Chapter 1

Introduction to SOA with Web Services

Complexity is a fact of life in information technology (IT). Dealing with the complexity while building new applications, replacing existing applications, and keeping up with all the maintenance and enhancement requests represents a major challenge.

If all applications were to use a common programming interface and interoperability protocol, however, the job of IT would be much simpler, complexity would be reduced, and existing functionality could be more easily reused. After a common programming interface is in place, through which any application can be accessed, existing IT infrastructure can be more easily replaced and modernized.

This is the promise that service-oriented development brings to the IT world, and when deployed using a service-oriented architecture (SOA), services also become the foundation for more easily creating a variety of new strategic solutions, including:

- Rapid application integration.
- Automated business processes.
- Multi-channel access to applications, including fixed and mobile devices.

An SOA facilitates the composition of services across disparate pieces of software, whether old or new; departmental, enterprise-wide, or inter-enterprise; mainframe, mid-tier, PC, or mobile device, to streamline IT processes and eliminate barriers to IT environment improvements.

These composite application solutions are within reach because of the widespread adoption of Web services and the transformational power of an SOA. The Web Services Description Language (WSDL) has become a standard programming interface to access any application, and SOAP has become a standard interoperability protocol to connect any application to any other. These two standards are a great beginning, and they are followed by many additional Web services specifications that define security, reliability, transactions, orchestration, and metadata management to meet additional requirements for enterprise features and qualities of service. Altogether, the Web services standards the best platform on which to build an SOA—the next-generation IT infrastructure.

The Service-Oriented Enterprise

Driven by the convergence of key technologies and the universal adoption of Web services, the service-oriented enterprise promises to significantly improve corporate agility, speed time-to-market for new products and services, reduce IT costs, and improve operational efficiency.

As illustrated in Figure 1-1, several industry trends are converging to drive fundamental IT changes around the concepts and implementation of service orientation. The key technologies in this convergence are:

- **Extensible Markup Language (XML)**—A common, independent data format across the enterprise and beyond that provides:
    - Standard data types and structures, independent of any programming language, development environment, or software system.
    - Pervasive technology for defining business documents and exchanging business information, including standard vocabularies for many industries.
    - Ubiquitous software for handling operations on XML, including parsers, queries, and transformations.
- **Web services**—XML-based technologies for messaging, service description, discovery, and extended features, providing:
    - Pervasive, open standards for distributed computing interface descriptions and document exchange via messages.
    - Independence from the underlying execution technology and application platforms.
    - Extensibility for enterprise qualities of service such as security, reliability, and transactions.
    - Support for composite applications such as business process flows, multi-channel access, and rapid integration.
- **Service-oriented architecture (SOA)**—A methodology for achieving application interoperability and reuse of IT assets that features:
    - A strong architectural focus, including governance, processes, modeling, and tools.
    - An ideal level of abstraction for aligning business needs and technical capabilities, and creating reusable, coarse-grain business functionality.

- A deployment infrastructure on which new applications can quickly and easily be built.
- A reusable library of services for common business and IT functions.

- **Business process management (BPM)**—Methodologies and technologies for automating business operations that:
    - Explicitly describe business processes so that they are easier to understand, refine, and optimize.
    - Make it easier to quickly modify business processes as business requirements change.
    - Automate previously manual business processes and enforce business rules.
    - Provide real-time information and analysis on business processes for decision makers.

Individually, each of these technologies has had a profound effect on one or more aspects of business computing. When combined, they provide a comprehensive platform for obtaining the benefits of service orientation and taking the next step in the evolution of IT systems.

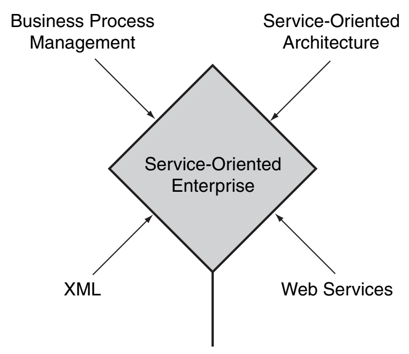
Business Process Management
Service-Oriented Architecture
Service-Oriented Enterprise
XML
Web Services

Figure 1-1 Trends converging to create the service-oriented enterprise.

Service-Oriented Development

Software vendors have widely adopted the paradigm of service-oriented development based on Web services. Service-oriented development is complementary to the object-oriented, procedure-oriented, message-oriented, and database-oriented development approaches that preceded it.

Service-oriented development provides the following benefits:

- **Reuse**—The ability to create services that are reusable in multiple applications.
- **Efficiency**—The ability to quickly and easily create new services and new applications using a combination of new and old services, along with the ability to focus on the data to be shared rather than the implementation underneath.
- **Loose technology coupling**—The ability to model services independently of their execution environment and create messages that can be sent to any service.
- **Division of responsibility**—The ability to more easily allow business people to concentrate on business issues, technical people to concentrate on technology issues, and for both groups to collaborate using the service contract.

Developing a service is different from developing an object because a service is defined by the messages it exchanges with other services, rather than a method signature. A service must be defined at a higher level of abstraction (some might say at the lowest common denominator) than an object because it's possible to map a service definition to a procedure-oriented language such as COBOL or PL/I, or to a message queuing system such as JMS or MSMQ, as well as to an object-oriented system such as J2EE or the .NET Framework.

It's also important to understand the granularity at which the service is to be defined. A service normally defines a coarse-grained interface that accepts more data in a single invocation than an object and consumes more computing resources than an object because of the need to map to an execution environment, process the XML, and often access it remotely. Of course, object interfaces can be very coarse-grained. The point is that services are designed to solve interoperability problems between applications and for use in composing new applications or application systems, but not to create the detailed business logic for the applications.

It's possible to create an aggregation of Web services such that the published Web service encapsulates multiple other Web services. This allows a coarse-grained interface to be decomposed into a number of finer-grained services (or multiple finer-grained services to be composed into a coarse-grained interface). The coarse-grained service may make more sense to publish, while the finer-grained services may make more sense as "private" Web services that can be invoked only by the coarse-grained Web service.

Services are executed by exchanging messages according to one or more supported message exchange patterns (MEPs), such as request/response, one-way asynchronous, or publish/subscribe.

At a project level, an architect typically oversees the development of reusable services and identifies a means to store, manage, and retrieve service descriptions when and where they are needed. The reusable services layer insulates business operations such as "get customer" or "place an order" from variations in the underlying software platform implementations, just as Web servers and browsers insulate the World Wide Web from variations in operating systems and programming languages. The ability of reusable services to be composed into larger services quickly and easily is what provides the organization the benefits of process automation and the agility to respond to changing conditions.

How XML Helps Simplify Systems Development and Integration

The use of XML in Web services provides a clear separation between the definition of a service and its execution. This separation in the standards is intentional so that Web services can work with any software system. The

continues

XML representation, provided through an XML Schema, of the data types and structures of a service allows the developer to think about the data being passed among services without necessarily having to consider the details of a given service implementation. This represents a change in the nature of the integration problem from having to figure out the implementation of the service in order to talk to it. Whether the service's execution environment is an object, message queue, or stored procedure doesn't matter. The data is seen through the filter of a Web service, which includes a layer that maps the Web service to whatever execution environment is implementing the service.

One way to help accomplish this significant turnaround in the way we think about how to design, develop, and deploy applications using services may be to divide the responsibility within IT departments between those who:

- **Create the services**—Dealing with the complexity of the underlying technology on which the service is being deployed and ensuring that the XML/Web services descriptions are what the service consumer needs and that they share the right data.
- **Consume the services**—Assembling new composite applications and business process flows, ensuring that the shared data and process flows accurately reflect operational and strategic business requirements.

This potential division of responsibility more cleanly separates the technical issues from the business issues.

Organizational Implications of SOA

Previously, the same individuals in the IT department were responsible for understanding both business and technical functions—and this remains a classic problem for IT, that is, getting the same person or persons to bridge business and technology domains. To gain the full benefit of Web services,

continues

SOA, and BPM technologies, IT departments should consider the best organization and skill set mix. It's important when adopting a new architecture and a new technology to identify new roles and responsibilities. Among the important considerations is that technical staff must be able to reorient themselves from thinking about doing the entire job to doing a piece of the job that will be completed by someone else. A service needs to be developed within a larger context than an object or a procedure because it is more likely to be reused. In fact, defining services for reuse is probably the most important part of service orientation. To obtain their highest value, services must be developed in the context of other services and used in combination with them to build applications. This change in thinking is likely to require someone in a departmental or corporate leadership position to help review designs and ensure that they are in line with these new IT goals.

Service Abstraction

A service is a location on the network that has a machine-readable description of the messages it receives and optionally returns. A service is therefore defined in terms of the message exchange patterns it supports. A schema for the data contained in the message is used as the main part of the contract (i.e., description) established between a service requester and a service provider. Other items of metadata describe the network address for the service, the operations it supports, and its requirements for reliability, security, and transactionality.

Figure 1-2 illustrates the relationship among the parts of a service, including the description, the implementation, and the mapping layer between the two. The service implementation can be any execution environment for which Web services support is available. The service implementation is also called the *executable agent*. The executable agent is responsible for implementing the Web services processing model as defined in the various Web services specifications. The executable agent runs within the *execution environment*, which is typically a software system or programming language.

