CHAPTER 1

Introduction

Service-Oriented Architecture (SOA) is a way of organizing software so that companies can respond quickly to the changing requirements of the marketplace. The technology is based on *services*, which are customized units of software that run in a network.

A service

- handles a business process such as calculating an insurance quote or distributing email, or handles a relatively technical task such as accessing a database, or provides business data and the technical details needed to construct a graphical interface
- can access another service and, with the appropriate runtime technology, can access a traditional program and respond to different kinds of requesters — for example, to Web applications
- is relatively independent of other software so that changes to a requester require few or no changes to the service, while changes to the internal logic of a service require few or no changes to the requester

The relative independence of the service and other software is called *loose coupling*. The flexibility offered by loose coupling protects your company from excessive costs when business or technical requirements change.

A service can handle interactions within your company, as well as between your company and its suppliers, partners, and customers. The location of service

requesters can extend worldwide, depending on security issues and on the runtime software used to access a particular service.

In most cases, the requesting code has no details on the service location. Like the requester, the service can be almost anywhere. The location is set when the network is configured, and changes to the location are sometimes possible at network run time.

SOA implies a style of development, with concern for the business as a whole and with an increased focus on modularity and reuse. SOA isn't only for new code, though. Migration of existing applications is especially appropriate in the following cases:

- The applications are monolithic, combining the logic of user interface, business processing, and data access, with update of one kind of logic requiring your company to test multiple kinds of behavior.
- The applications are hard to understand — first, because the logic is monolithic, but second, because logic was repeatedly patched rather than rewritten as requirements changed. Updates take extra time as developers try to decipher the logic, and as the complexity grows, additional errors accompany updates.
- The application inventory has duplicate logic. Requests for change are unnecessarily disruptive, requiring changes in several places.

From the point of view of a business developer, a change to SOA is a change in emphasis, and many aspects of the job are unaffected. Consider the task of function invocation, for example. When you invoke a function, you aren't concerned with the internal logic of the invoked code or with how the function receives arguments or returns a value. Similarly, when you code a service request, you care only about the syntax for requesting the service. At best, service requests are as easy as function invocations.

Open Standards

In many industries, companies adhere to standards that allow for greater prosperity than would be possible if each company followed its own proprietary rules. Standards in housing construction, for example, ensure that manufacturers of pipes

can benefit from economies of scale in pursuit of a larger market than would be available in the absence of industry-wide standards.

The primary benefit of SOA standards is that they make services *interoperable*, which means that services can communicate with one another, even if each implementation is written in a different computer language or is accessed by way of a different *transport protocol* (software that oversees the runtime transmission of data).

Standards also ensure that an SOA runtime product can support Quality of Service features, as described in Chapter 2.

SOA standards are *open* in the sense that any software manufacturer has the right to use those standards when developing an SOA-related product. In addition, the process of creating and revising the standards is based on a political process that is more or less democratic. Any interested party has the right to participate in all meetings that lead to decisions about a standard.

Each company that works on an open standard seeks a text that matches the company's marketplace strengths. The competition among those companies is one reason for the long delay in making a standard final.

Several major organizations oversee development of open standards for SOA:

- Open Grid Forum (*http://www.ogf.org*)
- Organization for the Advancement of Structured Information Standards (OASIS; *http://www.oasis-open.org*)
- Web Services Interoperability Organization (WS-I; *http://www.ws-i.org*)
- World Wide Web Consortium (W3C; *http://www.w3.org*)

Later chapters give you practical insight into standards that are in effect or under consideration, and Appendix A describes several others.

Open standards are distinct from *open source*, which is source code that you can learn from and use in your own projects, with certain legal restrictions. Open-source implementations of Service Component Architecture (SCA) and Service

Data Objects (SDO), for example, are being developed in the Tuscany incubator project of the Apache Software Foundation. For details and code, see the following Web sites: *http://incubator.apache.org/tuscany* and *http://www.apache.org*.

Structure of a Service-Oriented Application

A *service-oriented application* is an application composed largely of services. Often, the invoked services are in a hierarchy, as Figure 1.1 illustrates.

