Introducing Web Services

Like the effect of rail transportation on national economic systems, Web services are fundamentally changing the rules of Web commerce. They connect programs to each other across distant points on the global map, transporting large amounts of data more efficiently and cheaply than ever before. The result is faster, better, and more productive communication for businesses and consumers alike.

The Web started out supporting human interactions with textual data and graphics. People use the Internet daily to look up stock quotes, buy consumer goods, and read the latest news. This level of interaction is fine for many purposes. But the essentially text-based Web does not support software interactions very well, especially transfers of large amounts of data. A more efficient method is needed that allows applications to interact directly with one another, automatically executing instructions that would otherwise have to be entered manually through a browser.
2 Introducing Web Services

The current Web does not support software-oriented interactions very well.

Individuals and companies doing business over the Web need a way to publish links to their applications and data, in much the same way that they publish links to their Web pages. Internet-based applications need to be able to find, access, and automatically interact with other Internet-based applications. Web services improve Internet use by enabling program-to-program communication. Through the widespread adoption of Web services, applications at various Internet locations can be directly integrated and interconnected as if they were part of a single, large IT (information technology) system.

The Basics of Web Services

Web services are Extensible Markup Language (XML) applications mapped to programs, objects, or databases or to comprehensive business functions. Using an XML document created in the form of a message, a program sends a request to a Web service across the network, and, optionally, receives a reply, also in the form of an XML document. Web services standards define the format of the message, specify the interface to which a message is sent, describe conventions for mapping the contents of the message into and out of the programs implementing the service, and define mechanisms to publish and to discover Web services interfaces.

This technology can be used in many ways. Web services can run on desktop and handheld clients to access such Internet applications as reservations systems and order-tracking systems. Web services can also be used for business-to-business (B2B) integration, connecting applications run by various organizations in the same supply chain. Web services can also solve the broader problem of enterprise application integration (EAI), connecting multiple applications from a single organization to multiple other applications both inside and outside the firewall. In all these cases, the technologies of...
Web services provide the standard glue connecting diverse pieces of software.

As illustrated in Figure 1-1, Web services wrap, presenting to the network a standard way of interfacing with back-end software systems, such as database management systems, .NET, J2EE (Java2 Platform, Enterprise Edition), or CORBA (common object request broker architecture), objects, adapters to enterprise resource planning (ERP) packages, integration brokers, and others. Web services interfaces receive a standard XML message from the networking environment, transform the XML data into a format understood by a particular back-end software system, and, optionally, return a reply message. The underlying software implementations of Web services can be created by using any programming language, operating system, or middleware system.

![Figure 1-1  Web services interface with back-end systems.](image-url)
Web services combine the execution characteristics of programmatic applications with the abstraction characteristics of the Internet. Today's Internet technologies succeed in part because they are defined at a sufficiently high level of abstraction to enable compatibility with any operating system, hardware, or software. The Web services-based Internet infrastructure exploits this abstraction level and includes semantic information associated with data. That is, Web services define not only the data but also how to process the data and map it into and out of underlying software applications.

**A Simple Example: Searching for Information**

Today, most services are invoked over the Web by inputting data into HyperText Markup Language (HTML) forms and sending the data to the service, embedded within a uniform resource locator (URL) string:

```
http://www.google.com/search?q=Skate+boots&btnG=Google+Search
```

This example illustrates how simple Web interactions, such as a search, a stock purchase, or a request for driving directions, are accessed over the Web by embedding parameters and keywords in a URL. In this example, entering a simple search request for *Skate boots* into the Google search engine results in the URL shown. The `search` keyword represents the service being requested over the Web, whereas the `Skate+boots` keywords represent the search string entered into the HTML form displayed by the Google Web site. The Google search service then passes the request to a series of other search engines, which return lists of URLs to pages with text matching the search keywords `Skate+boots`. This inefficient way of searching the Web depends entirely on matching the given text strings to cataloged HTML pages.
XML provides a great many advantages for transmitting data across the Internet. Now the preceding request can be contained in an XML document instead:

```xml
<SOAP-ENV:Body>
<s:SearchRequest xmlns:s="www.xmlbus.com/SearchService">
  <p1>Skate</p1>
  <p2>boots</p2>
  <p3>size 7.5</p3>
</s:SearchRequest>
</SOAP-ENV:Body>
```

Sending the request within an XML document has many advantages, such as improved data typing and structure, greater flexibility, and extensibility. XML can represent structured and typed data—the size field can be typed as a decimal string or as a floating point, for example—and can contain a larger amount of information than is possible within a URL string.

This example is shown in the form of a Simple Object Access Protocol (SOAP) message, a standard form of XML messaging and one of the major enabling technologies in the Web services foundation (see Chapter 4). In SOAP messages, the name of the service request and the input parameters take the form of XML elements. The example also illustrates the use of XML namespaces (xmlns:), another critical element of Web services (see Chapter 2). Because XML documents support data typing, complex structures, and the association of XML schemas, modern Web services technology provides significant advantages over existing URL and HTML capabilities for accessing software applications.

The Next Generation of the Web

Web services are aimed at putting the vast global network of the Web, established for human interaction, to an entirely new use: XML documents are a better way to send data.
Introducing Web Services

purpose. Software-oriented interactions will automatically perform operations that previously required manual intervention, such as:

- Searching for and buying goods and services at the best price
- Coordinating travel tickets and restaurant tables for a given date
- Streamlining business procurement, invoicing, and shipping operations

The next generation of the Web will use software-oriented services to interoperate directly with applications built using any combination of objects, programs, and databases.

But Web services are not only about interfaces to objects, programs, middleware, and databases for access over the Internet. By combining a series of Web services into a larger interaction, Web services also provide the means to perform new types of interactions.

Suppose, for instance, that you live in San Francisco and wish to reserve a table at your favorite Paris restaurant and then make the necessary travel arrangements to be there at the agreed time. Today, you would have to call the restaurant directly to get the reservation, taking into account the 9-hour time difference and the language difference, and then call a travel agent to find a compatible flight and a hotel. But using Web services, you could schedule the dinner with your personal digital assistant (PDA) calendar and click on a button to automatically reserve a table at a convenient time. Once the reservation was made, the Web service could kick off other services that would book a cheap flight and reserve a room at a nearby four-star hotel.

Figure 1-2 shows how Web services can interact with a PDA connected to a wireless Web services processor to book a reser-
The Web services processor accepts requests from the calendar function of the PDA and discovers Web services related to extended calendar functions, such as reserving a restaurant table. After successfully reserving a table, the Web services processor contacts Web services for hotel and flight reservations to complete the requested scheduling action.