Newcomer_01.qxd 11/22/04 8:26 AM Page 9

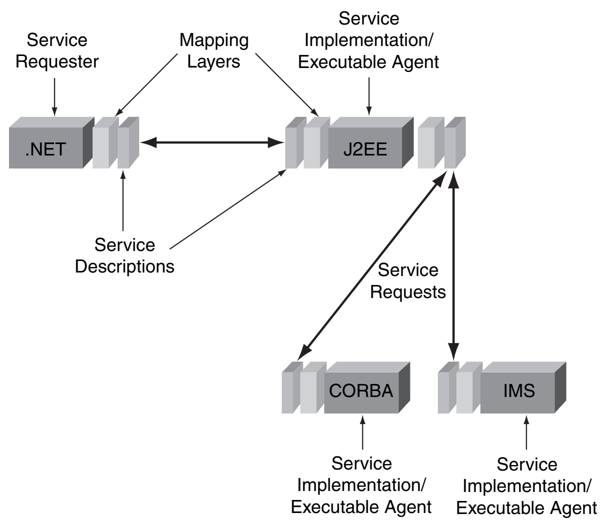
Service Requester
Mapping Layers
Service Implementation/ Executable Agent
.NET
J2EE
Service Descriptions
Service Requests
CORBA
IMS
Service Implementation/ Executable Agent
Service Implementation/ Executable Agent

Figure 1-2 Breakdown of service components.

An important part of the definition of a service is that its description is separated from its executable agent. One description might have multiple different executable agents associated with it. Similarly, one agent might support multiple descriptions. The description is separated from the execution environment using a mapping layer (sometimes also called a transformation layer). The mapping layer is often implemented using proxies and stubs. The mapping layer is responsible for accepting the message, transforming the XML data to the native format, and dispatching the data to the executable agent.

Web services roles include requester and provider. The service *requester* initiates the execution of a service by sending a message to a service provider. The service *provider* executes the service upon receipt of a message and returns the results, if any are specified, to the requester. A requester can be a provider, and vice versa, meaning an execution agent can play either or both roles.

As shown in Figure 1-3, one of the greatest benefits of service abstraction is its ability to easily access a variety of service types, including newly developed services, wrapped legacy applications, and applications composed of other services (both new and legacy).

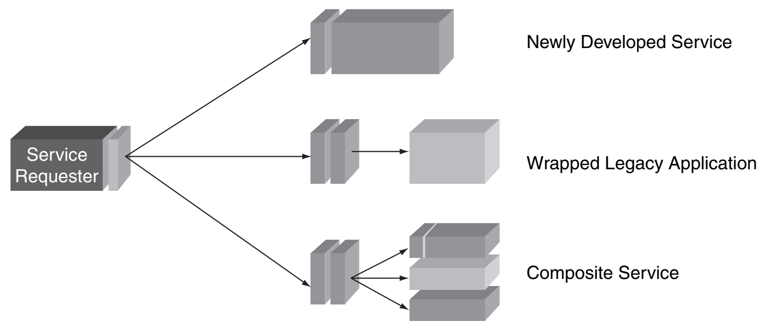

Figure 1-3 Requesting different types of services.

Separating the Service from the Product

Some software vendors still don't separate the idea of a service from the idea of an execution environment, and they continue to sell Web services implementations only as part of another, typically pre-existing product. This practice can make it more difficult to obtain the benefits of services because the products have features that may not be required to execute Web services and that may create incompatibilities if the products make the services dependent upon them.

Service-Oriented Architecture

This section introduces the major concepts and definitions for services and SOA.

What Are Services?

Before we continue to discuss technology, let's discuss the notion of services and processes from a business perspective. Most organizations (whether commercial or government) provide services to customers, clients, citizens, employees, or partners. Let's look at an example of service orientation in practice.

As illustrated in Figure 1-4, bank tellers provide services to bank customers. Different tellers may offer different services, and some tellers may be specifically trained to provide certain types of services to the customer. Typical services include:

- Account management (opening and closing accounts).
- Loans (application processing, inquiries about terms and conditions, accepting payments).
- Withdrawals, deposits, and transfers.
- Foreign currency exchange.

Several tellers may offer the same set of services to provide load balancing and high availability. What happens behind the counter does not matter to the customer, as long as the service is completed. Processing a complex transaction may require the customer to visit several tellers and therefore implement a business process flow.

Figure 1-4 Service analogy at the bank.

Behind the counter are the IT systems that automate the bank's services. The services are provided to the customer via the tellers. The services implemented by the IT systems must match and support the services provided by the tellers.

Newcomer_01.qxd 11/22/04 8:26 AM Page 12

A consistent approach to defining services on the IT systems that align with business functions and processes makes it easier for the IT systems to support the goals of the business and adapt more easily to providing the same service through humans, ATMs, and over the Web.

Figure 1-5 shows how the same service can be accessed from customers at the ATM, tellers on the office network, or Web users from their PCs. The services are designed and deployed to match the services that customers need. The implementation environments for the services don't matter; it's the service that's important. This figure also illustrates how two services can easily be combined to create another service, such as how the withdrawal and deposit service are composed into a transfer service.

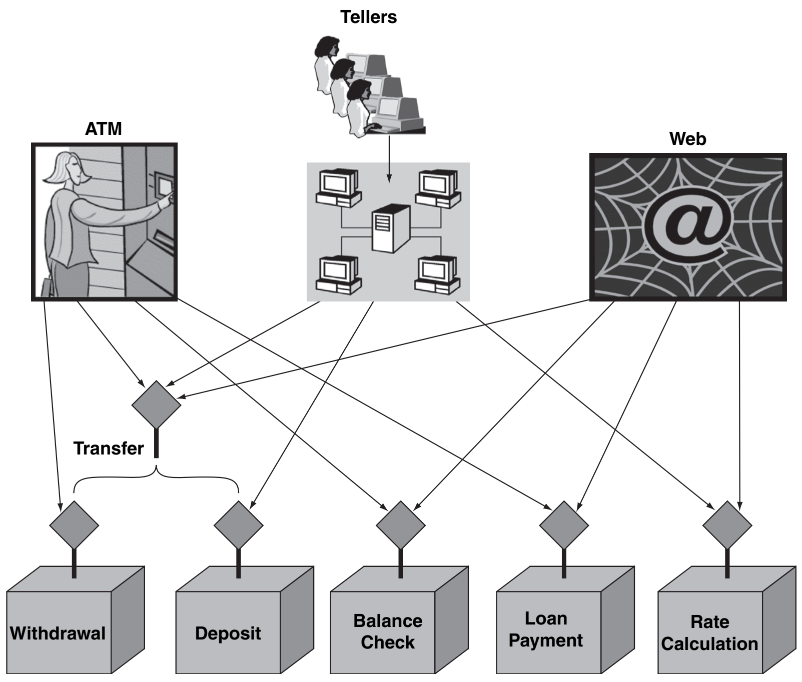

Figure 1-5 Accessing and composing services.

The definition of software services aligns with the business services that a bank offers to ensure smooth business operations and to help realize strategic goals such as providing ATM and Web access to banking services in addition to providing them in the branch office.

More complex applications can be composed from services, as we'll see, such as processing a purchase order or an insurance claim. Deploying services in the context of an SOA makes it easier to compose services into simple as well as complex applications, which can also be exposed as services that can be accessed by humans and IT systems alike.

What Is Service-Oriented Architecture?

A service-oriented architecture is a style of design that guides all aspects of creating and using business services throughout their lifecycle (from conception to retirement). An SOA is also a way to define and provision an IT infrastructure to allow different applications to exchange data and participate in business processes, regardless of the operating systems or programming languages underlying those applications.

An SOA can be thought of as an approach to building IT systems in which business services (i.e., the services that an organization provides to clients, customers, citizens, partners, employees, and other organizations) are the key organizing principle used to align IT systems with the needs of the business. In contrast, earlier approaches to building IT systems tended to directly use specific implementation environments such as object orientation, procedure orientation, and message orientation to solve these business problems, resulting in systems that were often tied to the features and functions of a particular execution environment technology such as CICS, IMS, CORBA, J2EE, and COM/DCOM.

Competitive Value of SOA

Businesses that successfully implement an SOA using Web services are likely to have a competitive advantage over those who do not because those who have services aligned with strategic IT business goals can react more quickly to changing business requirements than those who have IT systems

continues

aligned to a particular execution environment. It's easier to combine Web services, easier to change Web services compositions, and cheaper to change the Web services and XML data than it is to change execution environments. The advantages and benefits of SOA with Web services include a better return on investment for IT spending on projects, a faster time to results for the projects, and an ability to more quickly respond to changing business and government requirements. Any business that can implement an IT infrastructure that allows it to change more rapidly has an advantage over a business that cannot do the same. Furthermore, the use of an SOA for integration, business process management, and multi-channel access should allow any enterprise to create a more strategic IT environment, one that more closely matches the operational characteristics of the business.

SOA is the best way to capitalize on the value of service-oriented development. Service orientation reduces project costs and improves project success rates by adapting technology more naturally to the people who need to use it, rather than focusing (as the previous generations of IT systems have) on the technology itself, which forces people to adapt to the technology. The major difference between service-oriented development and previous approaches is that service orientation lets you focus on the description of the business problem, whereas previous approaches require you to focus more on the use of a specific execution environment technology. The way in which services are developed better aligns them with solving business problems than was the case with previous generations of technology.

The concept of SOA isn't new—what is new is the ability to mix and match execution environments, clearly separating the service interface from the execution technology, allowing IT departments to choose the best execution environment for each job (whether it's a new or existing application) and tying them together using a consistent architectural approach. Previous implementations of SOA were based on a single execution environment technology.

