Web Services: Why and How

Francisco Curbera, William A. Nagy and Sanjiva Weerawarana
IBM T.J. Watson Research Center
{e-mail: curbera, wnagy, sanjiva @us.ibm.com }

August 9, 2001

Abstract

Web services are a natural consequence of the evolution of the Web into an open medium which facilitates complex business and scientific application interactions. Web services are concerned with the problems of enabling systematic application-to-application interactions over the Web, and the integration of the existing network computer infrastructure into the Web. The key components of this work are a focus on interoperability, support for efficient application integration, and the creation of a uniform representation of applications within heterogeneous distributed systems.

The Web services model builds upon existing industry standardization efforts centered around the Extensible Markup Language (XML), and proposes a two pronged approach to deal with the interoperability requirements associated with heterogeneous systems: enable base interoperability using a small set of common protocols, and develop a uniform representation of network applications which are accessible using multiple communication protocols. The goal of Web services is to provide a flexible framework where universal interoperability does not preclude efficient integration.

1 Introduction

Web services are a natural consequence of the evolution of the Web. Since its beginnings as a way to share and distribute information on a global scale, effectively becoming a giant distributed content library, the Web has been progressively widening its reach to enable more sophisticated forms of interaction between browser clients and servers: single form-based interactions, retail e-commerce applications, and more complex business-to-business interactions. All of this has been accomplished without changing the Web's fundamental nature as a medium to enable human-to-application interactions. Lately, however, true Web-based application-to-application interactions have started to take shape, first in connection with the development of electronic marketplaces and automated business-to-business transactions, and, more recently, with the goal of enabling large-scale resource sharing [1]. Two questions arise naturally: what is the appropriate model to support Web-based application-to-application interactions in a systematic way, and how will this new distributed computing platform be integrated with pre-existing environments, such as those already present within private networks. The Web services model is an attempt to provide an answer to these questions.

Providing a satisfactory solution to these problems requires confronting the issue of how best to deal with a very heterogeneous distributed system. There are two reasons why dealing with heterogeneity is central to Web services. The first is the nature of the Web as a decentralized medium in which multiple platforms and programming models coexist. The second is the additional flexibility needed to seemlessly integrate with existing models.

The ability to function without centralized planning or control (except for the most basic services, such as domain name management) and to integrate a heterogeneous mix of platforms and programming models is one of the most remarkable characteristics of the Web. Its success is due, in part, to the fact that this problem was recognized and confronted from the early stages of design: interoperability was stipulated based upon the convergence to a minimal set of standards, HTTP as the basic application level communication protocol, and HTML markup for the formatting of information. It is important to observe that the convergence to this small set of standards has not implied the convergence to a common programming model. The result is a very shallow interaction model between a very heterogeneous set of clients and servers.

The adoption of a new paradigm of application integration inevitably affects the usage of other existing models, even when their domains of applicability are clearly differentiated. In particular, there is a positive tendency to simplify the deployed infrastructure, which can be successful only if it doesn't compromise the quality of service, and only if it can be carried through with little or no replacement of existing infrastructure. From the Web services design perspective, the issue at stake is again how to deal with system heterogeneity. This time, however, we consider systems that are heterogeneous in their interaction models, communication protocols, and quality of service levels.

The Web services approach to the problem of heterogeneity is two-fold: first, define a minimal set of standards to assure basic interoperability; second, provide a unified view of applications within heterogeneous systems that enables exploitation of alternate interaction models when available for both parties in the interaction. Like other distributed systems, Web services builds upon two fundamental blocks: a data exchange protocol and a set of metadata for the functional representation of Web applications:

1. **Basic interoperability at the information exchange level**. Web services interoperability is built on top of the Extensible Markup Language (XML). In the Web services model, the most basic level of application interaction involves the ability to exchange XML data. A simple and standard XML protocol [2], pervasively deployed, and which is able to be carried over the major Web communication protocols, assures Internet-wide application interoperability at the most basic level.

2. **Unified representation of applications within heterogeneous systems**. An interface definition language provides a common, platform independent, functional representation of applications. In the Web services framework [3], the IDL representation uses XML Schema as its data definition language, and application functionality is described in terms of the XML messages that can be exchanged.

   The use of an XML centered representation provides platform independence. To support heterogeneity at the interaction level, an abstract functional description is separated from its possible "protocol bindings". In this model, an application has a single abstract representation, but can be accessed by using one of several protocol bindings; interacting partners are able to select the optimal protocol from the application description, based on the available bindings. Abstract representation provide the basis for supporting protocol heterogeneity.

There are two additional concerns guiding much of the design of the Web services framework. One is to assure that the framework is able to take advantage of the open, dynamic nature of the Web by supporting just-in-time application integration. This typically requires late binding mechanisms and a seemlessly integrated application registry infrastructure in which applications can be selected based on their technical characteristics.

A second design concern relates to the enablement of a component-oriented development model for applications in the Web services framework. A component development model fits well in a framework of encapsulated applications which interact through well defined interfaces, and can draw many of the software engineering benefits associated with component-oriented software development [4]. In this model, Web-enabled applications are represented as "Web components", and the task of programming against these applications becomes a problem of component composition.

