12

CHAPTER TWELVE

Auditing Service-Oriented Architectures and Record Management Processes

ANY PROFESSIONAL WHO HAS been working with information technology (IT) hardware and software for more than a few years knows that IT technologies and techniques are always changing and evolving. What was a hot new concept just a few years ago sometimes disappears to be replaced by something new and different. In other cases, concepts that once seemed too advanced or even strange evolve into normal accepted practices. Client-server computer system configurations are an example of the latter. What was once a new and advanced concept in the 1990s is now a standard 1 IT process. Managing IT applications through service-oriented architecture (SOA) is another relatively new concept that will soon become part of the standard language of IT.

SOA is an IT systems approach where an application's business logic or individual functions are modularized and presented as services for consumer/client applications. A key concept is that the actual IT services provided are loosely coupled and independent of the actual application implementation. As a result, developers or system integrators can build applications by composing one or more services without knowing the underlying implementations of those services. For example, a service can be implemented in an advanced development language such as what is called dot-NET or Java, and the application consuming the service can be on a different platform or language.

This chapter introduces SOA concepts for the IT auditor and discusses internal control and IT auditor issues surrounding the development and operations of IT applications using this technology. SOA is not yet that much of an IT audit professional hot-button issue, with the Web site of the Information Systems Audit and Control Association (ISACA) only offering one other author's book on the subject but the current Web site of the Institute of Internal

Auditors (IIA) offers none, although it is a strongly evolving concept for IT applications development and implementations. IT auditors should have a general understanding of the controls and concepts surrounding SOA implementations.

The chapter also reviews the importance of effective records management systems in today's enterprises and IT environments. Today, almost all business records are created and most live their entire lives electronically. Failure to manage electronic records and physical records in accordance with established records management principles is to turn a blind eye to potential risk. This chapter also concludes with an overview of records management issues from an internal controls and IT audit perspective.

SERVICE-ORIENTED COMPUTING AND SERVICE-DRIVEN APPLICATIONS

Hardware and software vendors, as well as high-level software application developers, often use similar but slightly different terms to describe IT system concepts. We may encounter such expressions as service-oriented architectures, service-oriented design, and object-oriented design when hearing about the attributes of some new IT application. These expressions sound alike and generally reflect similar approaches to the manner in which IT applications are built and launched in our Web-oriented world today. In this chapter, we generally use the expression service-oriented architecture or SOA to describe this concept even though computer science purists may differ on some terminology details.

SOA is an IT architectural style whose goal is to achieve loose coupling among interacting software agents. It is an approach that allows interoperability between different IT systems and programming languages, providing the basis for integration between applications on different platforms through messages across network communication links. The software is built following both common and industry-specific components that are granular and modular. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Software agents play both provider and consumer roles on behalf of their owners.

The Web is filled with many often complex and sometimes differing definitions of SOA, and the definition here may be too abstract, but an IT auditor should not consider SOA as a difficult concept. Music CDs and CD players model SOA concepts. If you want to play a CD, for instance, you put your CD into a CD player, and the player broadcasts it for you. The CD player offers a CD music-playing service. When the CD ends, you can replace one music CD for another. You can also play the same music CD on a portable player or on a unit in your automobile. Both offer the same CD-playing service, but the quality of service may be different. The results of the CD music-playing service may result in a change of state for the listening consumer, perhaps a mood change from depressed to happy. We are the service client, the CD player is the service provider, and our music library of CDs acts as service broker.

Just as the CD player gives us prerecorded music, we hire someone else to provide services or do the work for us in other areas because they have the available resources and are experts. Consuming a service is usually cheaper and more effective than doing