SOA Isn't New, So What Is?

Everyone is talking about SOA, although the concepts behind it aren't new. The idea of separating an interface from its implementation to create a software service definition has been well proven in J2EE, CORBA, and COM, and even DCE before that. But the ability to more cleanly and completely separate—basically by interpreting a text file—a service description from its execution environment is new. This ability is part of what Web concepts and technologies bring to Web services. The traditional implementations of the interface concept might not have considered such a "loose" separation because the performance implications are negative. However, in many cases, the performance issue is less important than the ability to more easily achieve interoperability, something the industry has long strived for but only partially achieved until now. The success or failure of SOA, however, does not depend upon the advance in IT software brought about by Web services. Rather, it depends upon a change in approach. The greater separation of interface from execution environment in Web services facilitates the separation of work responsibilities as well. Separating the service description from its technology implementation means that businesses can think about and plan IT investments around the realization of operational business considerations, as represented by the description, more so than the capabilities of any individual product or software technology chosen to execute the description. In this case, the description becomes the definition of a kind of lowest common denominator—a set of features and functions that everything can support. But this is achievable only if businesses change their way of thinking about IT. A service is something that's just available—just there for the consumption. Of course, this is an exaggeration; plenty of services and applications still have to be developed to reach the ideals of automating a business operation so quickly and flexibly that it just becomes a given, like office-software automation. No single software solution can address every requirement any more than the same computer system can run a car and calculate the trajectory for the Mars Rover. The same type of computer system cannot, and should not, operate an insulin pump and process orders for Amazon.com and tracks Web links for Google. The world

continues

is by its very nature diverse, and SOA with Web services embraces this diversity and provides the ability to create IT systems that map better to business operations than anything previously. However, this will take a significant change in thinking, not just for IT departments, but also for the entire software industry. It's equally hard to imagine that a single software vendor would be an expert in all aspects of software as it is to imagine that a single hardware vendor is an expert in all aspects of computing. Specialization is what the industry needs, along with the ability to create reusable assemblies of those components. In this vision of a future environment, software vendors are likely to become even more specialized, perhaps shipping assemblies of services instead of complete products. In an SOA-enabled world, enterprises very likely will have to learn not only to think about services as distinct from execution environments, but also how to assemble applications out of components from a variety of vendors.

The real value of SOA comes from the later stages of deployment, when new applications can be developed entirely, or almost entirely, by composing existing services. When new applications can be assembled out of a collection of existing, reusable services, the best value for effort can be realized (that is, the lowest cost and fastest time to results and best ROI). But it takes some time to reach this point, and significant investment in service development may be required.

It's easy to understand the benefit of reusing common business services such as customer name lookup, ZIP Code validation, or credit checking. In a pre-service oriented development environment, these functions might be performed by reusable code libraries or class libraries that are loaded or linked into new applications. In SOA-based applications, common functions such as these, as well as typical system functions such as security checks, transaction coordination, and auditing are instead implemented using services. Using services not only reduces the amount of deployed code, but it also reduces the management, maintenance, and support burden by centralizing the deployed code and managing access to it. However, the performance implications of accessing services instead of using internal functions must be assessed because using a service typically consumes more computing resources than reusable code libraries.

The key to a successful SOA is determining the correct design and function of the services in the reusable service library, which should ultimately reflect the operational characteristics of the organization, such as how grants are applied for, how cash is managed overnight, or how containers are transferred from ships to trucks. The operational characteristics of the business are what need to be automated, and the successful SOA project will ensure that the reusable software services are properly aligned with the operational business processes. The successful alignment of business services and their implementation in software ensures that operational business processes can be changed quickly and easily as external environmental changes cause an organization to adapt and evolve.

CORBA vs. Web Services for SOA

People familiar with the CORBA standard often remark that Web services are simply CORBA implemented using XML and note the number of features Web services are missing compared to CORBA. Many CORBA deployments are SOAs, in fact, and the original goals of CORBA are very similar to the goals of Web services. Cynics say that CORBA didn't succeed widely because of vendor politics, and there's some truth to that. However, CORBA also hurt itself in its early days by not defining a standard for interoperability. When asked about this, OMG presenters used to say, "It's an exercise left up to the vendors and the customers to work out." The implication, and sometimes the direct statement, was that interoperability didn't matter if you had a standard interface. Web services actually started with SOAP, which is an interoperability standard. Many people still use SOAP without WSDL, which is perfectly possible, indicating another contrast between the two. In CORBA, it's impossible to use the interoperability transport without the interface definition language (IDL)—in fact, everything is generated from the IDL. Many Web services toolkits also generate proxies and stubs from WSDL and also generate the SOAP messages. But this is an implementation choice,

continues

not part of the SOAP standard. From a technical perspective, it is certainly true that you can use CORBA for almost everything you can use Web services for. And for many applications, CORBA remains a better choice. However, from a human perspective, which is what really counts in the end, if someone is unfamiliar with CORBA or new to distributed computing, Web services are much easier to learn and use, and the missing features don't matter as much as the interoperability.

Challenges to Adoption

The main challenges to adoption of SOA include ensuring adequate staff training and maintaining the level of discipline required to ensure the services that are developed are reusable. Any technology, no matter how promising, can be abused and improperly used. Services have to be developed not simply for immediate benefit, but also (and perhaps primarily) for long-term benefit. To put it another way, the existence of an individual service isn't of much value unless it fits into a larger collection of services that can be consumed by multiple applications, and out of which multiple new applications can be composed. In addition, the definition of a reusable service is very difficult to get right the first time.

Another challenge is managing short-term costs. Building an SOA isn't cheap; reengineering existing systems costs money, and the payback becomes larger over time. It requires business analysts to define the business processes, systems architects to turn processes into specifications, software engineers to develop the new code, and project managers to track it all.

Another challenge is that some applications may need to be modified in order to participate in the SOA. Some applications might not have a callable interface that can be service-enabled. Some applications are only accessible via file transfer or batch data input and output and may need additional programs for the purpose of service-enablement.

Of course, incrementally adopting SOA and leveraging SOA where it will have the greatest business impact can mitigate the challenges and amortize their costs, especially when services can be used to solve tactical problems along the way. Part of adopting Web services and SOA therefore is to identify those projects that provide immediate value by solving an immediate problem (such as integrating J2EE and .NET Framework applications) and at the same time lay the foundation for a departmental or enterprise SOA.

What Web Services Are Good For

Sometimes in the Web services literature, you will see a lot of discussion about things that Web services are not good for, such as developing and deploying mission-critical applications. It's a mistake, however, to assume that they never will be good for this. Other times, you'll see talk about identifying the "golden copy" or "single reference" data item instance for a particular data type, such as customer ID or customer name. This is indeed a problem that needs solving, but it also isn't part of the job of the Web services. This means it needs to be solved at the SOA level rather than at the Web services level. These discussions often indicate unfamiliarity with Web services and the problems they are trying to solve. The same people who would never imagine that a single tool in a workshop could be used for every purpose often make the mistake of thinking that Web services must be good for everything that all other technologies that preceded them are good for. This represents thinking that each new technology wave somehow obsoletes the previous wave or takes over everything that came before. Web services are not just adding more technology to the problems of IT; they are proposing a different approach to solving some of the problems of IT, especially around integration, because of new capabilities offered by the technology. Web services are not really a replacement technology; they are not the same thing as a new programming language like Java or C#, which you could reasonably assume must include all the major features of other successful programming languages. Web services are not really a new middleware system in the sense that J2EE, CORBA, and the .NET Framework are middleware systems. Web services are XML-based interface technologies; they are not executable; they do not have an execution environment—they depend upon other technologies for their execution environments. If you don't

continues

Newcomer_01.qxd 11/22/04 8:26 AM Page 20

rethink your approach to IT based on the features, functions, and capabilities of Web services, of course you're not going to get the value out of them that you should. Using Web services successfully requires a change in thinking about technology, not simply learning a new grammar for the same old way of building and deploying systems. Web services currently and will always require a mix of technologies. Therefore, Web services need to be understood in terms of what they add to the picture, not only in the context of what they replace.

SOA and Web Services

The major advantages of implementing an SOA using Web services are that Web services are pervasive, simple, and platform-neutral.

As shown in Figure 1-6, the basic Web services architecture consists of specifications (SOAP, WSDL, and UDDI) that support the interaction of a Web service requester with a Web service provider and the potential discovery of the Web service description. The provider typically publishes a WSDL description of its Web service, and the requester accesses the description using a UDDI or other type of registry, and requests the execution of the provider's service by sending a SOAP message to it. The basic Web services standards are good for some SOA-based applications but are not adequate for many others.

Why UDDI Is Not a Core Web Services Specification

It's safe to say that the original vision of UDDI has not been realized. When UDDI was launched in late 2000, it was intended to become a public directory. Companies were supposed to register their Web services with UDDI, and other companies were to come along later and dynamically discover the services they needed to access over the Internet. The assumption, which has not proven true, was that companies would be interested in discovering and requesting services from providers with whom they had no prior relationship. Also UDDI was developed before WSDL, so initially

continues

WSDL was not well supported. The data structures proved to be problematic because they are so open-ended with very little required information and a structure based on categorization data that isn't universally recognized. UDDI was also positioned as an inside-the-enterprise technology, and here it has gained some measure of success; however, the standards remain incomplete for this purpose. Companies adopting UDDI for internal use have to define their own naming conventions and categorization structure and metadata, which inhibits adoption. While SOAP and WSDL have gone on to tremendous success and widespread adoption, UDDI still struggles to find its proper place in the Web services universe. It is clear that a service registry is a required part of the Web services platform, but it isn't clear that UDDI will ever truly become that solution.