The rest of this paper is organized as follows. In Section 2 we will provide a brief characterization of Web service applications, Web services. Section 3 describes the data interoperability and uniform representation mechanisms of the Web service framework, and introduces other lines of development of the framework. In Section 4 we present the conclusions of the paper.

2 What is a Web Service

In this section we will try to characterize the nature of applications in the Web services framework. The term "Web service" is commonly used to identify these applications. It must be stressed that the actual definition of the Web services framework is not complete, and that it is a wide industry and standardization effort. The characterization of Web services presented here must be understood as a statement of what we consider to be the target application model that this effort will eventually produce.

The following "definition" tries to summarize three key pieces of the Web services framework: interoperability, common representation, and a heavy reliance on standards. Otherwise, the definition presented next is extremely open, and is in line with the aim of the Web services effort.

A Web service is a networked application that is able to interact using standard application-to-application Web protocols over well defined interfaces, and which is described using a standard functional description language.

It is not uncommon to see other characteristics stressed in definitions circulating in the industry. Accordingly, this definition is sometimes extended with a mention of the possibility of "discovering" Web services based on their technical properties, using Web services registries. In another, even more inclusive view, Web services are not require to support any particular protocol, and any application whose description is provided in a common description language ([3]) is deemed to constitute a web service.

We conclude our characterization with a set of properties that a Web service is expected to display.

1. "Gray box" components. Web services are encapsulated applications which are fully described by their description files. It is likely that the current interface descriptions will be extended in the near future by adding behavioral information (see in particular [5, 6]), and a wealth of non-functional characteristics necessary to support the some of the most basic Web interactions (such a security properties, transactionality, quality of service and others).

2. Loosely coupled interaction model. Loose coupling is a consequence of the heterogeneity of platforms, and is reflected in the nature of the interoperability protocols.

3. Flexible integration. Based on service descriptions, it is possible for a Web service (or the supporting middleware) to detect situations when a more efficient protocol (such as IIOP) is supported by both ends of the interaction. The level of integration between the two applications can then be extended to the level supported by the common protocol.

4. Messages instead of APIs. Instead of assuming common interfaces, the model relies on using message and document formats as the basis for describing services and for industry standardization. The focus shifts from the APIs supported by each platform to what is exchanged ("goes on the wire"). In this environment, message brokering and translation become key mechanisms for providing service interoperability.

3 A Web Services Framework

To assure widespread adoption, new Web-based frameworks must be standards based. XML extends and formalizes the markup model of HTML, and provides a means to represent data in a structured way. As such, XML has become the natural successor to HTML for representing information on the Internet, and is the industry standard upon which future Web technologies are being built.

It has been remarked that XML is verbose and that requires intensive parsing. However, verboseness actually becomes one of its greatest strengths when enabling communication between diverse sets of systems. XML's clear representation of structured data makes it an ideal foundation for the Web services framework.

3.1 Interoperability for Information Exchange

The success of the Web is due, in part, to the standardization around a few fixed communications protocols, such as HTTP and FTP. The Web services framework tries to replicate that pattern of success by relying upon a simple and standard XML protocol that in the most common cases will be used over existing Web communication protocols. This would minimize the amount of infrastructure that needs to be deployed, helping achieve pervasive deployment, and assuring Internet-wide application interoperability at the most basic level.

Use of a Web-based XML protocol is by far the best solution, although two other options could be considered. The first is the usage of a pre-existing non-Web based protocol, and the second is the design of a new non-Web based protocol. Most existing protocols are tied too closely to the programming models for which they were originally designed, exposing artifacts which limit their ability to interoperate with alternate environments, and thus are unsuitable for use in a truly heterogeneous system. Those protocols which do provide support for heterogeneous environments generally suffer from a lack of universal deployment, which also makes them undesirable.

Designing a new protocol also has drawbacks. In the unlikely case that this major effort is undertaken, the result would likely require the deployment of an alternate supporting infrastructure, including data representation formats, instead of building upon the functionality which is already in use on the Web. Unless the new protocol offers significant advantages over what is already available, it is also unlikely to lead to universal deployment.

Because of its basis in XML and widespread adoption, the Simple Object Access Protocol (SOAP) is likely to become the *lingua franca* for the Web services framework. SOAP is an XML messaging protocol which, at its core, is extremely simple, providing just a few conventions on how to structure headers and body in an XML message. SOAP is declared to be transport independent, so that SOAP messages can be sent over arbitrary transport protocols. As SOAP is still in its early stages of development, there are many requirements which have not yet been fully addressed, such as the need for integrated security and the support of a unified transaction

model. Despite these unresolved issues, SOAP is probably the most fundamental piece of the Web services framework, as it defines a basic communication channel between Web services and their consumers, and assures a base level of interoperability.

SOAP is in many ways similar to the Object Management Group's CORBA General Inter-ORB Protocol (GIOP) [7]. While they both support protocol independent communication within a heterogeneous environment, CORBA's lack of an XML base and less than universal adoption makes SOAP a better choice for use in the Web services framework.

