The Web Services Architecture

The goal of this hour is to explain the core architecture that makes up Web services. We will concentrate on the interactions between core elements of the architecture and how this interaction provides the desired result: the successful sending of a message and receiving of a response. In addition, we will cover why this architecture is important and how it can improve the way that we program computers.

In this hour, you will learn

- The goal of the architecture
- The major software pieces that combine to create Web services
- How the major pieces work together to make Web services work

The Goal of the Web Services Architecture

At the highest level, the goal of Web services is application-to-application communication over the Internet. More specifically, this communication is performed with the idea of facilitating *enterprise application integration (EAI)* and e-commerce, specifically business-to-business e-commerce.

Web services is not the first attempt to solve this problem. In 1989, the *Object Management Group (OMG)* was formed to address this exact problem. The consortium included hundreds of member companies and organizations and produced some good results. They released CORBA 1.0 in 1991, which was used in LAN-based systems.

In 1996, OMG released the CORBA 2.0 specification, which included the *Internet Inter-ORB Protocol (IIOP)*. The IIOP allows multiple *Object Request Brokers (ORBs)* to interoperate regardless of vendor.

In that same year, Microsoft shipped software based on its *Distributed Component Object Model (DCOM)*, which was an enhanced version of its previous component architectures such as *object linking and embedding (OLE)*, *Component Object Model (COM)*, and ActiveX. In many ways, DCOM was a direct competitor of CORBA.

Given that we have had half a dozen years to perfect these technologies, the question is why we would want to use anything else. Web services is essentially a new competitor in this populated area of the computer business.

The reasons that there is still a market for a new solution is proof of dissatisfaction with the previous two approaches.

- Both CORBA and DCOM are built around the idea of making synchronous calls to methods on distributed objects. This approach leaves a gap that messaging software such as IBM's MQSeries and the *Java Message Service (JMS)* were created to fill: handling asynchronous messages. Some messages need to wait on a response, but others just want to hand data to another application.
- Both CORBA and DCOM are weak in the area of data encoding. They provide a mechanism for representing simple data types, but this data was encoded in binary formats. Binary formats cause the application logic to worry about low-level details such as whether the data is represented as big-endian or little-endian format. In addition, binary data cannot be read very easily by a person, which makes debugging harder.
- Arbitrary data is not easy to work with in CORBA or DCOM. Because they were developed before the advent of XML and the cheap computing power that makes XML practical, they send data in pure binary formats. XML makes working with arbitrary data easier.

- Data validation has to be done by the program logic in DCOM and CORBA rather than declaratively with WSDL in Web services. This increases the complexity of the endpoints because the logic to handle validation must be written for each application.
- DCOM and CORBA are two competing and rarely interoperating approaches. The promise of interoperability is a big part of both technologies' reasons for existing in the first place, but the fact that they don't talk to each other is disappointing. We don't get too excited about being able to interconnect with half the world. Interoperability is an all-or-nothing proposition.
- Vendor acrimony between those in the CORBA and DCOM camps created feelings that are not easy to heal. It is politically impossible for either camp to surrender at this point.
- The question of control held back DCOM. The smaller vendor and user community could only petition Microsoft with suggestions for changes to DCOM. This created a lack of enthusiasm for these approaches among technical leaders in the industry.
- Vendor motives are naturally self-centered, as they should be. This causes their approaches to be purposely incompatible with other vendors' approaches. The rest of us have trouble getting excited about an approach that is perceived as serving a vendor's bottom line instead of the industry as a whole.

As you might expect, Web services provides an approach that allows us to avoid many of the problems that CORBA and DCOM suffer from.

- Web services specifications have been designed from the ground up with the idea of application integration being central. This allows Web services to support message-centric asynchronous transactions as well as RPC-style synchronous transactions with equal ease.
- Data encoding is done using XML schema data types and aggregations of those types. Each endpoint in the transaction is responsible for deciding how to represent that data in its code for the duration of the application's execution. The data is reformatted into XML for the next step or the return trip.
- XML documents are human and machine readable. This makes debugging easier because each message that is transmitted over the wire can be intercepted and analyzed to discover details about the nature of the problem.
- XML is designed to represent arbitrary data. The XML schemas for complex data can be used by the WSDL in its description.
- Data validation can be added to any XML document by adding a line of text to the top of the file. This removes the need to create custom data type validation code in each application endpoint.