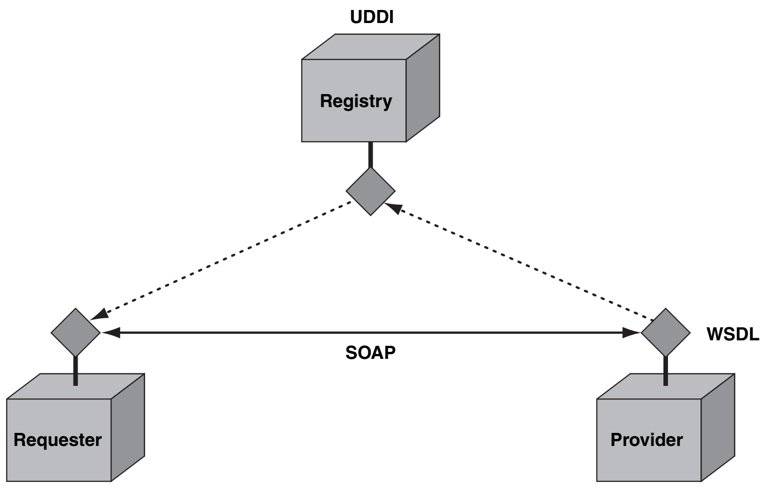

Figure 1-6 Basic Web services architecture.

Besides the core Web services specifications (SOAP and WSDL), a wide array of extended Web services specifications for security, reliability, transactions, metadata management, and orchestration are well on their way toward standardization, providing SOA-based solutions the necessary enterprise-level qualities of service to support a wide variety of mission-critical, corporate-wide projects.

Borrowing from the Web

Some of the important advantages of using Web services as the technology platform for an SOA are derived from the way in which the World Wide Web achieved its tremendous success; in particular, the fact that a simple document markup language approach such as HTML (or XML) can provide a powerful interoperability solution and the fact that a lightweight document transfer protocol such as HTTP can provide an effective, universal data transfer mechanism. On the Web, it doesn't matter whether the operating system is Linux, Windows, OS390, HP NonStop, or Solaris. It doesn't matter whether the Web server is Apache or IIS. It doesn't matter whether the business logic is coded in Java, C#, COBOL, Perl, or LISP. It doesn't matter whether the browser is Netscape, Internet Explorer, Mozilla, or the W3C's Amaya. All that matters is that the Web servers understand an HTTP request for an HTML file and that the browser understands how to render the HTML file into the display. Web services provide the same level of abstraction for IT systems. Similarly, all that matters for Web services is that they can understand and process an XML-formatted message received using a supported communications transport and return a reply if one is defined. Just as HTML and HTTP can be added to any computer system with a TCP connection, Web services can be added to any computer that understands XML and HTTP or XML and most other popular communications transports.

Figure 1-7 illustrates the features and capabilities of the complete Web services platform on which the broad range of SOA-based applications can be built. It includes the basic and extended Web services specifications. See Chapter 2, "Overview of Service-Oriented Architecture," for a complete description of the Web services platform.

The Web services platform contains the basic and extended features necessary to support an SOA, along with an enterprise service bus (ESB) to connect the services.

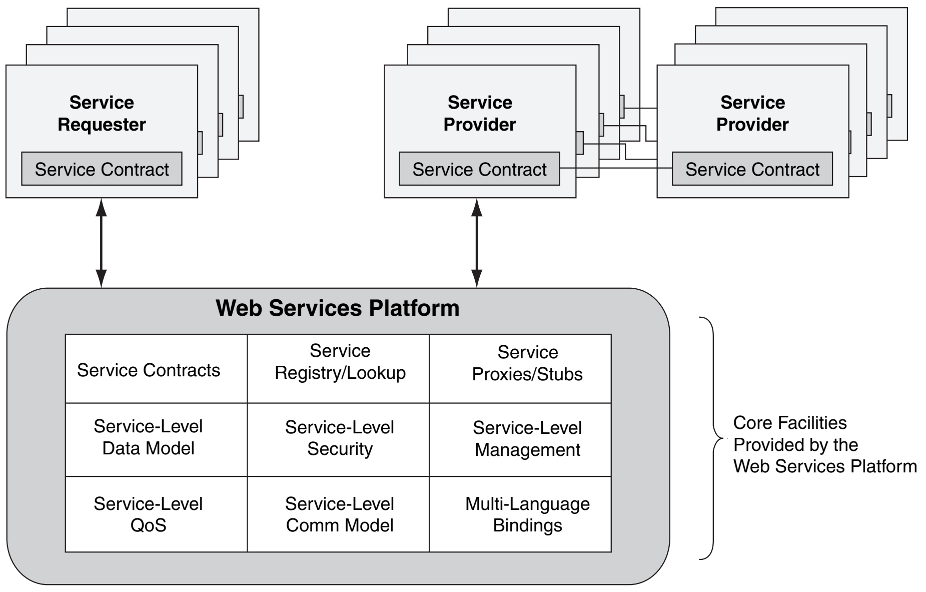

Figure 1-7 Web services platform.

Rapid Integration

A few years ago, businesses finding themselves in need of comprehensive integration solutions turned to products and practices developed specifically for that purpose. However, these enterprise application integration (EAI) products proved to be expensive, consumed considerable time and effort, and were subject to high project failure rates. Furthermore, because these various special purpose products are proprietary, many of the projects resulted in additional difficulties whenever a company invested in more than one of them.

Recent experience shows that a better answer is available by using Web services standards. Instead of dealing with the complexity of multiple incompatible applications on multiple computers, programming languages, and application packages by introducing an EAI product, it's possible to add a layer of abstraction that's open, standards-based, and easy to integrate with virtually any new and existing environment.

A new generation of integration products from BEA, IBM, IONA, Microsoft, SAP, SeeBeyond, Systinet, Tibco, WebMethods, and others, enabled by Web services technology, is emerging around the concepts of service-oriented integration.

The combination of Web services and SOA provides a rapid integration solution that more quickly and easily aligns IT investments and corporate strategies by focusing on shared data and reusable services rather than proprietary integration products.

To illustrate the benefits of service-oriented integration, consider the example of three fairly typical financial industry database applications, perhaps supporting retail banking, commercial banking, and mutual fund investment operations. The applications were developed using a classic three-tier architecture, separating presentation logic, business logic, and database logic.

As shown in Figure 1-8, it's possible to reuse a traditional three-tier application as a service-oriented application by creating services at the business logic layer and integrating that application with other applications using the service bus. Another benefit of service orientation is that it's easier to separate the presentation logic from the business logic when the business logic layer is service-enabled. It's easier to connect various types of GUIs and mobile devices to the application when the business logic layer is service-enabled than if a separate tightly coupled presentation logic layer has to be written for each. Instead of running the presentation logic tier as a tightly coupled interface on the same server, the presentation logic can be hosted on a separate device, and communication with the application can be performed using the service bus.

Applications can more easily exchange data by using a Web service defined at the business logic layer than by using a different integration technology because Web services represent a common standard across all types of software. XML can be used to independently define the data types and structures. And finally, the development of service-oriented entry points at the business logic tier allows a business process management engine to drive an automatic flow of execution across the multiple services.

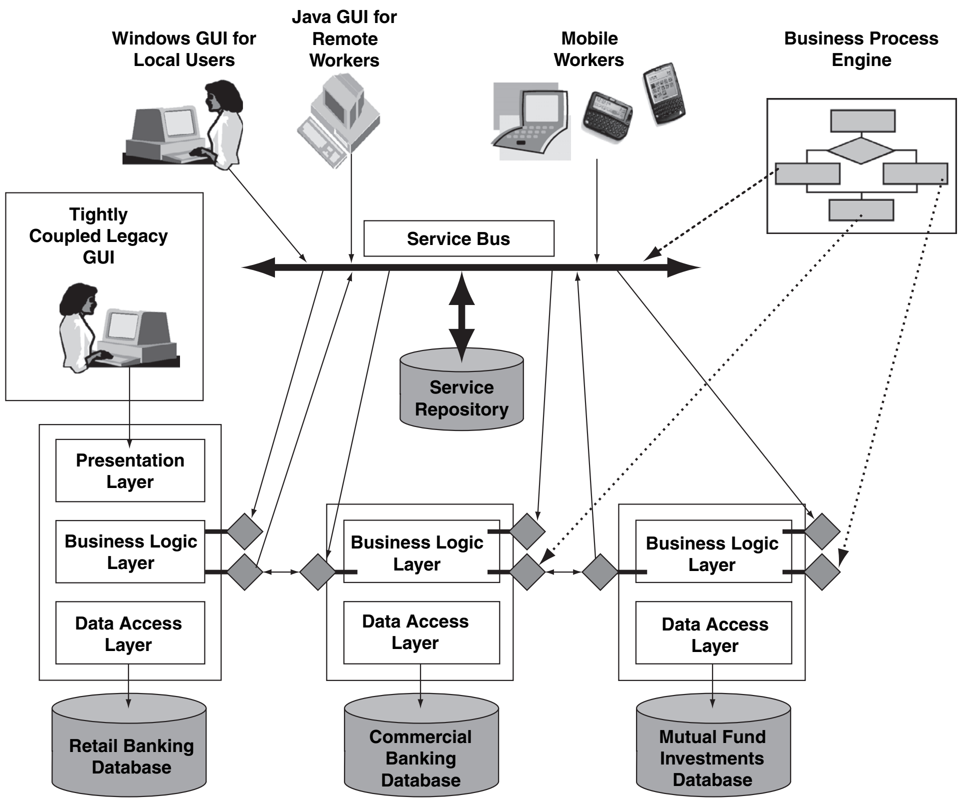

Figure 1-8 Designing for service-oriented integration.