Web services are also very useful for discovering and interacting with Internet sites that provide online order entry systems, such as the one for the Skateboots Company’s trendy skateboots—a boot with a retractable ice skate built in—like the ones that Batman and Robin used in the movie *Batman and Robin*. Sporting goods retailers interested in stocking the boots, this

1. Throughout the book, the diamond-on-a-stick convention is used to represent a Web services interface.
year’s hot new item, can use Web services to place advance bulk orders in batch, to check the status of an order, or to place in-season restocking orders and be immediately notified of back orders, if the manufacturer is out of stock. Web services building blocks provide standard components of the application for the Skateboots Company, which isn’t large enough to host its own entire application infrastructure. Web services hosting companies provide security services to ensure that Skateboots accepts orders only from approved retailers and to provide credit validation services for approving bulk advance orders. Still other companies help Skateboots by providing electronic funds collection and accounting services.

The entire Skateboots order entry system is exposed to the Internet as a Web service, but behind the top-level Web service are a number of other Web services working together to provide the necessary functionality. Figure 1-3 illustrates how Web services can change the way business applications are built and used. The retailer interested in stocking skateboots inputs a request to its local inventory management service, which is exposed to the shop computers as a Web service. The local inventory service then contacts the manufacturer’s Web service over the Internet and sends the order for the correct number of skateboots, based on available shelf space and the most popular sizes.

The Skateboots Company’s order entry system comprises multiple Web services, including a custom-built part that deals with the unique aspects of its product and several commodity parts that take care of standard functions, such as authenticating the user, credit authorization, and accounting and billing, all hosted by other companies that specialize in providing such services over the Internet.

Creating business applications using Web services entails putting into proper relationship a number of other Web services, which can be implemented by using any combination of programming
language, operating system, or packaged software, inside or outside the firewall. (This is also the way in which Web services solve the difficult EAI problem.) In establishing the proper relationship, or flow, of related Web services, it also automates the corresponding business processes and procedures.

Through the widespread adoption of Web services, the Internet is becoming more efficient, especially for business interactions. In the next generation of the Web, Web services building blocks will enable automatic Internet interactions, combining direct access to software applications and business documents, bypassing familiar text-based Web pages to access software-based data directly. Furthermore, fundamental Web services building blocks are very likely to be hosted and published by a variety of companies focusing on a specific functional component, such as authentication, transactional coordination, or

Web services create greater commercial efficiencies
Introducing Web Services

The level of abstraction at which Web services operate encompasses such interaction styles as RPC (remote procedure call) emulation, asynchronous messaging, one-way messaging, broadcast, and publish/subscribe. Most major database man-

### Toward a Common Understanding

Web services technology exists at a sufficiently high level of abstraction to support multiple simultaneous definitions, which are sometimes contradictory. At the simplest level, Web services can be thought of as Internet-oriented, text-based integration adapters. Any data can be mapped into and out of ASCII text, and this type of mapping has long been the lowest common denominator for graphical display systems and database management systems. If all else fails, the saying goes, map the data to text. Text-based systems also are behind the success of the World Wide Web, on which the additional abstraction of Web services is based. Any computer or operating system is capable of supporting HTML and Web servers, and browsers; and when they download files, they don’t care or even know what type of back-end systems they’re interacting with.

The same is true for Web services, which often leads to a lot of confusion when developers of traditional, or established, computing environments try to understand Web services technology in reference to a single type of distributed software system, such as CORBA, J2EE, or .NET. Because Web services are much more abstract—more like adapters than they are like interfaces—it will be some time before the industry settles on truly common definitions and conventions for them.

accounting and billing. This change to direct application-to-application interaction over the Web lies at the heart of Web services, what they mean, and how they work.

### Interacting with Web Services

The level of abstraction at which Web services operate encompasses such interaction styles as RPC (remote procedure call) emulation, asynchronous messaging, one-way messaging, broadcast, and publish/subscribe. Most major database man-
Interacting with Web Services

Web services encompass RPC and document-oriented interactions. RPC-oriented interactions are good for brief data exchanges.

RPC-Oriented Interactions

In RPC-oriented interactions, the Web services request takes the form of a method or a procedure call with associated input and output parameters. In contrast to the document-oriented interaction, the RPC-oriented interaction sends a document formatted specifically to be mapped to a single logical program or database, as shown in Figure 1-4. Because the “real-time” or in-season order for skateboots depends on available inventory, for example, the program accesses the database to

---

2. A single logical program can, of course, comprise multiple subprograms.
check the available supply of the ordered item. If everything is OK, the program returns an XML document to the distributor in the request/response format to indicate that the order has been accepted and will be shipped. If supply isn’t available, the return message indicates a back order or rejects the order entirely. In contrast to the document-oriented interaction style, the request and the reply are modeled as synchronous messages. That is, the application sending the message waits for a response.

**Document-Oriented Interactions**

In the document-oriented interaction style, the Web service request takes the form of a complete XML document that is intended to be processed whole. For example, a Web service that submits a complete purchase order, such as a preseason order for skateboots, would submit the entire bulk order to the manufacturer at once, as shown in Figure 1-5. This is like

*The document-oriented style is good for bulk data exchanges*
submitting a message to a queue for asynchronous processing. The manufacturer would typically send an e-mail or other form of acknowledgment to the retailer to indicate that the order was received and would be processed according to a predefined flow of execution. The flow might include such steps as checking the database for previous orders from the same retailer to ensure that it is not exceeding its credit limit or agreed capacity or scheduling a ship date for the order. In a real process flow, of course, many more steps are likely before the order is shipped and the invoice sent out, but the example shows only the final step: sending the XML invoice to the distributor for payment after the order has been shipped and received.
Introducing Web Services

Document-oriented interactions often assume that the parties to a Web services conversation have agreed to share a common business document, such as a purchase order, a ship bill, or an invoice. These parties are often identified as trading partners, or collaborating partners. Trading partners also typically agree on a common process flow, or interaction pattern, for exchanging the shared document, such as requiring an acknowledgment on receipt of a purchase order, returning specific status information in reply to an order query, or sending an e-mail alert when an order has been shipped. During the execution of the business process, a complete document might be exchanged. If the document is already held in common, fragments of information required to fill in specific sections of the shared document, such as purchase price or promised delivery date, might be exchanged.

In the Skateboots Company example, preseason bulk orders are handled by using purchase orders submitted in batches according to predefined terms and conditions that help the manufacturer plan capacity. During the season, immediate restocking orders are handled by more interactive services that depend on filling orders from available inventory and that can immediately identify back orders. Thus Skateboots.com provides Web services supporting both major types of interaction.