- In theory, Web services are interoperable. In reality, they are pretty interoperable, but their architecture is designed in such a way that this interoperability will increase over time.
- A surprising amount of cooperation between all vendors is the trademark of Web services. Because Web services are defined abstractly, for the most part, this allows them to be implemented using almost any software tool that you can think of.
- The question of control over the direction of Web services is easily answered. You and I control it. Certainly organizations, such as OASIS and W3C, provide direction to the specifications, but we are free to join those efforts at whatever level of interest that we have in them. Any company could join the W3C (for a fee of $15,000 U.S.), assign a few high-quality software engineers to a committee within those organizations, and exert considerable influence over the direction that the specification takes.
- Vendors are naturally self-centered, but when they cooperate in creating a specification, they tend to keep each other in line. As a result, the rest of us tend to believe that the specifications coming out of the W3C, OASIS, and other similar organizations are honest attempts to find the best solutions to problems that we all face.

The SOA

The *service-oriented architecture (SOA)* provides the theoretical model for all Web services. It is a simple model that contains three entities and three operations. Figure 6.1 shows this model.

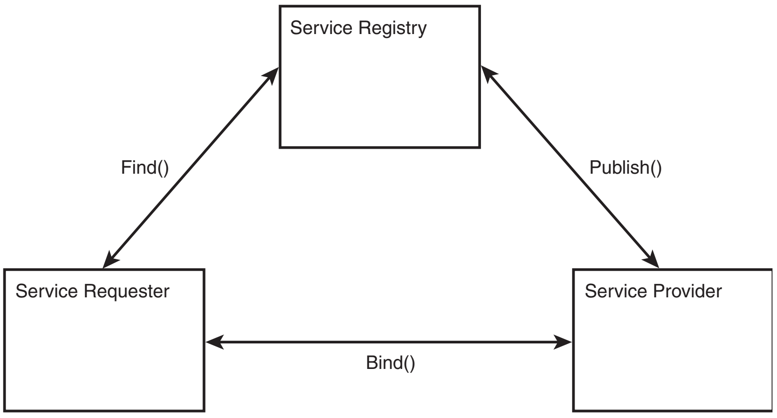


FIGURE 6.1
The SOA provides a useful model for understanding Web services.

New Term The *Service Requestor* is an application that wants to receive a service from another application. It doesn't know where this other application is or how to locate it, so it turns to the Service Registry and requests a `Find()` operation.

New Term The *Service Registry* is a well-known application that returns information about Web services in response to search criteria that has been submitted by a Service Requestor in the `Find()` operation. This information returns the contact information for Web services based on their classifications that match the search criteria. It also includes information about how to find out the connection details.

New Term The *Service Providers* `Publish()` these classification details as well as the connection details to the Service Registry. They do this for the same reason that a business would purchase a listing in the Yellow Pages of the local phone book.

The Service Requestor uses the connection details to `Bind()` to the Service Provider. Once bound, they can send messages and receive responses.

The Major Components of the Architecture

Web services are built on a foundation of different, but cooperating, specifications and standards. Some of the reasons for this are historical, but the primary reason that there are multiple standards is that no one person or group has a monopoly on the specification process. Everyone in the Web services community has a voice in determining the direction that the technology will take. At the present time, the following specifications are considered the primary software specifications that, when taken as a whole, compose what we know as Web services:

- HTTP/1.1
- RFC 2965:HTTP State Management Mechanism (cookies)
- SOAP 1.1
- UDDI version 2.04 API and the other UDDI 2.03 lesser specifications
- WSDL 1.1
- XML 1.0 (second edition)
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes

New Term The source of this list is the *Web Services Interoperability Organization (WS-I)*, but the contents are not controversial to our knowledge.

SOAP

New Term *SOAP* used to stand for Simple Object Access Protocol, but now SOAP is considered a name and not an acronym. As we stated earlier in the hour, SOAP is a message format that enables method calls to be sent in an XML format from one computer to another. In addition, it can send an entire XML document (that contains only data) instead of a method call if you prefer.

New Term The word SOAP was coined in 1998 during discussions between Microsoft, DevelopMentor, and Userland about how to best send remote procedure calls over HTTP using XML. After the usual stalls that accompany this type of discussion, Userland published a draft of the specification in the summer of 1998. After performing some updates to respond to feedback, SOAP 0.9 was submitted to the *Internet Engineering Task Force (IETC)* in September 1999. With a few changes, SOAP 1.0 was announced in December 1999.