Creating a common Web services layer or “overlay” of services into the business logic tiers of applications also allows you to use a common service repository in which to store and retrieve service descriptions. If a new application wishes to use an existing service into one of these applications, it can query the repository to obtain the service description and easily generate SOAP messages to interact with it.

Multi-Channel Access

The primary purpose of most organizations (commercial, government, non-profits, and so on) is to deliver services to clients, customers, partners, citizens, and other agencies. These organizations often use many channels for service delivery to reach their customers to ensure good service and maintain customer loyalty. Just as a bank prefers to offer services in a variety of ways for the convenience of their customers, whether using the Web, an ATM, or a teller window, many other organizations similarly benefit from being able to deliver a mixture of direct and indirect customer services over a mixture of access channels.

In general, business services change much less frequently than the delivery channels through which they are accessed. Business services represent operational functions such as account management, order management, and billing, whereas client devices and access channels are based on new technologies, which tend to change more frequently.

Web services interfaces are good for enabling multi-channel access because they are accessible from a broad range of clients, including Web, Java, C#, mobile devices, and so on. SOA with Web services is, therefore, well suited to simplifying the task for any organization to enable these multiple channels of access to their services.

Figure 1-9 illustrates an example of multi-channel access in which an organization's customer service application might expose various services for reporting a problem, tracking the status of a problem report, or finding new patches and error reports published to the user community. A customer account manager might want to access the services from his cell phone, to discover any updates to the customer's problem report list, for example, before going on a sales call. If the product was shipped through a reseller, the reseller might provide its own customer service, which would be tied to the supplier company's application to provide first- and second-level support, respectively. The customer service manager for a given major account might benefit from direct access to all of the features, functionality, and information stored in the organization's customer service application. The customers themselves might want to check the status of a trouble ticket from a mobile PDA device. And finally, the support center employees need access to the services in the application to perform their own jobs of interacting with the customers who have problems.

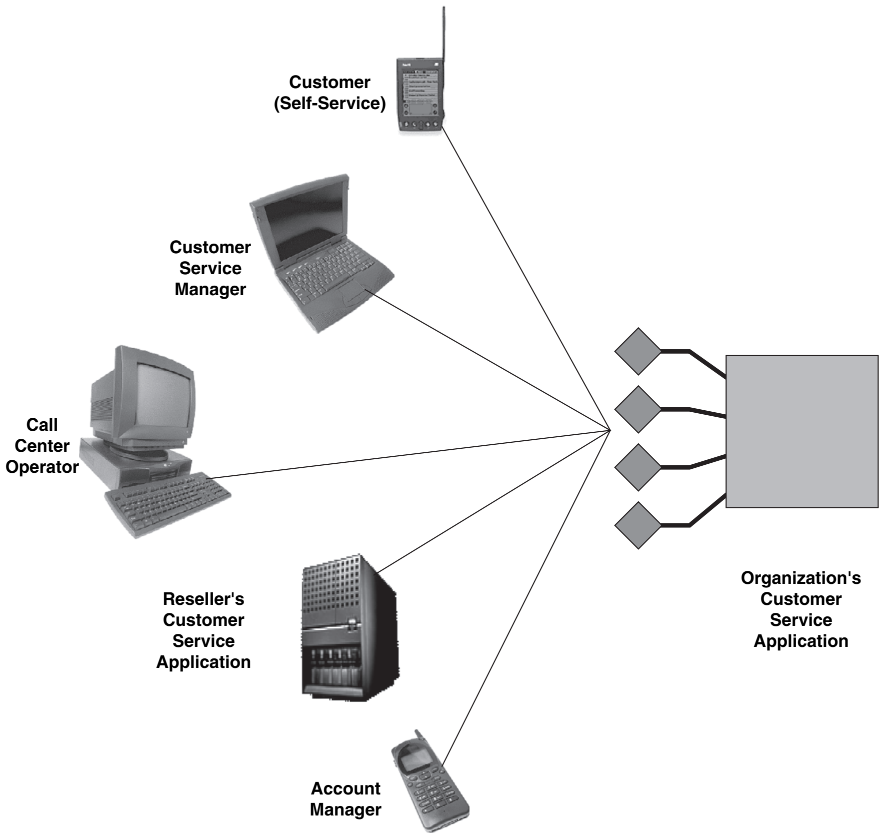
Customer (Self-Service)
Customer Service Manager
Call Center Operator
Reseller's Customer Service Application
Account Manager
Organization's Customer Service Application

Figure 1-9 Multi-channel access to customer service.

In the past, organizations often developed solutions like this as monolithic applications tied to a single access channel, such as a 3270 terminal, PC interface, or a browser. The proliferation of access channels, including new end-user devices, represents an opportunity for a service-oriented enterprise to better serve its customers, suppliers, and partners anytime and anywhere. However, it also represents a significant challenge to IT departments to convert existing monolithic applications to allow multi-channel access. The basic solution is, of course, to service-enable these applications using an SOA with Web services.

Occasionally Connected Computing

Integrating mobile devices into an SOA presents specific challenges because the mobile devices may not always be connected to a network. Mobile devices may also move through network connectivity zones, picking up new IP addresses when they reconnect. Most applications in production today are not designed to handle such variances in network connectivity. A new generation of "mobilized" software is emerging to integrate mobile devices quickly and easily into the service-oriented enterprise.

As shown in Figure 1-10, the SOAP messages in a mobile software solution are transported from the mobile client to the server using a store-and-forward asynchronous protocol that allows the mobile client to keep working even when a network connection isn't available. When the connection is available, transactions flow to the server-side SOA-based application directly using the messaging transport. When a connection isn't available, transactions are stored locally so that they can be transmitted when a connection becomes available.

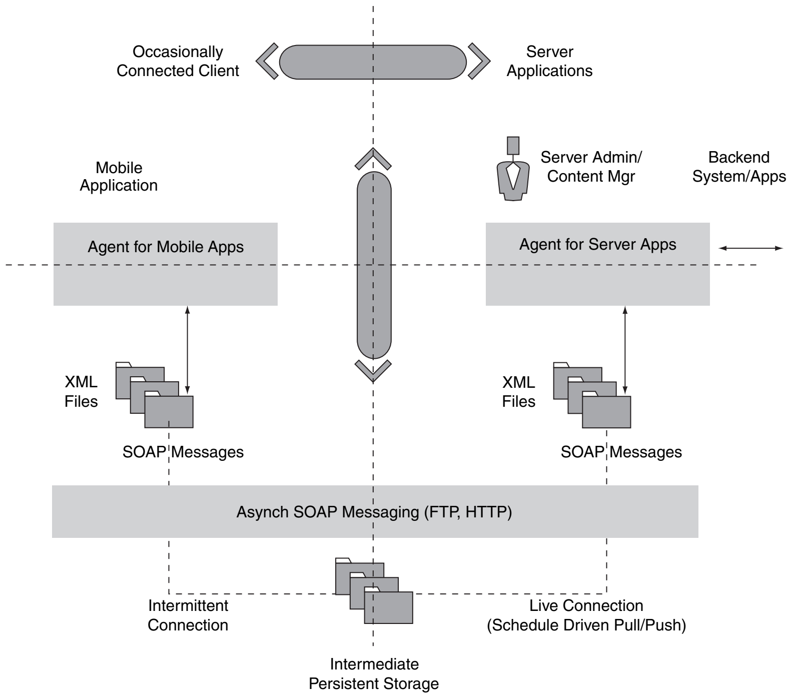

Figure 1-10 Occasionally connected architecture for mobile devices.

Business Process Management

A business process is a real-world activity consisting of a set of logically related tasks that, when performed in the appropriate sequence and according to the correct business rules, produce a business outcome, such as order-to-cash, purchase order fulfillment, or insurance claim processing.

Business process management (BPM) is the name for a set of software systems, tools, and methodologies that addresses how organizations identify, model, develop, deploy, and manage such business processes. A business process may include IT systems and human interaction or, when completely automated, simply the IT systems. Various BPM solutions have been in use for a long time, starting with early workflow systems and progressing up to modern Web services orchestration.

BPM techniques can exploit the foundation and the architectural work provided by an SOA to better automate business processes. This is among the important reasons for investing in a Web services-based SOA because Web services help achieve the goals of BPM more quickly and easily.

BPM systems are designed to help align business processes with desirable business outcomes and ensure that the IT systems support those business processes. BPM systems let business users model their business processes graphically, for example, in a way that the IT department can implement. All IT systems support and implement business processes in one form or another. What makes BPM unique is the explicit separation of business process logic from other application code (this contrasts with other forms of system development where the process logic is deeply embedded in the application code).

Separating business process logic from other application code helps increase productivity, reduce operational costs, and improve agility. When implemented correctly (for example, using BPM as a consumer of an SOA with Web services), organizations can more quickly respond to changing market conditions and seize opportunities for gaining a competitive advantage. Further efficiencies are gained when the graphical depiction of a business process can be used to generate an executable specification of the process.