The Technology of Web Services

Programs that interact with one another over the Web must be able to find one another, discover information allowing them to interconnect, figure out what the expected interaction patterns are—a simple request/reply or more complicated process flow?—and negotiate such qualities of service as security, reliable messaging, and transactional composition. Some of these qualities of service are covered in existing technologies and proposed standards, but others are not. In general, the Web services community is working to meet all these requirements, but
Web services are sometimes portrayed as “silver-bullet” solutions to contemporary computing problems, filling the role previously played by the original Web, relational databases, fourth-generation languages, and artificial intelligence. Unfortunately, Web services by themselves can’t solve much. Web services are a new layer—another way of doing things—but are not a fundamental change that replaces the need for existing computing infrastructure. This new layer of technology performs a new function—a new way of working—but, most importantly, provides an integration mechanism defined at a higher level of abstraction.

Web services are important because they are capable of bridging technology domains, not because they replace any existing technology. You could say that newer languages, such as Visual Basic, C#, C/C++ and Java—replace older languages, such as COBOL and FORTRAN, although a lot of programs in those languages are still around, as are Web-services mappings for them. Web services, like Web servers, are complementary to, not in conflict with, existing applications, programs, and databases. Application development continues to require Java, VB, and C#. All that’s new is a way of transforming data in and out of programs and applications, using standard XML data formats and protocols to reach a new level of interoperability and integration.

Developers may have to take Web services into account when designing and developing new programs and databases, but those programs and databases will still be required behind Web services wrappers. Web services are not executable things in and of themselves; they rely on executable programs written using programming languages and scripts. Web services define a powerful layer of abstraction that can be used to accomplish program-to-program interaction, using existing Web infrastructure, but they are nothing without a supporting infrastructure.

it’s an evolutionary process, much like the Web itself has been. Web services infrastructure and standards are being designed and developed from the ground up to be extensible, such as XML and HTML before them, so that whatever is introduced in
Web services require the use of several related XML-based technologies to transport and to transform data into and out of programs and databases.

- **XML (Extensible Markup Language)**, the basic foundation on which Web services are built provides a language for defining data and how to process it. XML represents a family of related specifications published and maintained by the World Wide Web Consortium (W3C) and others.

- **WSDL (Web Services Description Language)**, an XML-based technology, defines Web services interfaces, data and message types, interaction patterns, and protocol mappings.

- **SOAP (Simple Object Access Protocol)**, a collection of XML-based technologies, defines an envelope for Web services communication—mappable to HTTP and other transports—and provides a serialization format for transmitting XML documents over a network and a convention for representing RPC interactions.

- **UDDI (Universal Description, Discovery, and Integration)**, a Web services registry and discovery mechanism, is used for storing and categorizing business information and for retrieving pointers to Web services interfaces.

### Usage Example

The basic Web services standards are used together. Once the WSDL is obtained from the UDDI or other location, a SOAP message is generated for transmission to the remote site.

As shown in Figure 1-6, a program submitting a document to a Web service address uses an XML schema of a specific type,
such as WSDL, to transform data from its input source—a structured file in this example—and to produce an XML document instance in the format consistent with what the target Web service expects, as described in the same WSDL file. The WSDL file is used to define both the input and the output data transformations.

The sending computer’s SOAP processor transforms the data from its native format into the predefined XML schema data types contained in the WSDL file for text, floating point, and others, using mapping tables. The mapping tables associate

Figure 1-6  Web services use XML documents and transform them into and out of programs.
native data types with corresponding XML schema data types. (Standard mappings are widely available for Java, Visual Basic, CORBA, and other commonly used type systems. Many XML mapping tools are available for defining custom or special mappings.) The receiving computer’s SOAP processor performs the transformation in reverse, mapping from the XML schema data types to the corresponding native data types.

The URL, in widespread use on the Web, points to a TCP (Transmission Control Protocol) address containing a Web resource. Web services schemas are a form of Web resource, contained in files accessible over the Internet and exposed to the Web using the same mechanism as for downloading HTML files. The major difference between HTML file downloading and accessing Web services resources is that Web services use XML rather than HTML documents and rely on associated technologies, such as schemas, transformation, and validation, to support remote communication between applications. But the way in which Web services schemas are published and downloaded is the same: an HTTP operation on a given URL.

When it receives a document, a Web service implementation must first parse the XML message and validate the data, perform any relevant quality-of-service checking, such as enforcing security policies or trading-partner agreements, and execute any business process flow associated with the document. The Web service at the fictional skateboots.com Web site is located in the skateboots.com/order folder, which is what the URL points to.³

³. A more generic term, the Uniform Resource Indicator (URI), often appears in Web services specifications in place of the URL. A URI is a categorical syntax term that includes the Uniform Resource Locator (URL) and the Uniform Resource Name (URN). A URN is a name that does not reference a physical resource. In practice, there is little difference between URI and URL.
The Web services available at this Internet address are identified within a public WDSL file that can be downloaded to the sending computer and used to generate the message. The Skateboots Company also posted a listing in the public UDDI directory, pointing to the same WSDL file, for customers who might discover the company through the UDDI service. In general, anyone wishing to interact with the Web services that place or track orders for the Skateboots Company over the Web must find a way to obtain and to use that particular WSDL file to generate the message.

Programs at the skateboots.com address provide an HTTP listener associated with the Web services in order to recognize the XML messages sent in the defined format. The programs include XML parsers and transformers and map the data in the SOAP message into the formats required by the Skateboots Company order entry system.

These technologies are enough to build, deploy, and publish basic Web services. In fact, even basic SOAP is enough. Other technologies are continually being added to the expanding Web services framework as they emerge. These fundamental technologies are enough to support use of the Internet for basic business communication and to bridge disparate IT domains, however; and this form of Web interaction is being adopted very quickly.

Over time, as standards for registry, discovery, and quality of service mature, the vision of an ad hoc, dynamic business Web will start to take hold, and Web services will begin to operate more like the current Web, allowing companies to find and to trade with one another purely by using Internet-style communications. In the meantime, the basic Web services technologies and standards covered in this book are sufficient for many solutions, such as integrating disparate software domains—J2EE and .NET, for example—connecting to packaged applications, such
as SAP and PeopleSoft, and submitting documents to predefined business process flows.

**XML: The Foundation**

In the context of Web services, XML is used not only as the message format but also as the way in which the services are used for multiple purposes.