New Term IBM joined with the other companies in submitting a specification for SOAP 1.1 to the *World Wide Web Consortium (W3C*) in May 2000. This was significant because IBM is closely identified with Java and J2EE. Their joining the submission created the correct perception that this was more that a single-vendor Microsoft submission.

New Term SOAP defines an *envelope* that contains a *header* and a *body*. The header provides instructions on how the message is to be processed. Sometimes this is information that the message is to be forwarded on or that the return path is to be a certain URL. Many of the draft specifications that are now circulating propose enhancements to this header. The SOAP body contains the *payload*. This payload is the method call from the originator and it is the response from the remote system.

SOAP 1.1 is commonly used today, but SOAP 1.2 is, at this writing, a candidate for release as a recommendation (the W3C equivalent of a standard).

Extensible Markup Language

Extensible Markup Language (XML) has become the common language of the computing business. It is a tag-oriented language that looks superficially like HTML, but its purpose is different. HTML describes the way a document should look when displayed in a browser. XML, on the other hand, describes what the data that you are looking at means, independent of the way that it is displayed. Hour 7, "Understanding XML," covers XML in some detail.

Briefly, XML was created to remove the ambiguity from data. Suppose that you create a string containing the following:

```
7 3 2003
```

If you send this string to my computer program, how is the program supposed to interpret it? Should we send you seven cases of books with a serial number of three and charge you $2,003 for it? If we add the following tags to the string, we can completely clear up the ambiguity:

```
<date>
   <day>3</day>
   <year>2003</year>
   <month>7</month>
</date>
```

The meaning is absolutely clear now. You are sending me a date that means “July 3, 2003.” Adding these XML style tags to the data makes it easy to parse and use.

An additional feature of XML is the existence of a description of what the tag structure must look like in a document for it to be valid. The original description was called the *Document Type Description (DTD)*, but it is fast being made obsolete by the XML schema. They both serve the same purpose, but the XML schema is more powerful and allows for more precise descriptions.

The XML specification is also maintained by the W3C. This organization accepts requests for changes and enhancements to this specification and occasionally publishes a new version. The version of XML that is in common use at this writing is XML 1.0 (second edition).

Hypertext Transport Protocol

Hypertext Transport Protocol (HTTP) is the workhorse of the Web. It is managed by the W3C as well. Its current version is 1.1, and all activity has ceased on it because it is considered complete and stable.

The purpose of HTTP is to provide a protocol to move requests and responses between clients and servers. It will carry any information that is placed in it from point A to point B without regard for its data type. As a result, it is a popular way to transport SOAP messages between clients and Web services.

HTTP is not required in SOAP 1.2 because other protocols have been added. The vast majority of Web service messages will be carried by HTTP in the near term, however.

HTTP State Management Mechanism (Cookies)

Also known as RFC 2965, this document specifies how to create stateful sessions using HTTP requests and responses. It adds three headers to HTTP, which carry state information between participating clients and servers.

Web Services Description Language

Web Services Description Language (WSDL) is a specification that tells us how to describe a Web service. Consider the case in which you are told about a Web service that provides some information or processing that you would like to access. Your first questions would be

- What method calls does it accept?
- What parameters will it accept?
- What responses will it send back?
- What protocols can it process?
- What data format specifications can it accept?
- What are the URLs for the service?

A WSDL document is an XML document that contains all the information that you need to contact a service. In addition to being verbose, it is platform and language neutral. A programmer or program is able to read this document and create an unambiguous message that can call a method or methods in this service.

This is a marvelous achievement because now you have a specification that describes a way to describe any piece of software, including non–Web service software, in a precise way. This means that you can write software that can generate messages based on the logic in your program combined with the information in the WSDL.

Normally, a potential Web service consumer would obtain the WSDL first. Using the WSDL, this would-be client could either have a programmer create software to this WSDL or use software that is capable of generating a program to do the communications part of the client processing.

WSDL is also managed by the W3C. It is currently at version 1.1, but a version 1.2 is in draft status. In addition to the minor changes that you would expect, version 1.2 is attempting to add an abstract model to the specification. At this point, a consensus has not been reached on what that abstract model should look like.

The Structures and Data Types of XML Schema

The structures part of the XML schema specifies the schema definition language that can be used to describe the structure of XML documents. It can also be used to constrain the contents of a document.

The datatypes part of the XML schema specification defines a way for specifying datatypes in XML schemas and other XML specifications. The facilities described here are more powerful than those available in the old DTDs.

The XML schema specification is managed by the W3C also, where it has the status of a recommendation (a W3C standard).