Business Operational Changes

Most businesses have special operational characteristics derived from the reasons they got into business in the first place. One example is a pizza shop with a 30-minute delivery guarantee. To really make this happen without losing money, all kinds of operational characteristics have to be taken into account, such as pizza baking time, order-taking time, and delivery time within a defined area. Obviously, this is not the kind of operation that can be completely automated because it relies upon human drivers, but certainly many parts of the process could be automated, such as ordering from the Web, automatically triggering the delivery of new supplies to the restaurant, and using robots to prepare and cook the pizzas. Ideally, you'd like to be able to introduce as many operational efficiencies as possible with minimal cost to the IT systems involved. An SOA-based infrastructure can help.

BPM simplifies the problem of how to combine the execution of multiple Web services to solve a particular business problem. If we think about a service as the alignment of an IT system with a business function such as processing a purchase order, we can think about the BPM layer as something that ties multiple services together into a process flow of execution to complete the function, such as validating the order, performing a credit history check on the customer, calculating inventory to determine the ability to fill the order, and finally shipping the order and sending the customer an invoice. By taking the process flow out of the application-level code, the business process can be more easily changed and updated for new application features and functions such as a change in suppliers, inventory management, or the shipping process.

Figure 1-11 illustrates the kind of graph that a business analyst might produce for automating the flow of purchase order processing. The flow starts with the input of a purchase order document. The first processing step is responsible for accepting the document, checking security credentials and providing acknowledgment so that the sender knows the document was received.

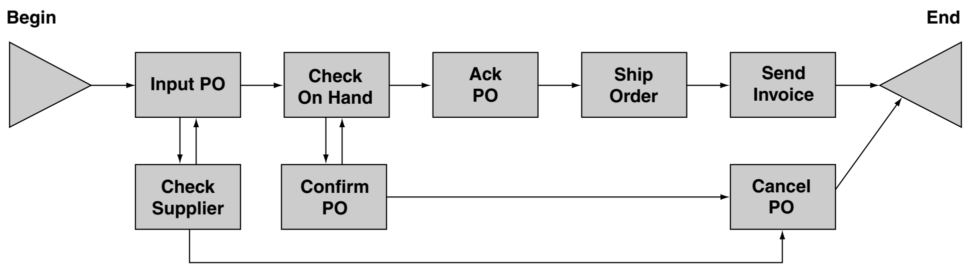

Figure 1-11 Example business process flow for a purchase order.

A typical process engine persists the input document so that the subsequent steps in the graph can access it. After the document is validated, a reference to the document's location in the database or file system is passed to the next step so that it can check whether on-hand inventory is sufficient to fill the required item quantities. If sufficient quantity is on hand, the next step in the flow acknowledges receipt of the purchase order and lets the purchaser know that the company can fill it. This acknowledgment can be sent using email or a Web service message.

At this point, the customer may have an opportunity to reconfirm the order, given the quoted price and delivery schedule. If the customer does not reconfirm, the process is cancelled and moves to the cancellation step (which basically erases the work of steps in the business process to that point, perhaps including one or more compensating transactions to release inventory that had been reserved for this customer and to remove the order from scheduled deliveries). If the customer reconfirms, the next step is activated to prepare the order for shipment. After the shipment confirmation has been received, the process moves to the final step, which sends an invoice to the customer.

Process flows are typically broken into individual tasks that call a Web service. Flows typically include tests that branch the flow based on the results of exectuting a task. Branches of the flow can handle errors. For example, when an item is out of stock from one supplier, the flow might branch to a task that invokes a Web service request to another supplier to find out whether that supplier has the item. Sometimes no one has the item, in which case, the flow might raise an error to the operator or submitter so they can decide what to do.

With a common services-based solution available to IT managers, architects, and developers, productivity increases will be much easier to achieve, and business systems will support the kind of flexibility demanded by the constant market and regulatory changes that are a fact of business and government life. In particular, laying an SOA foundation for business process management allows enterprises to concentrate on higher-level issues such as implementing the best business processes, rather than worrying about the technical details of application implementations.

Extended Web Services Specifications

Following the broad adoption and use of the basic Web services specifications—SOAP and WSDL—requirements have grown for the addition of extended technologies such as security, transactions, and reliability that are present in existing mission-critical applications. These extended features are sometimes also called *qualities of service* because they help implement some of the harder IT problems in the SOA environment and make Web services better suited for use in more kinds of SOA-enabled applications.

A class of applications will find the core specifications sufficient, while other applications will be blocked or hampered by the lack of one or more of the features in the extended specifications. For example, companies may not wish to publish their Web services without adequate security or may not wish to accept purchase orders without reliable messaging guarantees.

The core specifications were defined with built-in extensibility points such as SOAP headers in anticipation of the need to add the extended features.

Standardization

Web services specifications progress toward standardization through a variety of ways, including small groups of vendors and formally chartered technical committees. As a general rule of thumb, most specifications are started by a small group of vendors working together and are later submitted to a standards body for wider adoption. Specifications initially created by Microsoft and IBM, together with one or more of their collaborators (these vary by specification, but typically include BEA, Intel, SAP, Tibco, and Verisign), tend to gain the most market traction. Microsoft and IBM are the de facto leaders of the Web services specification movement and have defined or helped to define all the major specifications. Several of the WS-* specifications remain under private control at the time of writing, but we expect them to be submitted to a standards body in the near future.

Standards bodies currently active in Web services include:

- **World Wide Web Consortium (W3C)**—Making its initial name on progressing Web standards, notably HTTP, HTML, and XML, the W3C is home to SOAP, WSDL, WS-Choreography, WS-Addressing, WS-Policy, XML Encryption, and XML Signature.
- **Organization for the Advancement of Structured Information Standards (OASIS)**—Originally started to promote interoperability across Structured Generic Markup Language (SGML[1]) implementations, OASIS changed its name in 1998 to reflect its new emphasis on XML. OASIS is currently home to UDDI, WS-Security, WS-BPEL, WS-Composite Application Framework, WS-Notification, WS-Reliability, Web Services Policy Language (part of the Extensible Access Control Markup Language TC), and others such as Web Services for Remote Portlets, Web Services Distributed Management, and Web Services Resource Framework, which are not covered in this book.[2]
- **Web Services Interoperability (WS-I)**—Established in 2002 specifically to help ensure interoperability across Web services implementations, WS-I sponsors several working groups to try to resolve incompatibilities among Web services specifications. WS-I produces specifications called *profiles* that provide a common interpretation of other specifications and provides testing tools to help Web services vendors ensure conformance to WS-I specifications.

[1] Both HTML and XML are derived from SGML.

[2] These specifications are not all covered in this book because the book is focused on SOA.

- **Internet Engineering Task Force (IETF)**—The IETF is responsible for defining and maintaining basic Internet specifications such as TCP/IP and SSL/TLS. Their relationship to Web services is indirect in that TCP/IP is the most common communications protocol used for the HTTP transport, and basic IETF security mechanisms are used in Web services. The IETF collaborated with the W3C on XML Signature.
- **Java Community Process (JCP)**—Established by Sun to promote the adoption of Java and control its evolution, the JCP is home to several Java Specification Requests (JSRs) that define various Java APIs for Web services, including JAX-RPC for the Java language bindings for SOAP, JAX-B for XML data binding, and Java APIs for WSDL.
- **Object Management Group (OMG)**—Initially established to create and promote specifications for the Common Object Request Broker (CORBA), the OMG is home to specifications that define WSDL language mappings to C++ and CORBA to WSDL mappings.

Web services standardization started with the submission of the SOAP 1.1 specification to the W3C in mid-2000. After that, SOAP with Attachments, XKMS, and WSDL were submitted to W3C. At the same time, UDDI was launched in a private consortium and was later submitted to OASIS. Other major specifications submitted to OASIS include WS-Security, WS-BPEL, WS-CAF, and WS-Notification. More recently, WS-Addressing and WS-Policy were submitted to W3C, signaling a potential shift back toward W3C as the home of most of the major specifications.

Historically, OASIS is also the home of the ebXML set of specifications, which overlap to a large extent with the Web services stack. Web Services and ebXML share SOAP, but beyond that, the stacks diverge. ebXML has its own registry and its own orchestration (or choreography) language.

Standardization and Intellectual Property Rights

One of the difficulties with respect to Web services standardization is the fact that no single standards body is clearly in a leadership position.

continues

Specifications work is split across W3C, OASIS, WS-I, IETF, and OMG. How does a specification become an adopted standard? If anyone had a magic formula for this, they would be millionaires. In the end, it's market acceptance and adoption that makes the difference, and this means the economic factors become paramount, both in terms of vendor investment and customer procurement. Web services vendors often initiate work on specifications informally in small teams for the sake of rapid progress, publish the specifications themselves, and then submit the specifications to a standards body. Microsoft pioneered this approach with SOAP 1.1, and other vendors including BEA, HP, IBM, IONA, Oracle, SAP, Sun, Tibco, WebMethods, and others have significantly contributed to specifications this way. However, a major question with this approach often arises when it's time to submit the specification to an open standards body. In particular, it's necessary to resolve issues related to intellectual property (IP) rights, including copyrights and patents. It's an important step in a specification's lifetime, even if the standards body doesn't change it very much. Only when a specification is submitted to an open standards body and the IP issues resolved (or at least publicly declared) can that specification truly achieve widespread adoption. Without this step, software vendors not among the initial authoring community have no visibility into specification changes, which could invalidate their investments in products, and they might be faced with potential IP licensing fees, whether royalties or otherwise, that the specification owners might wish to charge.