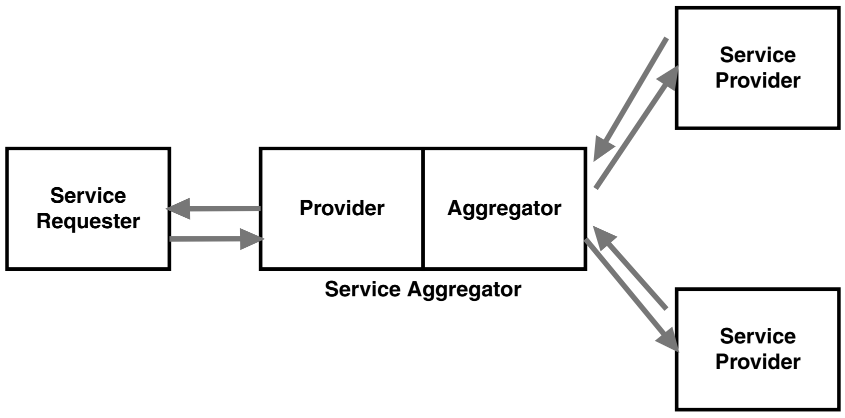

EXHIBIT 12.1 Role of the Service Aggregator

the work ourselves. Most of us realize that we are not smart enough to be experts in everything. The same rule applies to building software systems.

This concept of using the music service of a personal CD player from a library of music CDs leads to SOA concepts. The service is the basic building block of SOA; it is a way of accessing repeatable business capabilities, organized as simple software interfaces available for all providers and consumers. Exhibit 12.1 shows this basic concept of an SOA configuration with a service aggregator. The service requestor requests services from a provider who is an aggregator with a catalog of available service offerings, who will pull the request from any of multiple service providers, who return the service to the requestor and then to the provider.

If an SOA is to be effective, we need a clear understanding of the term *service*. A service is a function that is well defined, self-contained, and does not depend on the context or state of other services. Services are the building blocks of SOA and can be snapped together to make other services or assembled in sequences to make processes.

In SOA, services are usually organized in what is called a *registry*, where separate services can be snapped together to create composite applications and then fitted together into what is called an SOA blueprint. IT auditors generally encounter SOA efforts when performing an overall review of enterprise general controls, as discussed in Chapter 6, or reviewing specific application controls. While our objective is not to make an IT auditor into a computer scientist or software developer, several important SOA concepts that an IT auditor may encounter when discussing SOA processes are discussed next.

- **SOA component granularity** describes the size of the components that make up a system. An SOA implementation would try to achieve larger, or coarsely grained, components known as business services. These usually are be built out of smaller, or fine-grained, components as well as preexisting technical services. This set of components matters because chunks of larger granularity make SOA services easier for an enterprise to understand, reuse, and manage.
- **Interface versus implementation concepts** separate *what* a service does from *how* it does it. This issue matters because it simplifies the concept that the business

user's view of SOA should focus on what the service will do rather than the details of how the technology works under the hood (to use an automobile analogy).

- **Contracts** define the obligations between the service provider and service consumer or requestor. Obligations can include service expectations such as availability, reliability, and key performance indicators, as well as service cost and support. This definition of obligations matters because it helps business users make rational decisions about which services they can rely on. These concepts are similar to the service-level agreements (SLAs) discussed in Chapter 7.
- **Loose coupling** is a way of designing services that are more flexible and less dependent on each other. This method helps services to snap together or couple and recombine. An SOA environment with loose coupling matters because it is faster to assemble business solutions out of premade blocks than it is to write every new business functions from scratch.

Adopting SOA in an enterprise is not a one-application-at-a-time process but describes an overall architecture where all IT processes are organized together and connected. Services provided with Web applications where a user just double-clicks to pull up an HTML reference are an example of an SOA concept. Exhibit 12.2 illustrates a hypothetical enterprise-wide SOA configuration.

At the lower lever of the exhibit are enterprise data sources and other multiplatform systems. All elements here would be defined as services such that a custom application, for example, might be built of a combination of mainframe/legacy and Web application

