Technology*.

Related Information

Web Sites

AXIS	http://xml.apache.org/axis
DISCO	http://msdn.microsoft.com/xml/general/disco.asp
EbXML	http://www.ebxml.org/
OAG	http://www.openapplications.org/
SOAP	http://www.w3.org/TR/SOAP/
UDDI	http://www.uddi.org/
XMLP	http://www.w3.org/2000/xp/
WSDL	http://www.uddi.org/submissions.html
XML Schema Part 1	http://www.w3.org/TR/xmlschema-1/
XML Schema Part 2	http://www.w3.org/TR/xmlschema-2/

Other Papers

Web Services Flow Language (WSFL 1.0)

Web Services Conceptual Architecture (WSCA 1.0)

IBM Web Services Roadmap

Web Services and Business Process Management Technology

International Business Machines Corporation
Software Communications Department
Route 100, Building 1
Somers, NY 10589
U.S.A.

05-01

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.