Specification Composability

As mentioned previously, the SOAP and WSDL specifications are designed to be extended by other specifications. Two or more of the extended specifications can be combined within a single SOAP message header. For example:

```
<S:Header>

  <wsse:Security>
  ...
  </wsse:Security>

  <wsrm:Sequence>
  ...
  </wsrm:Sequence>

</S:Header>
```

This example illustrates the use of extended headers for security and reliability. The security header typically includes information such as the security token that can be used to ensure the message is from a trusted source. The reliability header typically includes information such as a message ID and sequence number to ensure the message (or set of messages) is reliably received.

Note the separate namespaces used for the security and reliability headers, `wsse:` and `wsrm:`, respectively. The headers use different namespaces so that they can be added incrementally to a SOAP message without concern over potential name clashes. Duplicate element and attribute names are not permitted in an XML document (and a SOAP message is an XML document, after all). Namespace prefixes provide a unique qualifier for XML element and attribute names, thus protecting names from duplication. This is one way in which SOAP header extensions work composably with each other.

Adding extended features may or may not require modification to existing Web services—that is, the extended features can be added into the SOAP headers without changing the SOAP body in many cases. But the execution environments and mapping layers may need to change in order to handle the extensions. Certainly at least adding SOAP headers for extended features must be done within the context of knowing whether the execution environment can support them and how; otherwise, the extended headers will not work.

Web service extensions are also added to the responsibility of the SOAP processors in the execution environment. Policy declarations associated with the WSDL contracts can be used during the generation of SOAP messages to determine what should go into the headers to help the execution environment negotiate the appropriate transport protocol or to agree on features such as transaction coordination.

As illustrated in Figure 1-12, each additional extended feature added to the Web service invocation results in additional processing before sending the message or after receiving it. Later chapters provide further detail on each of these extended features. The extended features may also be related to requirements from a business process engine and may need to be supported by the registry.

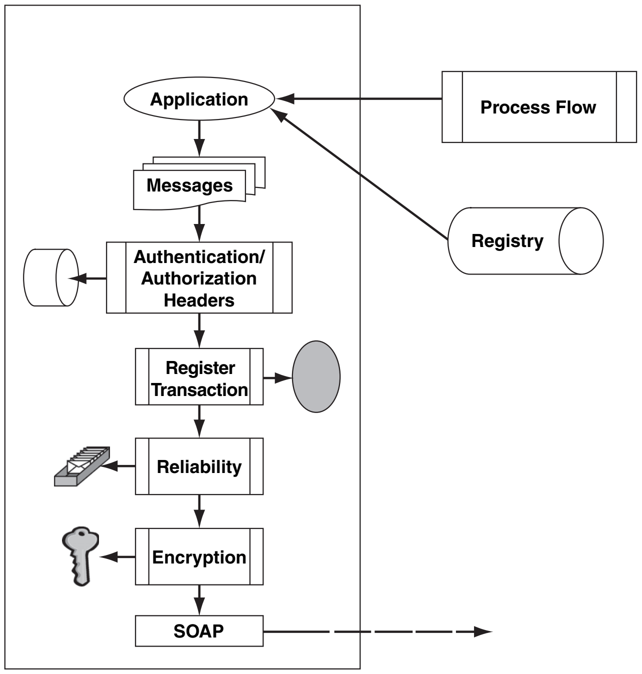

Figure 1-12 Adding extended features to SOAP.

Composability and Complexity

Supposedly, the composability of extended Web services specifications allows their incremental use or "progressive discovery" of new concepts and features. IBM and Microsoft, as the de facto leaders of the Web services specifications development, are obviously keen to avoid making them too complex. A large part—if not the largest part—of the value of Web services derives from their relative simplicity. CORBA, for example, is often criticized for being too complex and too hard to use. Of course, CORBA is

continues

easier than what preceded it, but CORBA is relatively complex compared to Web services. The issue with complexity is finding people well enough trained to use the technology productively, and because the highest IT cost is still labor, complexity typically means additional project expense. So for Web services to keep their promises, and to avoid having the whole effort fall apart, IBM and Microsoft want to preserve simplicity even as they start adding complex features to the basic specifications. One argument is that Web services are designed to inherently support composition of new features, meaning existing applications can be extended instead of being changed. However, this is a relatively untested assertion because products that fully implement the extended specifications are not yet available, and it isn't at all clear that the products will implement the extended features in the same way. Furthermore, there is no overall architecture for Web services that defines how the extended features really work with each other. When you receive a complex SOAP message full of headers for security, reliability, and transactions, what do you process first? The security header? The reliability header? The transaction header? No one really knows. If this all proves too complex, people may just go back to plain old XML over HTTP and hand code the extensions.

Metadata Management

Metadata management includes the description information about a Web service necessary to construct a message body (including its data types and structures) and message headers so that a service requester can invoke a service. The provider of the service publishes the metadata so that a requester can discover it and use it to construct messages that can be successfully processed by the provider.

When invoking a service, it's important to understand not only the data types and structures to send but also the additional qualities of service provided (if any), such as security, reliability, or transactions. If one or more of these features are missing from the message, it may prevent successful message processing.

Metadata specifications include:

- **XML Schema**—For message data typing and structuring and expressing policy information.
- **WSDL**—For associating messages and message exchange patterns with service names and network addresses.
- **WS-Addressing**—For including endpoint addressing and reference properties associated with endpoints. Many of the other extended specifications require WS-Addressing support for defining endpoints and reference properties in communication patterns.
- **WS-Policy**—For associating quality of service requirements with a WSDL definition. WS-Policy is a framework that includes policy declarations for various aspects of security, transactions, and reliability.
- **WS-MetadataExchange**—For querying and discovering metadata associated with a Web service, including the ability to fetch a WSDL file and associated WS-Policy definitions.

Service binding is different for an SOA based on Web services compared to an SOA based on J2EE or CORBA, for example. Instead of binding via reference pointers or names, Web services bind using discovery of services, which may be dynamic. If the service requester can understand the WSDL and associated policy files supplied by the provider, SOAP messages can be generated dynamically to execute the provider's service. The various metadata specifications are therefore critical to the correct operation of an SOA based on Web services.

Addressing

Addressing is an important requirement of extended Web services because no directory of Web services endpoint addresses exists on the Web. SOAP messages must include the endpoint address information within the message for all but the simplest MEP. WS-Addressing replaces earlier proposals called WS-Routing and WS-Referral.

Without an addressing solution, when you send a Web service request to a provider, typically the only address the provider has is the return address to the requester, and then only for the duration of the session. If anything goes wrong on the reply, there's no good way to retry it—basically the requester's address can be lost when there's a communication failure. Also there's no good way to specify a different return address than the requester's address. And finally, there's no way to define address schemes or to identify endpoint addresses for complicated message exchange patterns or multi-transport support.

Policy

Policy is necessary for expressing any extended Web services features of a service so that the requester can provide the necessary security, transaction, or reliability context in addition to the data requirements for the messages expressed in the WSDL file.

Policy provides a machine-readable expression of assertions that a service requester must adhere to in order to invoke upon a provider. Does the service require security or support transactions? The latter can be very important when trying to figure out whether or not a long running, complex interaction can involve a transaction, or whether a transaction can span across all the Web services identified for it.

WS-Policy is necessary for achieving interoperability for the extended features because the policy declarations are the only way in which a requester can discover whether a provider requires some or all of the extended features. In the case of security, for example, different providers may support different kinds of tokens, such as X.509 or Kerberos. WS-Security is designed as a kind of open framework that can carry any token type. However, if the token type the provider expects isn't declared, the requester can only guess at what it must be.

When making the decision to invoke the provider's service, it may also be important to discover whether it supports reliability or transactions. You might want to know, for example, whether the provider's service can accept retries if the original submission fails and whether it will let you know when it has successfully received a message. Finally, you may want to know whether or not to send a transaction context to the provider to enroll the provider Web service in the transaction.

Acquiring Metadata

It's possible that a requester will obtain the metadata it needs using WS-MetadataExchange or another similar mechanism that queries the WSDL and associated policy files directly. WS-MetadataExchange uses a feature of WS-Addressing called "actions" to access the metadata published by a service provider. WS-MetadataExchange is designed to provide any and all information about a Web service description—essentially replacing UDDI for most applications.

Developers may or may not use UDDI, despite its existence. It's fair to say that the public UDDI does not provide the metadata management facilities required to support interoperability requirements at the extended specification level and that WS-MetadataExchange may be needed for requesters to ensure they have the information they need to achieve interoperability with providers using extended features.

Security

Security concerns apply at every level of the Web services specification stack and require a variety of mechanisms to guard against the numerous challenges and threats that are a part of distributed computing. The mechanisms may have to be used in combination to guard against a particular threat or combination of threats. In the Web services and SOA world, it's particularly important to evaluate the need for protection at the network layer, the message layer, and for the data in the message.

Basic security protection mechanisms are built around encryption, authentication, and authorization mechanisms and typically include comprehensive logging for problem tracking. The industry has achieved consensus around a single specification framework, WS-Security, although ongoing work is necessary to complete the profiles and additional related specifications.

WS-Security was developed to provide message-level security on an end-to-end basis for Web services messages. Typical HTTP-based security mechanisms, such as SSL, provide only transport-level point-to-point protection. Sometimes additional security may be provided through the use of an alternative transport mapping, such as CORBA's IIOP or WebSphere MQ, but as with the rest of the extended features, the security specifications are written for HTTP as a kind of default or lowest common denominator transport and therefore can be applied to any transport.