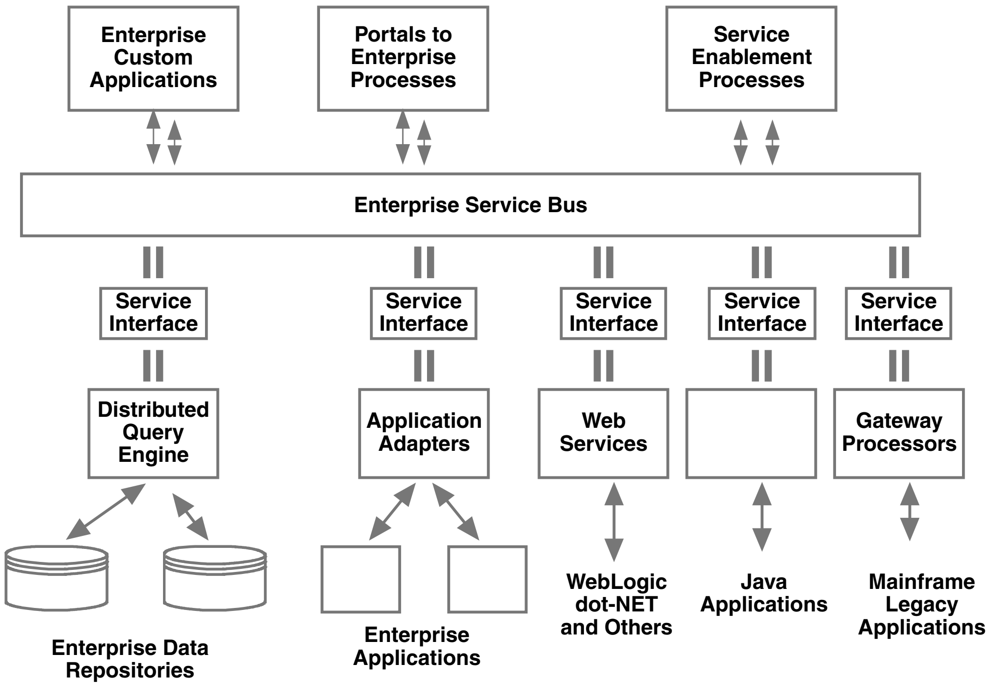

EXHIBIT 12.2 Enterprise-Wide SOA Example Configuration

components and assembled with the requests through a service bus to the proper adaptor and application. When any of those application services need an element of data—another service—a request would go up to the communication bus and back to the appropriate data source. Such an overall application would be built using some of the newer application software development tools, such as Microsoft's dot-NET[1] or Oracle's BEA WebLogic[2] and would be configured and orchestrated in clearly specified processes.

An IT auditor does not find a full SOA environment in most enterprises. Rather, an enterprise IT function, with management support, typically implements their SOA environment on a step-by-step basis following an overall blueprint to build that model. With the growing use of Web services surrounding most IT environments and software sold and delivered as individual program components rather than as full applications, IT auditors can expect to see partial and even full SOA implementations going forward.

SOA Internal Control Issues and Risks

An enterprise does not just move to an SOA IT environment by purchasing a SOA featured software package, training the IT staff, and then assuming they are off and running. SOA is more than just a new way of organizing existing IT systems. Detailed planning and new IT policies and procedures are required to move to that new environment. Even more important, such a transition requires a gradual and very structured implementation where IT processes are brought into an SOA environment in a controlled manner. In some cases, the advantages of SOA operations will be obvious to senior managers and others. For example, an enterprise that has moved a large segment of its applications to a Java or dot-NET Web-based environment often can easily justify the advantages of moving its legacy applications and other processes to an SOA environment.

IT auditors can play a strong role in reviewing processes and helping their enterprise management and IT functions transition to SOA processes. Part of doing this requires that an IT auditor understand the preimplementation review processes, as discussed in Chapter 10, be comfortable with auditing Chapter 7's IT infrastructure controls in an SOA environment, and understand good project management controls, as discussed in Chapter 16. An IT auditor needs to understand the somewhat unique characteristics of SOA and its control issues and risks. This chapter briefly discusses implementing effective SOA processes from an IT audit perspective. The next sections discuss the importance of building a planning blueprint for an SOA implementation, cleaning up existing applications and processes in advance, establishing appropriate SOA policies and procedures, and testing and implementing SOA in an effective manner.

Planning and Building an SOA Implementation Blueprint

The third letter of SOA stands for architecture, and the building of an effective architecture is an essential early step to launch SOA in an enterprise. Even if the enterprise chief information officer (CIO) has been exposed to SOA concepts or if the enterprise has implemented an SOA-like single application that gained enthusiastic comments and praise, an enterprise still needs to build an overall framework to launch any SOA process implementation.