---

### Reinventing the Wheel

Some people say that Web services are reinventing the wheel because they share many characteristics with other distributed computing architectures, such as CORBA or DCOM. Web services do share considerable common ground with these and other distributed computing architectures and implementations, but there's also a good reason for inventing a new architecture. The Web is established, and to take advantage of this tremendous global network, the concepts of distributed computing need to be adapted. First, the Web is basically disconnected; that is, connections are transient and temporary. Distributed computing services, such as security and transactions, traditionally depend on a transport-level connection and have to be redesigned to provide equivalent functionality for the disconnected Web. Second, the Web assumes that parties can connect without prior knowledge of one another, by following URL links and observing a few basic rules. For Web services, this means that any client can access Web services published by anyone else, as long as the information about the service—the schema—is available and understandable and XML processors are capable of generating messages conforming to the schema.

Traditional distributed computing technologies assume a much more tightly coupled relationship between client and server and therefore cannot inherently take advantage of the existing World Wide Web. Because Web services adopt the publishing model of the Web, it's possible to wrap and to publish a specific end point, or business operation, using a Web services interface definition, without requiring a specific type of client for that end point. The paradigm shift that clients can develop and integrate later has many advantages in solving the problem of enterprise integration.
defined. Therefore, it is important to know a little bit about XML itself, especially within the context of how it is used to define and to implement Web services.

**Purposes of XML**

XML was developed to overcome limitations of HTML, especially to better support dynamic content creation and management. HTML is fine for defining and maintaining static content, but as the Web evolves toward a software-enabled platform, in which data has associated meaning, content needs to be generated and digested dynamically. Using XML, you can define any number of elements that associate meaning with data; that is, you describe the data and what to do with it by using one or more elements created for the purpose. For example:

```xml
<Company>
  <CompanyName region="US">
    Skateboots Manufacturing
  </CompanyName>
  <address>
    <line>
      200 High Street
    </line>
    <line>
      Springfield, MA 55555
    </line>
    <Country>
      USA
    </Country>
  </address>
  <phone>
    +1 781 555 5000
  </phone>
</Company>
```

In this example, XML allows you to define not only elements that describe the data but also structures that group related data. It’s easy to imagine a search for elements that match certain criteria, such as `<Country>` and `<phone>` for a given company, or for all `<Company>` elements and to return a list of those entities identifying themselves as companies on the Web.
Furthermore, as mentioned earlier, XML allows associated schemas to validate the data separately and to describe other attributes and qualities of the data, something completely impossible using HTML.

Of course, significant problems result from the great flexibility of XML. Because XML allows you to define your own elements, it's very difficult to ensure that everyone uses the same elements in the same way to mean the same thing. That's where the need for mutually agreed on, consistent content models comes in.

Two parties exchanging XML data can understand and interpret elements in the same way only if they share the same definitions of what they are. If two parties that share an XML document also share the same schema, they can be sure to understand the meaning of the same element tags in the same way. This is exactly how Web services work.

Technologies

XML is a family of technologies: a data markup language, various content models, a linking model, a namespace model, and various transformation mechanisms. The following are significant members of the XML family used as the basis of Web services:

- **XML v1.0**: The rules for defining elements, attributes, and tags enclosed within a document root element, providing an abstract data model and serialization format
- **XML schema**: XML documents that define the data types, content, structure, and allowed elements in an associated XML document; also used to describe semantic-processing instructions associated with document elements
- **XML namespaces**: The uniquely qualified names for XML document elements and applications
The Future of the Web

The inventor of the World Wide Web, Tim Berners-Lee, has said that the next generation of the Web will be about data, not text; XML is to data what HTML is to text. The next generation of the Web is intended to address several shortcomings of the existing Web, notably the difficulty searching the Web for exact matches on text strings embedded in Web pages. Because the Web has been so successful, however, the future of the Web must be accomplished as an extension, or an evolution, of the current Web. It’s impossible to replace the entire thing and start over! Solutions for application-to-application communication must be derived from existing Internet technologies.

If the future of the Web depends on its ability to support data communications as effectively and easily as it supports text communications, Web services need to be able to refer dynamically to Web end points, or addresses (URLs), and to map data to and from XML transparently. These end points, or addresses, provide the services that process the XML data, in much the same way that browsers process HTML text. These addresses also can be included in any program capable of recognizing a URL and parsing XML. Thus it will be possible to communicate from your spreadsheet to a remote source of data or from your money management program to your bank account management application, make appointments with colleagues for meetings, and so on.

Microsoft and others are already developing these kinds of standard services accessible from any program, and a large part of Microsoft’s .NET strategy is focused on development tools for creating and stitching together applications that use predefined Web services. But getting this to happen requires significant standardization, comparable to the effort involved in standardizing PC components, and might therefore not happen for several years.

- **XML Information Set**: A consistent, abstract representation of the parts of an XML document
- **XPointer**: A pointer to a specific part of a document; **XPath**, expressions for searching XML documents; and **XLink**, for searching multiple XML documents
Extensible Stylesheet Language Transformations (XSLT): Transformation for XML documents into other XML document formats or for exporting into non-XML formats

DOM (Document Object Model) and SAX (Simple API for XML): Programming libraries and models for parsing XML documents, either by creating an entire tree to be traversed or by reading and responding to XML elements one by one

These technologies and others are described in further detail in Chapter 2.

WSDL: Describing Web Services

The Web Services Description Language (WSDL) is an XML schema format that defines an extensible framework for describing Web services interfaces. WSDL was developed primarily by Microsoft and IBM and was submitted to W3C by 25 companies. WSDL is at the heart of the Web services framework, providing a common way in which to represent the data types passed in messages, the operations to be performed on the messages, and the mapping of the messages onto network transports.

WSDL is, like the rest of the Web services framework, designed for use with both procedure-oriented and document-oriented interactions. As with the rest of the XML technologies, WSDL is so extensible and has so many options that ensuring compatibility and interoperability across differing implementations may be difficult. If the sender and the receiver of a message can share and understand the same WSDL file the same way, however, interoperability can be ensured.

4. To date, 25 is the highest number of companies to join any W3C submission. A submission is a specification proposed for adoption by W3C—see www.w3.org/Submission/2001/07.
WSDL is divided into three major elements:

- Data type definitions
- Abstract operations
- Service bindings

Each major element can be specified in a separate XML document and imported in various combinations to create a final Web services description, or they can all be defined together in a single document. The data type definitions determine the structure and the content of the messages. Abstract operations determine the operations performed on the message content, and service bindings determine the network transport that will carry the message to its destination.