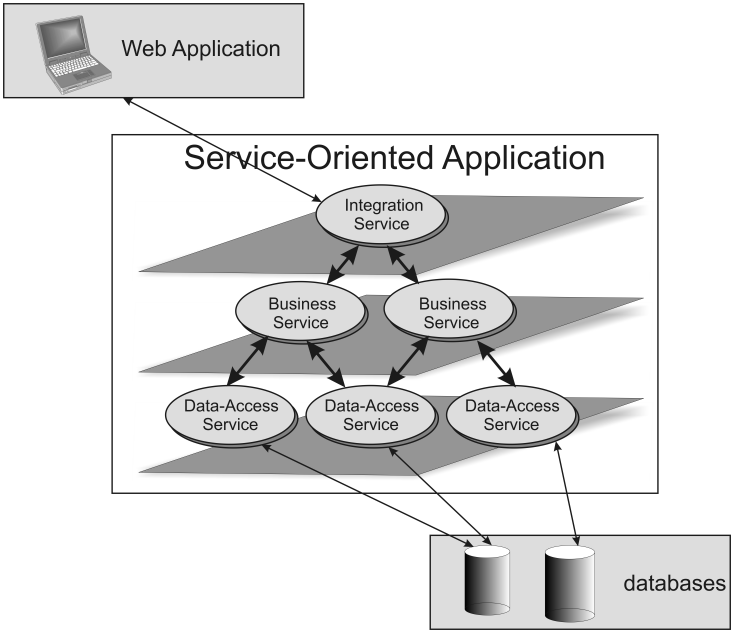

Figure 1.1: Service-oriented application

The topmost level contains one or more *integration services*, each of which controls a flow of activities such as processing an applicant's request for insurance coverage. Each integration service invokes one or more business services.

The second level is composed of services that each fulfill a relatively low-level business task. For example, an integration service might invoke such *business services* to verify the details provided by an insurance-policy applicant. If the business services return values that are judged to mean "issue a policy," the integration service invokes yet another business service, which calculates a quote and returns the quote to the software (for example, a Web application) that invoked the service-oriented application.

The third level consists of *data-access services*, each of which handles the relatively technical task of reading from and writing to data-storage areas such as databases and message queues. A data-access service is most often invoked from the business layer.

Great complexity is possible. Some integration services, for example, provide different operations to different requesters, and some invoke other integration services and are said to be *composed* of those services. Many applications, however, fulfill the three-level model described here.

Web and Binary-Exchange Services

This book also classifies services by the format of the data exchanged between a service and its requesters. A *Web service* exchanges data in a format based on Extensible Markup Language (XML). The W3C Web site suggests that the use of XML for data transfer is the only defining characteristic of a Web service. In many cases, the following description also applies:

- Details on the data are described in Web Services Description Language (WSDL).
- The format of the transmitted data is Simple Object Access Protocol (SOAP).
- The transport protocol is Hypertext Transfer Protocol (HTTP), the primary mechanism for exchanging program data over the Internet or a corporate intranet.

In contrast to a Web service, a *binary-exchange service* exchanges data in a format associated with a particular computer language or a specific vendor. Although a service written with Enterprise Generation Language (EGL) can be deployed as a Web service, for example, the logic also can be deployed as a service that exchanges binary data.

The use of binary-exchange services provides several benefits:

- allows a faster runtime response than is possible with Web services
- avoids the need to maintain WSDL definitions and related files
- avoids the need to learn the Web-service technologies

The cost, however, is reduced accessibility. A binary-exchange service is directly accessible only to software that transmits data in the binary format expected by the service.

Business Implications

SOA has several important implications for business. First, to the extent that loose coupling is in effect, changes made to the service logic won't force significant changes to requesters of that software. When each component is a relatively stand-alone unit, your company can respond to business or technological changes more quickly and with less expense and confusion. At best, a service can even be re-deployed to another machine without changing logic or recompiling the code. A further implication is that your company can develop services for different platforms and with different implementation languages, letting the organization use the best available technologies.

In general, a company's ability to respond quickly and well to change is known as *agility*. The main promise of service-oriented architecture is that a well-crafted SOA will increase agility over time.

SOA also has an important effect on how people work together. Aside from the most technical services, a well-written service is *coarse-grained*, meaning that the area of concern is broad enough so business people can understand the purpose of the service even if they know little about software. To the extent that a collection of coarse-grained services handles your company's business procedures, the firm's business analysts and software professionals can share information knowledgeably, can include end users in early deliberations about the purpose and scope of each service, and can understand all the implications of changing a business procedure. Ease of human communication is an important benefit of SOA and suggests that the architecture will become the primary organizing principle for business processing.

Last, well-designed services are more likely to be reusable. Your company benefits from reuse in at least two ways: first, by avoiding the expense of developing new software, and second, by increasing the reliability of the software inventory over time. The firm can do less extensive testing if an existing service is placed in a new application, in comparison to the testing required to deploy software that was written from scratch.

Criteria for SOA Implementation

A company is most likely to develop an SOA if the firm anticipates substantial change, has problems with existing applications or application access, and is willing to make the initial investment in analysis and planning.

A company is most likely to embrace an SOA based on Web services if it wants its software to be accessed by partners and customers — specifically, partners and customers that lack the binary-exchange solutions the company might otherwise favor. A Web service implementation is also the approach of choice for companies that want to depend less on particular software vendors.