3.2 A Unified Representation for Services

A key goal of the Web services framework is to provide a common representation of applications which use diverse communication protocols and interaction models, while at the same time enabling Web services to take advantage of more efficient protocols when they are available at both ends of the interaction.

In the Web services framework this goal is achieved by separating abstract application descriptions from protocol bindings. An abstract application description uses XML Schema for describing the data types that the application uses. It also defines the messages exchanged, and the application interfaces as collections of operations exchanging those messages. In this respect, Web services descriptions do not fundamentally differ from those created in other interface description languages (e.g. [7]), except maybe in the focus on messages and in the use of XML Schema as its type definition language. However, a Web service abstract definition is extended by "protocol bindings". Bindings map an abstract interface to a specific communication protocol, providing all the information necessary to effectively access the application, except for the particular endpoint address. A pair consisting of an endpoint address and a protocol binding defines a service endpoint or "port". The description above roughly corresponds to the current Web Services Description Language [3], now a *de facto* industry standard.

The important point in this design is the separation between the functional description of the application and the communication protocol information. A single application can provide multiple protocol bindings, reflecting the fact that the same functionality may be accessed using different communication mechanisms. The application initiating an interaction can choose from the available protocol bindings the one that best serves its needs. For example, if both applications happen to share a private network, one of them may discover that, in addition to the basic HTTP-SOAP access, the other application has a binding for a reliable messaging system (such as IBM's MQSeries), and decide to use that access mechanism for the additional performance and quality of service that it provides.

Providing a common representation is also a central piece of the OMG's Object Management Architecture [7], which uses IDL as its common, platform independent representation language. The central difference lies in the fact that WSDL provides explicit representation of available protocols, while in the OMG model the protocol is fixed at the time the ORB infrastructure is deployed and is not visible to the developer or the application.

3.3 Addressing Outlying Requirements

The two specifications, SOAP and WSDL, described above address the most basic problems underlying the development of the Web services framework. A number of additional specifications, in different stages of development, address other functional aspects of the Web services model. We will only mention two of these aspects: services registries and service composition.

Service registries enable two basic functions in the Web services model: first, they allow application developers to find services and to develop code that relies on those services; second, they enable just-in-time integration of service components. In both cases the registry must be able

to provide a representation of the service's technical characteristics. To enable just-in-time integration, however, the registry must support searching for services based on compatible technical specifications. The Universal Description Discovery and Integration (UDDI) [9] specification defines the operation of a registry supporting this type of service location.

In UDDI, technical specifications like WSDL descriptions can be registered and then used to qualify the registry description of a compliant service. The separation between abstract interfaces and deployment information in WSDL is particularly well suited for this purpose, because it can provide generic specifications which are implemented by large classes of services. Compliant services can be located through a SOAP query to the UDDI registry, and be integrated at run-time by the calling application.

Service composition is the process of assembling service components to create new services. Service composition requires that a well defined component interaction model be developed, and it is probably fair to say that WSDL already provides a good base model. Assembling components to create applications or new components can be done using multiple methods, like scripting languages, standard object-oriented, or component composition languages [8].

Two composition languages for Web services have been proposed, the Web Services Flow Language (WSFL) and XLANG [5, 6]. Both of them use a flow model as their composition mechanism, party because of the desire to address the problem of business process modeling on the Web as a Web services composition problem and the long-standing tradition of using flows to represent business processes. From the component interaction perspective, these languages add a flow interaction model (the interaction between the actions in a flow) to the basic interactions that are mediated by the component's interface.

4 Conclusion

The Web services framework is an attempt to enable systematic application-to-application interactions on the Web, ensuring, at the same time, a smooth integration of existing infrastructure into the model. The main challenge is the wide heterogeneity of the target applications, with respect to the variety of programming models and communication protocols used. The Web services framework defines a two pronged model to deal with the problem: definition of a basic interoperability protocol at the data exchange level which is based upon existing Web standards, and creation of a unified application representation mechanism that makes explicit the multiplicity of available protocols. The Web services framework is currently undergoing widespread adoption and continuing development.

References

[1] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", to appear in International Journal of Supercomputer Applications, 2001.

[2] E. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, D. Winer, "Simple Object Access Protocol (SOAP) 1.1", May 2000. Available at http://www.w3.org/TR/SOAP .

[3] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1", W3C Note, March 2001, Available at http://www.w3.org/TR/wsdl .

[4] C. Szyperski, "Component Software: Beyond Object-Oriented Programming", Addison-Wesley, New York, 1999.

[5] IBM Corporation, "Web Services Flow Language (WSFL 1.0)", May 2001, Available at http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf .

[6] S. Thatte, "XLANG: Web Services for Business Process Design", May 2001, Available at http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm .

[7] Object Management Group, "CORBA 2.4.2 specification", Available at http://www.omg.org .

[8] S. Weerawarana, F. Curbera, M.J. Duftler, D.A. Epstein, J. Kesselman, "Bean Markup Language: A Composition Language for JavaBeans Components," Proc. 6th USENIX Conference on Object–Oriented Technologies and Systems, Jan. 2001.

[9] UDDI Project, "UDDI Technical White Paper", September 2000, Available at http://www.uddi.org .