Figure 1-7 shows the elements of WSDL, layered according to their levels of abstraction, which are defined independently of the transport, specifically so that multiple transports can be
used for the same service. For example, the same service might be accessible via SOAP over HTTP and SOAP over JMS. Similarly, data type definitions are placed in a separate section so that they can be used by multiple services. Major WSDL elements are broken into subparts.

**WSDL interfaces are like CORBA or DCOM interfaces**

The definition parts include data type definitions, messages, and abstract operations, which are similar to interface definitions in CORBA or DCOM. Messages can have multiple parts and can be defined for use with the procedure-oriented interaction style, the document-oriented interaction style, or both. Through the abstraction layers, the same messages can be defined and used for multiple port types. Like the other parts of WSDL, messages also include extensibility components—for example, for including other message attributes.

**Web service data types are based on XML schemas but are extensible to any other mechanism**

WSDL data type definitions are based on XML schemas, but another, equivalent or similar type definition system can be substituted. For example, CORBA Interface Definition Language (IDL) data types could be used instead of XML schema data types. (If another type definition system is used, however, both parties to a Web services interaction must be able to understand it.)

**Abstract messages and operations are mapped to specific transports**

The service bindings map the abstract messages and operations onto specific transports, such as SOAP. The binding extensibility components are used to include information specific to SOAP and other mappings. Abstract definitions can be mapped to a variety of physical transports. The WSDL V1.1 specification includes examples of SOAP one-way mappings for SMTP (Simple Mail Transfer Protocol), SOAP RPC mappings for HTTP, SOAP mappings to HTTP GET and POST, and a mapping example for the MIME (multipurpose Internet messaging extensions) multipart binding for SOAP.

**Namespaces ensure WSDL element names’ uniqueness**

XML namespaces are used to ensure the uniqueness of the XML element names used in each of the three major WSDL elements. Of course, when the WSDL elements are developed...
separately and imported into a single complete file, the name-
spaces used in the separate files must not overlap. Associated
schemas are used to validate both the WSDL file and the mes-
sages and operations defined within the WSDL file.

It’s safe to say that WSDL is likely to include many extensions,
changes, and additions as Web services mature. Like SOAP,
WSDL is designed as an extensible XML framework that can
easily be adapted to multiple data type mappings, message
type definitions, operations, and transports. For example, IETF
(Internet Engineering Task Force) working groups are propos-
ing a new protocol standard—Blocks Extensible Exchange
Protocol (BEEP)—to define a useful connection-oriented trans-
port. (HTTP, by contrast, is inherently connectionless, making
it difficult to resolve quality-of-service problems at the trans-
port level.) Companies interested in using Web services for
internal applications or integration may choose to extend
WSDL to map to more traditional protocols, such as DCOM or
IIOP (Internet Inter-Orb Protocol).

**SOAP: Accessing Web Services**

So far, you have defined the data (XML) and expressed the
abstraction of the service necessary to support the communica-
tion and processing of the message (WSDL). You now need to
define the way in which the message will be sent from one
computer to another and so be available for processing at the
target computer.

The SOAP specification defines a messaging framework for
exchanging formatted XML data across the Internet. The mes-
saging framework is simple, easy to develop, and completely
neutral with respect to operating system, programming lan-
guage, or distributed computing platform. SOAP is intended to
provide a minimum level of transport on top of which more
complicated interactions and protocols can be built.
SOAP is fundamentally a one-way communication model that ensures that a coherent message is transferred from sender to receiver, potentially including intermediaries that can process part of or add to the message unit. The SOAP specification contains conventions for adapting its one-way messaging for the request/response paradigm popular in RPC-style communications and also defines how to transmit complete XML documents. SOAP defines an optional encoding rule for data types, but the end points in a SOAP communication can decide on their own encoding rules through private agreement. Communication often uses literal, or native XML, encoding.

As shown in Figure 1-8, SOAP is designed to provide an independent, abstract communication protocol capable of bridging, or connecting, two or more businesses or two or more remote business sites. The connected systems can be built using any combination of hardware and software that supports Internet
access to existing systems such as .NET and J2EE. The existing systems typically also represent multiple infrastructures and packaged software products. SOAP and the rest of the XML framework provide the means for any two or more business sites, marketplaces, or trading partners to agree on a common approach for exposing services to the Web.

SOAP has several main parts:

- **Envelope**: Defines the start and the end of the message
- **Header**: Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end point
- **Body**: Contains the XML data comprising the message being sent
- **Attachment**: Consists of one or more documents attached to the main message (SOAP with Attachments only)
- **RPC interaction**: Defines how to model RPC-style interactions with SOAP
- **Encoding**: Defines how to represent simple and complex data being transmitted in the message

Only the envelope and the body are required.

**UDDI: Publishing and Discovering Web Services**

After you have defined the data in the messages (XML), described the services that will receive and process the message (WSDL), and identified the means of sending and receiving the messages (SOAP), you need a way to publish the service that you offer and to find the services that others offer and that you may want to use. This is the function that UDDI (universal distribution, discovery, and interoperability) provides.
Introducing Web Services

The UDDI framework defines a data model in XML and SOAP application programming interfaces (APIs) for registering and discovering business information, including the Web services a business publishes. UDDI is produced by an independent consortium of vendors, founded by Microsoft, IBM, and Ariba, to develop an Internet standard for Web service description registration and discovery. Microsoft, IBM, Hewlett-Packard, and SAP are hosting the initial deployment of a public UDDI service, which is conceptually patterned after DNS, the Internet domain name service that translates Internet host names into TCP addresses. In reality, UDDI is much more like a replicated database service accessible over the Internet.

Inside the Enterprise

Many companies are exploring the potential advantages of using Web services both inside and outside the enterprise. This is analagous to using browsers and Web servers inside the enterprise in internal networks. Existing internal Web infrastructure can be put to good use in support of Web services–style interactions. Although unlikely to replace existing distributed computing environments, such as COM and CORBA, Web services can be a valuable supplement to existing technologies. Sometimes, all you have is an HTTP or an SMTP connection. Because they represent a completely neutral format that can be used to achieve a new level of interoperability, Web services can also be used to bridge across COM, CORBA, EJB, and message queueing environments. Finally, because Web services use existing HTTP infrastructure, the impact on system administrators is minimal compared to introducing other distributed computing technologies into an IT department. Performance is certainly an issue compared to more traditional binary-oriented transports and protocols, but the potential benefits outweigh the costs for many applications, and performance issues tend to get solved over time, as they have been for the original Web.