Universal Description, Discovery, and Integration

NEW TERM The *Universal Description, Discovery, and Integration (UDDI)* specification describes a special type of registry that lists Web services that you might potentially be interested in. It contains quite a bit of information oriented toward allowing you to search its contents for a specific characteristic or feature. It uses special classification schemes called taxonomies that categorize a Web service in ways that are meaningful to potential clients.

The registries can be of various types:

- **Public**—A public registry is one that is open to the public for searching. Several major companies maintain public registries, including IBM and Microsoft. All the entries in the public registries are replicated in the other public registries so that a search performed against one registry will be able to access data about every publicly registered Web service.
- **Private**—A private registry is one that exists behind the firewall of one company. The purpose of this registry might be to provide a way to search for internal Web services. It might also contain entries to other software systems in the company that are not exposed yet as Web services. In this scenario, the UDDI registry acts as a software reuse catalog.
- **Restricted**—A restricted registry can only be accessed by certain organizations who have been granted permission to access it. Trading partners can use this information about each other's systems to find out how to interact with them better.

The industry has been slow to accept public registries because of security concerns and fear of getting out on the bleeding edge. In addition, UDDI represents a new way of doing business, and businesses are always very conservative about making drastic changes.

The current version of UDDI that is in popular use is version 2.04. A version 3 has been published, but it take some time for all the software vendors to bring their offerings up to the new release. Unlike many of the other Web services specifications, UDDI is not managed by the W3C. Instead, it is managed by a group called the *Organization for the Advancement of Structured Information Standards (OASIS)*.

Understanding Interactions Between Components

Sometimes it is hard to visualize the interaction between parts of a complex transaction. For this reason, we are going to walk through a scenario to show how software written to each of these standards interacts with the others in a typical transaction.

Our hypothetical Web Service (Service Provider) begins its life as a COBOL program that accepts a file of addresses. It compares the addresses against the official USPS address database, corrects the address (if needed) and adds the last four digits of the ZIP Code (if missing).

NEW TERM The engineer in charge of creating the Web service uses Apache Axis as his SOAP engine. He creates a special piece of Java software that accepts SOAP remote procedure calls as input and makes calls to the legacy Java system. In Axis jargon, this is a special type of *handler* called a *dispatcher*.

He decides to write the WSDL by hand to gain experience. This is not too hard because the service is very simple. He describes the method calls that his Web service will accept by following a combination of the WSDL and schema specifications.

He next publishes information about the Service Provider to the Service Registry of his choice, the Microsoft public registry (which is written to the UDDI specification). There he enters in data about the Web site. Part of the information that our engineer places on the site is the URL of the WSDL for the Web service.

A potential customer performs a find operation on the Service Registry of his choice, the IBM public registry. This registry replicates the entries made on the Microsoft registry every night, so our Web service is listed in both registries. This replication is done according to the UDDI specification.

The potential customer finds the entry for our service and uses the listed URL to download a copy of the WSDL. Using the WSDL, he writes a Visual Basic program to serve as the Service Requestor to access the service. When the client-side programming is complete, he tests it by requesting that it perform a bind operation on the Web Service. After the bind operation is successful, the client passes in a version of his home address that purposely contains a misspelling and a missing ZIP Code and waits for a response.