Migration of Existing Applications

A company can develop an SOA to increase the rationality of its existing applications. A migration can be costly, though, in part because the design team must complete an analysis that reflects a concern for the business as a whole. A business-wide focus means that the team is better able to isolate services for use in multiple applications, including applications that are likely to arise in response to future requirements. The need is for knowledge and vision, so that interaction with business people can reduce the amount of duplicate logic in a company's software and can increase the ease of future updates.

Although planning for service development is essential, the actual development can occur in stages, with the cost of work spread over time and over several projects. As a start, a company might convert code that has strategic value, is accessed by different systems, or is likely to change in any case.

An incremental migration lets the company learn from experience. Decision-makers may begin a migration to an SOA that's based on Web services, for example, and then turn to a binary-exchange solution. The company is likely to benefit from whatever design was accomplished in the initial phase. It can even benefit from completed Web services because in most cases, code that depends on binary-exchange technology can access Web services.

Reasons to Reject Web Services

A company may require that its software respond more quickly than is possible with Web services. In this case, the company can use a binary-exchange technology and still gain the benefits of SOA. For example, if COBOL programs

access one another, a firm may want to avoid the runtime overhead that comes from converting data into XML and back to native COBOL.

Similarly, if a subsystem requires hardware or software from specific vendors, a company may consider using binary-exchange services that continue the firm's reliance on these vendors. A cost of this strategy is that the company is vulnerable to the vendors' pricing and customer response.

Last, a company may avoid developing Web or binary-exchange services to replace software that isn't expected to require a lot of change. This consideration applies when applications have fulfilled their mission for years and an appropriate statement is, "If it ain't broke, don't fix it."

Presentation Services

An important variation on the themes addressed in this book is the *presentation service*, which provides business data along with a stream of technical detail for constructing a graphical interface. The interface in turn lets the user interact with the data and access remote services and other software.

Different kinds of presentation services are in use. Although a comprehensive review is outside our scope, let us introduce a kind of presentation service that is widely used.

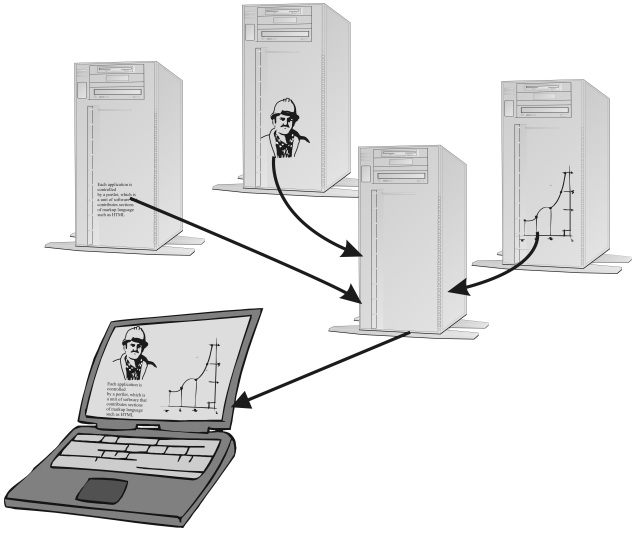

Figure 1.2: Portal technology

As Figure 1.2 illustrates, a *portal* is software that resides on a server and coordinates different interfaces, each affecting a different area of a Web page. From the perspective of a user, the output of a portal can provide a variety of news, business interaction, and entertainment, although an employee who accesses a company's internal portal is working primarily with applications that support the company's business.

Each application is controlled by a *portlet*, which is a unit of software that contributes sections of markup language such as Hypertext Markup Language (HTML). The portal collects those sections and submits them to devices such as Web browsers and personal digital assistants.

Portal technology empowers the user, who can select a subset of applications and may be able to customize the runtime behavior of individual portlets. Portals can even retain user preferences so that the experience is individualized as soon as a user logs on.

Traditionally, the portlets resided on the same server as the portal. In a natural extension of the technology, the portal requests data from a set of remote portlets, each of which acts as a presentation service. The standard that guides the interaction between portals and remote portlets is Web Services for Remote Portlets, as referenced in Appendix A.

SOA Runtime Products

Software vendors are now creating SOA runtime products that oversee a network of services. The general direction of that work is to allow a programmer, business analyst, or network coordinator to change product-configuration settings that affect (for example) the following issues:

- how security is handled
- which of several identical services at different locations is accessed by a requester
- whether a set of services is invoked in response to a requester's invocation of a single service
- what log information is collected

An SOA runtime product might allow the configuration of *intermediaries*, which are processing centers that do administrative or technical tasks during the transmission of data between a requester and a service. An intermediary might reroute messages in response to network traffic or business priorities; might reformat messages because the requester uses a transport protocol different from the one used by the service; or might provide security — for example, by authenticating requesters or by shielding the service from a flood of messages.

In the future, a subset of runtime products will incorporate guidelines from Service Component Architecture, an emerging open standard that permits flexibility at development, configuration, and run time.