UDDI registers and publishes Web service definitions

The UDDI framework defines a data model in XML and SOAP application programming interfaces (APIs) for registering and discovering business information, including the Web services a business publishes. UDDI is produced by an independent consortium of vendors, founded by Microsoft, IBM, and Ariba, to develop an Internet standard for Web service description registration and discovery. Microsoft, IBM, Hewlett-Packard, and SAP are hosting the initial deployment of a public UDDI service, which is conceptually patterned after DNS, the Internet domain name service that translates Internet host names into TCP addresses. In reality, UDDI is much more like a replicated database service accessible over the Internet.

UDDI is similar in concept to a Yellow Pages directory. Businesses register their contact information, including such
details as phone and fax numbers, postal address, and Website. Registration includes category information for searching,
such as geographical location, industry type code, business

type, and so on. Other businesses can search the information
registered in UDDI to find suppliers for parts, catering services,
or auctions and marketplaces. A business may also discover
information about specific Web services in the registry, typi-
cally finding a URL for a WSDL file that points to a supplier’s
Web service.

Businesses use SOAP to register themselves or others with UDDI;
then the registry clients use the SOAP query APIs to search regis-
tered information to discover a trading partner. An initial query
may return several matches from which a single entry is chosen.
Once a business entry is chosen, a final API call is made to ob-
tain the specific contact information for the business.

Figure 1-9 shows how a business would register Web service in-
formation, along with other, more traditional contact information,

Figure 1-9  The UDDI repository can be used to discover a Web service.
Introducing Web Services

with the UDDI registry. A business first generates a WSDL file to
describe the Web services supported by its SOAP processor (1)
and uses UDDI APIs to register the information with the reposito-
tory (2). After a business submits its data to the registry, along
with other contact information, the registry entry contains a
URL that points to the SOAP server site’s WSDL or other XML
schema file describing the Web service. Once another busi-
ness’s SOAP processor queries the registry (3) to obtain the
WSDL or other schema (4), the client can generate the appro-
priate message (5) to send to the specified operation over the
identified protocol (6). Of course, both client and server have to
be able to agree on the same protocol—in this example, SOAP
over HTTP—and share the same understanding, or semantic
definition of the service, which in this example is represented
via WSDL. With the widespread adoption of these fundamental
standards, however, this common understanding of WSDL
seems ensured.

XML for Business Collaboration: ebXML

Several additional technologies, beyond what’s provided in
the basic Web services technologies, are required to support
true business-to-business interaction over the Web. The Elec-
tronic Business XML (ebXML) consortium, for example, has
defined a comprehensive set of specifications for an industrial-
strength usage pattern for XML document exchange among
trading partners. The ebXML messaging specification is based
on SOAP with Attachments and does not use WSDL but does
add several qualities of service, such as security, guaranteed
messaging, and compliance with business process interaction
patterns.

The ebXML spec
provides more than
basic Web services
technologies

The ebXML spec
defines XML use for
cooperaive busi-
tness processes

The ebXML initiative, the first phase of which ended in May
2001, was sponsored by an international group established by
the United Nations Center for Trade Facilitation and Electronic
Business (UN/CEFACT) and OASIS to research, develop, and
Web Services and EDI versus ebXML

Although electronic data interchange (EDI) has been around for more than two decades, it is very complex, has multiple interpretations, requires significant technical expertise to deploy, and is based on a tightly coupled, inflexible architecture. Although they can be deployed on public networks, EDI applications are most often used on expensive dedicated networks and require a lot of expertise to set up and run.

By contrast, ebXML and Web services hold the promise of realizing the original goals of EDI, making it simpler and easier to exchange electronic documents over the Internet. However, ebXML and Web services also will have to mature for several years before they encompass all EDI’s current functionality and feature set.

Although the ebXML consortium has completed its initial work, OASIS, UN/CEFACT, and other organizations continue to promote the adoption of the architecture and specifications to a broader audience, hoping to establish a global e-business marketplace through the standardized exchange of XML documents and messages, regardless of geographic or political boundaries.

5. Although the original ebXML effort ended in May 2001, work continues on specific OASIS (The Organization for the Advancement of Structured Information Standards) and UN/CEFACT committees to enhance and further extend the original ebXML specifications.
Introducing Web Services

and with the qualities of service that businesses expect. The ebXML architecture defines

- Business processes and their associated messages and content
- A registry and discovery mechanism for publishing business process sequences with related message exchanges
- Company profiles
- Trading-partner agreements
- A uniform message transport layer mapped to SOAP with multipart MIME attachments

Similarly to the way in which UDDI facilitates a search for Web service definitions, the ebXML architecture allows businesses to find one another by using a registry, to define trading-partner agreements, and to exchange XML messages in support of business operations. The goal is to allow all these activities to be performed automatically, without human intervention, over the Internet. The ebXML architecture has many similarities to SOAP/WSDL/UDDI, and some level of convergence is already taking place with the adoption of SOAP in the ebXML transport specification. RosettaNet also announced its adoption of the ebXML transport, as have many other vertical industry consortia.

The ebXML architecture clearly centers on document-oriented interactions; as ebXML gains acceptance, it may come to define the paradigm for B2B-oriented Web service interactions. Companies that already have been exchanging information electronically, perhaps using EDI standards, will find many parallels in the goals of ebXML, although ebXML aims at addressing this type of requirement more broadly and for the Internet.

Web Services versus Other Technologies

Web services are not as much like traditional distributed computing technologies such as CORBA, DCOM, and EJB, as they are like Web servers, HTML, and HTTP, on which they are based. Web services are fundamentally one-way, asynchronous messages mapped onto executable software programs. Web services define a data format independent of programming language, operating system, network transport, and data storage mechanism; therefore data has to be mapped into and out of the independent format. Data typing and structure are abstracted from underlying implementations of services.

Comparison of ebXML and SOAP

Initially, it seemed that the ebXML group was competing with the group of companies sponsoring SOAP, WSDL, and UDDI. In fact, the ebXML specifications cover a lot of the same territory as SOAP, WSDL, and UDDI. You could view the SOAP effort at W3C as a “bottom-up” approach, starting with a definition of the way to map XML documents to HTTP messages, and look at the ebXML effort as a “top-down” approach, starting with a definition of the business process as a series of messages mapped to any transport.