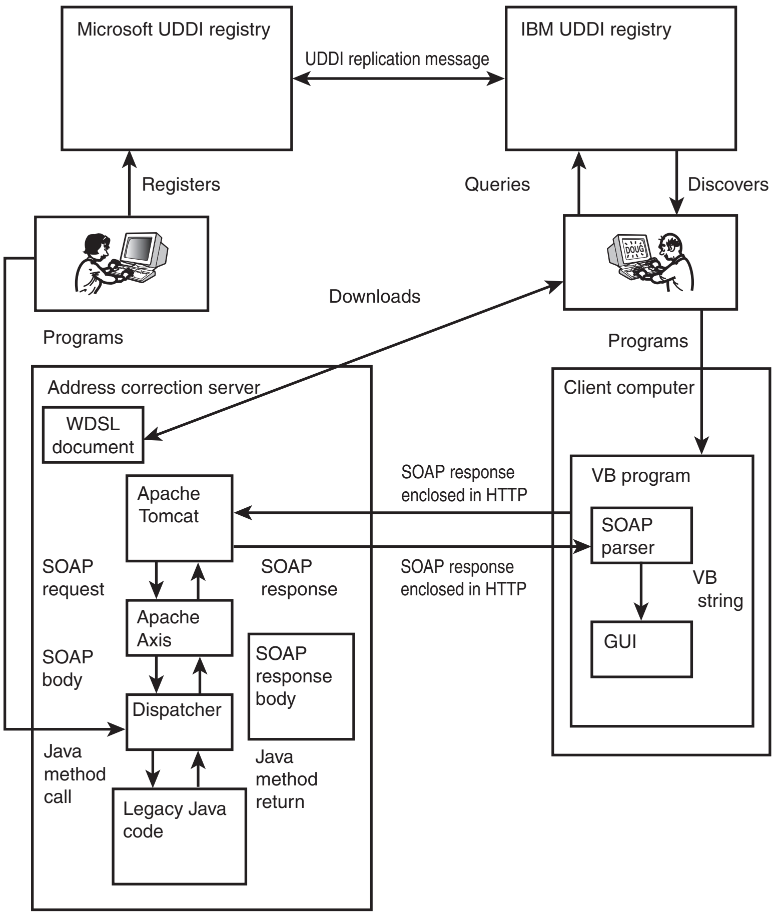
The client software packages his request into a SOAP envelope and sends it to the URL for the site using the HTTP. The Tomcat Web server receives the HTTP message and strips off the HTTP headers. It passes the SOAP message to the Axis SOAP engine. The SOAP engine removes the header portion and processes any directives that might appear in the header.

The Axis SOAP engine calls the dispatcher program. The dispatcher calls the methods in the legacy system and passes the incorrect address to it as a parameter. The legacy address correction system fixes the address and adds the correct ZIP Code. It passes back the corrected address to the dispatcher. The SOAP engine creates the SOAP response message, adds any SOAP headers that are needed, and returns it to the Tomcat Web server. Tomcat adds the HTTP-specific data to the response and sends it back to the client.

The client program converts the SOAP message into a Visual Basic data type and returns it to the Visual Basic program. This program stores the response in a text box and displays it on the screen. The Visual Basic programmer looks at the corrected address as proof that everything worked properly. Figure 6.2 shows this process graphically.

In essence, every Web service transaction follows this same general approach with the exception of the find, which is either done the first time or omitted altogether. Sometimes the message is forwarded on to the next Web service in a chain, but even then, the basic transactions are just being chained together. In the future, the chains will become more complex as new specifications become accepted, but the basic transaction is the same.

FIGURE 6.2
All the software written to the different specifications work together to produce a successful transaction.



Summary

In this hour, we looked at the Web services architecture from three angles. First, we examined the weaknesses of the DCOM and CORBA approaches and how Web services address these areas in their own architecture.

Second, we looked at the Service-Oriented Architecture (SOA) and how its components, the Service Requestor, Service Provider, and Service Registry interact. Third, we looked at the basic specifications that form the foundation of Web services. Finally, we talked through a simple example in which the SOA model and the standards were discussed in the context of the interaction between actual software components written to these specifications.

Q&A

Q What is the goal of the Web services architecture?

A The goal is to facilitate the transfer of information between computers of differing manufacture and operating system type.

Q How do the goals of Web services differ from those of CORBA and DCOM?

A The goals are virtually identical, but the details of the implementation differ in nearly every aspect.

Q Why didn't everyone just agree to use CORBA or DCOM?

A The computing world broke into two camps who couldn't find any compromise. The two systems were very difficult to make interoperable, and neither side was interested in changing.

Q Why did they agree on the Web services architecture?

A They agreed for many reasons, but the primary one might have been "battle fatigue." Both sides were tired of the contention and wanted to find something that worked.

Workshop

The Workshop is designed to help you review what you've learned and begin learning how to put your knowledge into practice.

Quiz

1. Where did CORBA and DCOM come from?
2. What are the components of the SOA model?
3. What are the methods of the SOA model?
4. What are the names of each component as they are commonly used in Web services?

Quiz Answers

1. CORBA was created by the Object Management Group (OMG). DCOM was created by Microsoft.
2. The core components are the Service Registry, Service Requestor, and Service Provider.
3. The core methods or operations are `Find()`, `Bind()`, and `Publish()`.

4. The Service Registry is commonly called the UDDR Registry or the Public Registry. The Service Requestor is often called the client or service consumer. The Service Provider is normally just called the Web service, but it can be called the provider also.

Activities

1. Name the weaknesses of the CORBA and DCOM approaches.
2. Name the specifications that are considered the basic Web services specifications, along with their version number.