The key to an effective SOA is to identify an existing plan or build this service set of procedures using blocks that will define the SOA similar to a series of LEGO® building blocks.[3] The blocks of these popular children's toys can be interconnected in a variety of elaborate structures and then disassembled to build a different structure. However, SOA services are more complex than LEGO blocks that are built in cubical and triangular shapes; SOA services are much broader and generally may have many different dimensions.

A key process here is to define the various service elements and the service stakeholders, including the flow of activities and decision points between the stakeholders involved in the process, in this way:

- **Business owner.** The business owner provides the application requirements for a new business capability, solution, or process to be implemented. A business owner or supporting IT staff members should develop process models to make any understanding of IT implementation service requirements easier. The business owner also needs to define nonfunctional requirements, such as expected quality of the service, for the capability, solution, or process.
- **SOA architect.** In order to launch an SOA initiative, an enterprise needs to designate a skilled SOA architect to analyze the business requirements and break them into service and process designs. The SOA architect may decide, for example, to reuse existing components rather than create new ones. When such an architect proposes a new or changed services or process implementations, this SOA architect also should deliver design specifications in terms of state diagrams, process models, and interface designs. The SOA architect formalizes the nonfunctional requirements of the component to be implemented, including its availability, security, and performance requirements.
- **Application developer.** A developer implements components based on the design specifications delivered by the SOA architect and creates test plans based on these specifications. To aid in the convergence of technology and methodology, an application developer should use parts generated by the SOA architect for implementation, including code generation or model refinements.
- **Quality manager.** A team member designated as the quality manager uses the input provided by the business owner, architect, and developer to review the correctness of the service or process that was implemented. This topic is discussed in more detail in Chapter 31. A quality manager then uses the test plans provided by the developer to execute a solution test in a staging environment and validates quality metrics, side effects, and nonfunctional characteristics.

Once a key team had been identified, the enterprise should develop an SOA blueprint that indicates a target state or a complete picture of what the SOA implementation should look like when it is completed. The SOA blueprint should outline a complete list of these necessary components:

- Business services
- Service description requirements
- Service performance metrics

Improve Business Visibility. Integrate systems and aggregate data for a consistent, accurate views of customers, including:

- Up-to-the-minute information for improved customer service
- Cross-enterprise information for targeted 1:1 activities
- Consistent, accurate, and more comprehensive information for better decision making

Achieve Business Flexibility. Create an integrated, agile software infrastructure for quickly responding to business needs:

- Provide rapid delivery of new business capabilities.
- Reduce impact of business and technology changes.
- Protect investments while creating new functionality.

Gain Business Efficiency. Streamline, automate, and enable the better tracking and visibility of enterprise business processes:

- Securely share business processes inside and outside the IT systems firewall.
- Bridge silos of data to better ensure data integrity.
- Proactively manage business decisions with key performance indicators from other sources.

EXHIBIT 12.3 Service-Oriented Architecture Key Benefits

- Interoperability standards
- Data schemas
- Policies
- Service discovery and classification requirements

Such a blueprint provides an overall outline of where enterprise and general IT management want to go with SOA. Of course, an enterprise should not begin to implement or even approve an SOA strategy unless there is a good understanding of the benefits to be achieved from the approach. Exhibit 12.3 describes some of the key benefits that an enterprise should realize from moving to an SOA environment.

Transitioning Current Applications and Processes to SOA

The transition of existing processes or applications to an SOA environment, whether older legacy systems or more recent Web-based applications, contains some of the same challenges older professionals went through when we approached the Y2K problem in the late 1990s. Although some may have now forgotten, many COBOL-based programs had system calendar date fields in a yymmdd format and many program decisions based on taking action by sorting on dates. The year 2000 caused those date-based program decisions to be recorded as 0000. The Y2K concern was that many systems would fail because of bad, out-of-sequence program dates, and at the time, many older applications were poorly documented. IT functions and IT auditors went through major efforts to get ready for Y2K, and we face some of these same concerns when reviewing and cleaning up existing processes to transition to SOA operations.

The concern with a transition to SOA is not undocumented yymmdd-format COBOL dates; rather, the typical IT function has layers of IT applications and programs, redundant and often mutually inaccessible systems, and much jumbled point-to-point