The ebXML group was formed primarily to create business process standards, the areas in which the work of ebXML has the most promise. The areas of transport, services description, and registry seem more appropriate to efforts focused more purely on issues of infrastructure than of business process and document interaction. One of the major motivators for ebXML is to produce standards that serve the same or similar purpose as EDI, including support for the emerging industry-specific XML “vocabularies.” It seems appropriate to consider the ebXML architecture as requirements on W3C and other XML-oriented initiatives as a way of ensuring that Web services will be ready for real business use, rather than as a competitive effort to define core infrastructure services.
Web services are often compared to remote procedure call invocations or software components. However, Web services are more appropriately compared to enterprise application integration adapters. Web services define a canonical message format, as EAI software systems, such as MQSeries, TIBCO, NEON, Vitria, and IONA’s Orbix E2A, do and define the way in which the message is directed to a service interface through which the data is mapped or transformed onto an underlying application. In other words, the intelligence for understanding how to map a message into a software program is not contained within the interface itself, as it is in CORBA, J2EE, and DCOM, all of which are based on RPC concepts, which tightly couple the service name to the program being invoked. Rather, that intelligence is contained within the XML processor, which consumes the message and follows associated instructions on how to parse the message and map the data into whatever program implements the Web service.

In addition, Web services do not require or assume the existence of the same software system on both ends of a communication path. EAI adapters similarly accept a canonical message format and map the information in the message to an enterprise resource planning (ERP) or other type of enterprise application. Web services are defined at a similar level of abstraction, which allows the same message type to be mapped to multiple applications, including, but certainly not limited to, RPC-based components.

Unlike RPC-oriented middleware, such as CORBA and DCOM, Web services use unidirectional, asynchronous messaging, which is more naturally mapped to a message queuing system, such as MQSeries or JMS, than to CORBA or DCOM; although, of course, Web services are also often mapped to CORBA-, J2EE-, and DCOM-based products. Web services support a request/response paradigm typical of synchronous, RPC-style communications through emulation; that is, the XML processor rather than the protocol correlates requests with replies. The HTTP map-
ping of SOAP, for example, does not support protocol-level request/reply correlation. The Web services emulation of an RPC is easily mapped to such traditional RPC-based systems as CORBA, EJB, and DCOM, although qualities of service (e.g., security, transactions, and exception handling), are likely to be very different from those available in traditional distributed computing technologies, which are often tied closely to the transport layer, and are specific to each technology.

Because interactions with Web services are accomplished through the programs and databases to which the Web services are mapped, the user experience is likely to be very different from a typical browser-based experience: Web services are more like traditional applications than like browsers, although, of course, browsers may be used. (As mentioned previously, Web services by themselves are not executable but instead have to be mapped to a program, an object, a middleware system, or a database management system.)

### Additional Technologies

Core Web services technologies, such as SOAP, WSDL, and UDDI, are useful for bridging disparate technology domains and submitting documents to business process flows. However, to become useful for more types of applications and to fulfill the complete vision of Web services as enabling the use of application building blocks over the Internet, Web services technologies have to be extended to encompass additional features, functions, and qualities of service.

The ongoing work of evolving Web services toward a more useful technology substrate is very similar to the evolution of

7. Various proposals address this issue, including HTTPR (reliable HTTP) and BEEP, a new session-oriented protocol from IETF; see Chapter 7 for further information.
the common object request broker architecture, undertaken by the Object Management Group (OMG) during the 1990s. The OMG work defined a comprehensive software architecture that guided an open, collaborative effort that produced a rich set of specifications for transactions, asynchronous messaging, security, failover, fault tolerance, and so on. The same type of effort is being initiated at W3C for Web services, and a similar architecture is evolving.

In the world of Web services, the major industry software vendors have already agreed on the core standards, which is the true test of standardization. Microsoft, IBM, Sun Microsystems, BEA Systems, Oracle, IONA, and others have agreed on implementing SOAP, WSDL, and UDDI, although some difference of opinion remains on the role of the ebXML registry. However, other than for the fundamental standards, proposals often compete, such as the difference of opinion between Microsoft and IBM on business process flow definition, that is, XLANG versus WSFL (Web Services Flow Language), and competing proposals for handling security context.

Additional technologies are focused primarily in the following key areas:

- Security
- Process flow
- Transactions
- Messaging

Some of the most important additional technologies for Web services involve security technologies.

Security is important to ensure the confidentiality and integrity of Web services data. No one other than the intended recipient of the data should be allowed to examine or to tamper with message contents. Security also is necessary to control access to
Web services, especially when multiple Web services are used together, so that only those for whom they are intended use them.

Proposed standards exist for authentication and authorization (SAML, or Security Authorization Markup Language) and for public key management for encryption (XKMS, or XML Key Management Specification). Of course, fundamental to all Internet security is Secure Socket Layer (SSL) and, for HTTP-based protocols, HTTPS (secure HTTP) for basic encryption-level security.

In addition to HTTPS, firewalls, SAML, XKMS, the use of digital signatures, and XML encryption, Microsoft has proposed WS-License for credential management and WS-Security\(^8\) for propagating security credentials associated with Web service interactions.

Process flow is critical to automating business process interactions over the Web and inside an enterprise. Process flow is also often called *orchestration* because it defines the relationship among a series of interactions necessary to accomplish a given purpose, such as completing a purchase order, processing a travel reservation, or executing a manufacturing plan. A flow is modeled as a sequence of steps defined for a given business process. The series of steps creates an aggregation of functions for which a Web service interface can be defined.

In the world of automated business operations, transactions have long played the part of enforcer, ensuring that the execution platforms produced consistent results from a series of related operations on data, despite software or hardware failures. These traditional protocols and techniques are not directly applicable to the Web, however, as they are designed for a tightly coupled environment in which it's possible to hold database

\(^8\) Recently submitted to OASIS.
locks pending notification of the transaction result and in which a connection-oriented protocol is available to detect communication failures automatically. The Business Transaction Protocol (BTP) proposal from OASIS is designed to resolve this problem for Web services by defining a loosely coupled protocol that ensures that the results of multiple Web service interactions are correctly propagated and shared.

Messaging protocols execute the communication patterns defined for Web service interactions, such as asynchronous one-way, request/response, broadcast, and conversational, or peer-to-peer. Additional Web services technologies also may depend on the messaging layer for certain qualities of service, such as reliable or guaranteed delivery, propagation of security and transaction contexts, and correctly routing messages along a defined path that includes one or more intermediaries. IBM has proposed reliable HTTP (HTTPR) to address requirements in this area.

IBM and Microsoft have collaborated on the WS-Inspection proposal for discovering information about Web services available at a particular message target. Microsoft has also proposed WS-Referral and WS-Routing to define a specific message path for a Web service, including any number of intermediaries, and how to route messages forward and backward along the specified route.