WS-Security headers include the ability to carry strong authentication formats such as Kerberos tickets and X.509 certificates and can use XML Encryption and XML Signature technologies for further protecting the message contents. Although a WS-Security authorization profile for the Security Assertion Markup Language (SAML) is being developed, SAML can also be used on its own for exchanging authorization information.

Additional specifications from IBM, Microsoft, Verisign, and others further extend and complement WS-Security, including:

- **WS-SecurityPolicy**—Defines security assertions detailing a Web service's requirements so that the service requester can meet them.
- **WS-Trust**—Defines how to establish overall trust of the security system by acquiring any needed security tokens (such as Kerberos tickets) from trusted sources.
- **WS-SecureConversation**—Defines how to establish and maintain a persistent context for a secure session over which multiple Web service invocations might be sent without requiring expensive authentication each time.
- **WS-Federation**—Defines how to bridge multiple security domains into a federated session so that a Web service only has to be authenticated once to access Web services deployed in multiple security domains.

Because Web services are XML applications, and because XML has security challenges of its own (it is basically human-readable text sent over the Internet), XML-based security technologies are also often important for protecting the XML data before and after it's included in a SOAP message. These technologies include:

- **XML Encryption**—Designed to provide confidentiality, using a variety of supported encryption algorithms, of part or all of an XML document to ensure that the contents of a document cannot be intercepted and read by unauthorized persons.

- **XML Signature**—Designed to provide integrity, using a variety of encryption and signature mechanisms, to ensure that service providers can determine whether or not documents have been altered in transit and that they are received once and only once.

XML Encryption and XML Signature can be used to protect Web services metadata as well as data.

Reliability and Messaging

Messaging includes SOAP and its various message exchange patterns (MEP). The industry has not achieved consensus on a single, unified set of specifications for advanced messaging. However, competing specifications in the categories of reliability and notification work essentially the same way, and so an amalgam of the two is used here for the sake of introduction.

In general, reliable messaging is the mechanism that guarantees that one or more messages were received the appropriate number of times. Reliable messaging specifications include:

- WS-Reliability.
- WS-ReliableMessaging (from IBM and Microsoft).

Reliable messaging is designed to ensure reliable delivery of SOAP messages over potentially unreliable networks such as the HTTP-based Internet.

Reliable messaging is a protocol for exchanging SOAP messages with guaranteed delivery, no duplicates, and guaranteed message ordering. Reliable messaging works by grouping messages with the same ID, assembling messages into groups based on message number, and ordering them based on sequence number.

Reliable messaging automates recovery from certain transport-level error conditions that the application would otherwise have to deal with on its own. Reliable messaging also supports the concept of bridging two proprietary messaging protocols over the Internet.

Also in the general messaging area are specifications for extended MEPs such as event notification and publish/subscribe, which basically extend the asynchronous messaging capability of Web services. Specifications in this area include:

- WS-Eventing.
- WS-Notification.

Notification delivers messages through an intermediary often called a message broker or event broker. Subscribers identify the channels or topics for which they wish to receive messages. Publishers send messages to the channels or topics on which subscribers are listening. Notification is a messaging mechanism that can be used to set up broadcast and publish/subscribe messaging.

Transactions

Transactions allow multiple operations, usually on persistent data, to succeed or fail as a unit, such as processing an order or transferring funds. One of the most important aspects of transaction processing technologies is their ability to recover an application to a known state following an operating system or hardware failure. For example, if any failure occurs before a funds transfer operation is completed (that is, both the debit and credit operations), transactions ensure the bank account balances are what they were before the funds transfer operation started.

Many Web services applications may require only the transaction processing capabilities inherent in the underlying execution environment, such as those provided by application servers and databases. Others may require multiple Web service invocations to be grouped into a larger transactional unit, including a transactional context within SOAP headers so that the transaction can be coordinated across multiple execution environments.

Web services transaction specifications extend the concept of the transaction coordinator, adapt the familiar two-phase commit protocol for Web services, and define new extended transaction protocols for more loosely coupled Web services and orchestration flows. Transaction coordinators currently exist in most execution environments, including J2EE, the .NET Framework, and

CORBA. Web services specifications define extensions for these (and other) coordinators for compensation-based protocols and long-running coordinator-coordinator protocols that bridge software domains.

Coordination is a general mechanism for determining a consistent, predefined outcome for composite Web service applications. The coordinator model includes two major phases: the acquisition phase in which Web services that are participating in the composite are enrolled with the coordinator for a specific protocol (such as two-phase commit, compensation, or business process) and a second phase in which the coordinator drives the agreed-upon protocol to completion following the end of the execution of the set of services in the composite. When a failure occurs, the coordinator is responsible for driving the recovery protocol (if any).

The specifications in this area include:

- **WS-Transactions family** from BEA, IBM, and Microsoft:
  - **WS-AtomicTransactions**—Defines volatile and durable variations of a standard two-phase commit protocol for short-lived executions.
  - **WS-BusinessActivity**—Defines variations on the idea of tentative commit and compensation-based undo protocols for longer-lived executions.
  - **WS-Coordination**—Defines the coordinator for the two pluggable protocols (and their variations).
- **WS-Composite Application Framework (WS-CAF)** from OASIS:
  - **WS-Context**—Defines a standalone context management system for generic context (that is, for non-transaction protocol contexts such as security, device and network IDs, or database and file IDs).
  - **WS-CoordinationFramework**—Defines a coordinator for the basic context specification and the pluggable transaction protocols in the WS-TransactionManagement specification.
  - **WS-TransactionManagement**—Defines three transaction protocols for the pluggable coordinator: ACID, long-running actions (compensation), and business process management.

Both sets of specifications are centered on an extended coordinator with pluggable transaction protocols. Both sets of specifications define atomic transactions (i.e., two-phase commit) and compensation-based transactions. WS-CAF breaks context management into a separate specification and adds a third transaction protocol specifically designed for business process management.

Orchestration

Web services can and eventually will be published for most software systems and applications within a given IT environment, and in fact across multiple organizations' IT environments. Rather than have Web services invoke each other using one or more of the message exchange patterns supported by SOAP and WSDL, an orchestration engine can be used to create more complex interaction patterns in long-running business process flows with exception handling, branching, and parallel execution. To accomplish this, the orchestration engine has to preserve context and provide correlation mechanisms across multiple services.

A Web service orchestration may also be published as a Web service, providing an interface that encapsulates a sequence of other Web services. Using the combination of MEPs and orchestration mechanisms, entire application suites can be built up out of Web services at multiple levels of encapsulation, from those that encapsulate a single software module to those that encapsulate a complex flow of other Web services.

The industry has reached a consensus around a single orchestration specification: the OASIS Web Services Business Process Execution Language (WS-BPEL). WS-BPEL assumes that Web services are defined using WSDL and policy assertions that identify any extended features.

Typically, a flow is initiated by the arrival of an XML document, and so the document-oriented Web services style tends to be used for modeling the entry point to a flow. Parts of the document are typically extracted and operated upon by the individual tasks in the flow, such as checking on the inventory availability for each line item from a different supplier, meaning the steps in the flow may be implemented using a combination of request/response and document-oriented Web services.

The WS-BPEL specification differs from other extended specifications in that it defines an executable language compatible with various software systems that drive business process automation. Whereas most other Web services specifications are XML representations of existing distributed computing features and capabilities that extend SOAP headers, orchestration represents the requirement for composing Web services in a declarative manner.

The W3C's Web Services Choreography Definition Language (WS-CDL) is another specification in the general area of orchestration. Choreography is defined as establishing the formal relationship between two or more external trading partners. One difference between WS-CDL and WS-BPEL is that WS-CDL does not require Web services infrastructure at all of the endpoints being integrated.

Strategic Value of Orchestration

Some people say orchestration is where Web services gain their strategic value. Web services have intrinsic value because of the relative ease with which they allow developers to solve interoperability problems across disparate types of software. Web services orchestration can be used in a variety of ways ranging from creating composite services in a declarative (non-programmatic) manner to full-blown business process management. Using Web services orchestration for BPM is clearly more difficult because it requires a deep understanding of an enterprise's business processes. In any case, it is true that the orchestration layer is where everyone expects the solution to be found to the hard problems of data type and structure incompatibilities, semantic data matching, and correlating the results of multiple Web services. It will take a while to prove whether or not these problems can really be solved at the orchestration layer and whether or not automated business process management is something companies really want or need at either the departmental or enterprise level.

Summary

So, we can see that the Web services standards, both the core and extended specifications, contribute significantly to the ability to create and maintain service-oriented architectures on which to build new enterprise applications. These applications are often called composite applications because they work through a combination of multiple services.

We've seen that SOA is not an end in itself but a preparation for a longer journey. It's a set of maps and directions to follow that lead to a better IT environment. It's a blueprint for an infrastructure that aligns IT with business, saves money through reuse of assets, rapid application development, and multi-channel access.

Web services have had an initial success with the core standards and are now on to the next step in the journey, which is to define extended features and functions that will support more and more kinds of applications.

Service orientation provides a different perspective and way of thinking than object orientation. It's as significant a change as going from procedure-oriented computing to object-oriented computing. Services tend toward complementary rather than replacement technology, and are best suited for interoperability across arbitrary execution environments, including object oriented, procedure oriented, and other systems.