The Blocks Extensible Exchange Protocol (BEEP) from IETF defines a connection-oriented Internet protocol. A SOAP mapping for BEEP has been defined, and in this case, SOAP messages inherit the additional qualities of service from BEEP for maintaining session context at the sender and the receiver nodes. The context can be used to relate multiple messages into a larger unit of transfer and to relate multiple messages as coming from the same source or intended for the same target. Security and transaction context can also be associated with a connection.
Other relevant standards and technologies include many of those defined by the following organizations:

- OASIS, hosting ongoing ebXML and other related XML proposals, such as BTP, SAML, UDDI, and WS-Security
- RosettaNet, influencer of Web services concepts, developed by a group of electronics vendors for B2B business process flow interaction over the Internet
- UserLand, developer of XML-RPC, a precursor of SOAP
- OAGI (Open Applications Group, Inc.), defining canonical XML document formats for business and industry

The work of these and other groups often focuses on promoting the adoption of XML for specific business purposes, such as building on the base standards to define document formats and protocols for the electronics, financial, health care, and other industries. Because Web services are based on XML, the work of almost any standards body or consortium promoting the use of XML-related technologies for Internet business is relevant. Some of the other work, such as BTP and SAML, emerges as

The Long Road Ahead

Additional technologies, such as security, transactions, and reliable messaging, currently found in existing distributed computing environments, have to be defined again for Web services because of the fundamental shift involved in the infrastructure—XML and HTTP—on which they now need to be built. The World Wide Web Consortium will undertake the effort to define Web services architecture, just as OMG defined architecture for CORBA, although this is likely to be a very difficult and daunting task. The W3C is not set up to resolve major differences of opinion among its members, especially when those differences are motivated by commercial interests. This is the downfall of many standards efforts, in fact.
candidate technology for adoption by W3C within its Web services architecture activity.

**Vendor Approaches to Web Services**

Software vendors, both large and small, are providing Web services implementations as product add-ons or as entirely new products.

Web services do not fundamentally change existing software systems, although they can change how software systems are put together. The differences in implementation usually follow the differences in philosophy, or approach, of the vendors: Are Web services a fundamental enabling technology? Or are they simply entry and exit points to and from existing software systems? In other words, vendors vary in their approach to Web services, depending on the extent to which they view Web services as impacting existing software system architectures. For example, do Web services invalidate J2EE, or are they complementary? The answers to these and other similar questions can be discovered in a vendor’s approach.

The five basic approaches to Web services are to map them

- Into and out of a database management system
- Into and out of an application server
- Into and out of an integration broker
- Between technology domains
- To architectural software building blocks or functional components

In other words, Web services implementers fundamentally distinguish between Web services technologies and underlying software implementations. Web services, therefore, are either incidental aspects of existing software systems or a required part of the infrastructure.
Can an application server, object request broker, or database management system successfully continue to exist without support for Web services? Or can Web services exist on their own?

Thus the question is, where does the value lie? With application servers, database management systems, and integration brokers, leaving Web services to be merely a means of mapping data into and out of existing software systems? Or does the value lie with the Web services themselves, as fundamental to a new category of software systems?

Vendor implementations tend to be divided among these varying views of the value of Web services. Not surprisingly, Microsoft has its own view, whereas Sun Microsystems, IBM, BEA Systems, Oracle, and others are taking an alternative view. To some extent, this divergence of vendor views, or initiatives, represents a continuation of the Visual Basic/Java developer battle, but Microsoft is taking a very bold and aggressive stance on Web services, even breaking current Visual Basic applications to ensure that the future version of VB will support Web services as fundamental enabling technology. The Java community is taking a less radical view, extending Java APIs for Web services rather than requiring a rewrite to incorporate them.

Industry business consortia, such as ebXML and OASIS, as well as integration broker products from such vendors as IBM, Microsoft, IONA, and WebMethods, tend to focus on the business process, or document-oriented type of applications for Web services. Other vendors’ products, such as the Web services toolkits shipped with BEA’s WebLogic and IONA’s J2EE Edition, tend to focus on the RPC style of interaction. The same XML-based technologies and standards can generally be used for either, but initiatives and products tend to focus on one or the other because the paradigms are so different. In general, application servers tend to support the RPC style of interaction, whereas integration brokers tend to support the asynchronous document-oriented style of interaction.
Integration broker vendors, such as WebMethods, Vitria, SeeBeyond, Software AG, and Mercator, typically view Web services as an extension of classic enterprise and business-to-business integration technologies and have built adapters for Web services as they would build adapters for any other technology with which their products have to integrate. Other vendors, such as IONA, take a more neutral and encompassing view of Web services as both enabling technology for extending existing application server, CORBA, and COM middleware and as fundamental to the next generation of enterprise and business integration standards. IONA’s Orbix E2A product line provides not only Web services adapters for asynchronous, document-oriented processing and RPC-oriented Web services interfaces for CORBA and J2EE-compliant objects but also fundamental Web services building blocks. The IONA business process engine, XML conversion and transformation engine, packaged application adapters, and business protocol framework all export Web service interfaces. The IONA products support a consistent approach to application integration, using Web services technologies inside and outside the firewall.
Finally, a number of vendors view Web services as an interesting and potentially profitable technology in their own right and have developed “pure-play” Web services products. These products, based entirely on Web services technology, typically require other technologies and products. For example, Cape Clear markets a Web services product aimed at bridging J2EE and .NET. Shinka markets a product that presumes that Web services are a fundamental design center and that programs will be developed to map into them, rather than vice versa, which is what most other vendors appear to believe.

Summary

Web services are quickly becoming significant technology in the evolution of the Web and distributed computing. Web services leverage the data independence of XML to solve enterprise integration problems, both inside and outside the firewall. Web service interfaces are shells, or wrappers, that map to any type of software program, middleware system, database management system, or packaged application.

New types of applications are being created by using standard Web services building blocks, thus creating greater economies of scale in automating business and consumer interactions with the Web and with each other. Web services technologies are rapidly changing, and a long list of additional features and functionality is required to complete the vision. The basic Web services standards—SOAP, WSDL, and UDDI—are immediately useful for many applications, such as publishing interfaces to automated business processes, bridging disparate software domains, and connecting wireless clients to Web functions.

The ebXML initiative offers an alternative view of an XML-enabled distributed computing infrastructure, specifically aimed at connecting business process interactions among Internet trading partners. ebXML represents a form of industrial-strength
Web services, although ebXML does not include WSDL or UDDI. Many vendors view Web services and ebXML as significant aspects to be added to their existing products; other vendors view Web services as sufficient technology on which to base entire products.