An Integrated Service Model Approach for Enabling SOA

Liang-Jie Zhang, *IBM T.J. Watson Research Center*
Jia Zhang, *Northern Illinois University*

To effectively align business and IT using a service-oriented architecture (SOA), a proposed integrated service model divides service construction into loosely coupled perspectives according to a service's business logic, interface, and implementation.

Although the service-oriented architecture (SOA) has become a prominent strategic model in the modern business world, SOA solution design and development is often still ad hoc, rather than part of a systematic implementation.[1] Companies lack a business-aligned service model to guide and facilitate the design, development, and management life cycle of highly reusable services and service components. Existing approaches, such as the SOA triangular model[2] and various hybrid system integrating methods,[3–5] provide high-level guidance instead of detailed instructions.

Over the past 50 years, researchers have established a wealth of architectural models that guide software application design and development. However, some significant challenges arise when we attempt to directly apply these architectural models to a SOA-based solution design process because of the unique features of SOA requirements. For example, a SOA solution requires that a system center around reusable services instead of specific software components, and a SOA solution must be able to adapt to changing business requirements. Existing software engineering architectural models insufficiently address these SOA-related needs.

Based on the Service-Oriented Reference Architecture (S3),[6] we propose an integrated service model (ISM) that decouples three perspectives of a service: business logic, interface, and implementation. By enhancing the existing SOA triangular model, our ISM supports increased flexibility, extensibility, and adaptability of services. Based on industry best practices, we also introduce an ISM-based methodology to guide service decomposition and composition. (For the purposes of our discussion, a *service* implies a broader domain than just a Web service. In fact, a service can be implemented using various technologies, which is an important way of realizing solution architectures for services computing.[1])

 Published by the IEEE Computer Society

Separating the service model into logical layers hides the complexity of leveraging existing applications to deliver services. Such an approach can help companies leverage existing applications as reusable assets for constructing new business services and coordinating multiple business processes. Our proposed logical service model can guide and facilitate the rapid creation of business-IT-integrated services at the enterprise level.

Motivation

To date, the triangular conceptual model is the most well-known and widely accepted SOA-based architectural model;[2] it provides the backbone for creating, registering, and discovering interface-exposed services. As Figure 1 shows, a SOA's three roles over a service include acting as

- a *service provider* that offers services by publishing them to a service registry,
- a *service registry* that helps service requestors find service providers for proper services by organizing registered services and providing search, and
- a *service requestor* that invokes services by querying a service registry and then binding to the service provider to invoke those services.

According to this SOA triangular model, an existing software application (whether it's a packaged application, customer application, or legacy system) can be wrapped with a service-compliant interface and then published as a Web service into a service registry. The encapsulated application in a Web service might range from a single application component to a comprehensive large-scale software product containing many components as well as other software products. Meanwhile, this model allows a new application, developed from scratch, to be published as a Web service. Based on this service model, a wealth of wrapping and development platforms have been developed, such as IBM's WebSphere (www.ibm.com/websphere), BEA's AquaLogic (www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/aqualogic), and open source software products such as Sun's JBoss (which is available at www.jboss.org).

This model's major drawbacks stem from its simplicity. It doesn't systematically identify and address service-handling-related issues, suchas

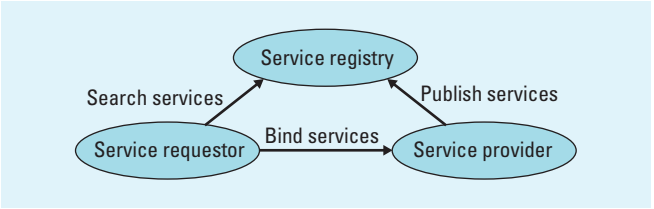

Figure 1. Service-oriented architecture (SOA) triangular model. The service provider, registry, and requestor roles are accompanied by their behaviors and responsibilities.

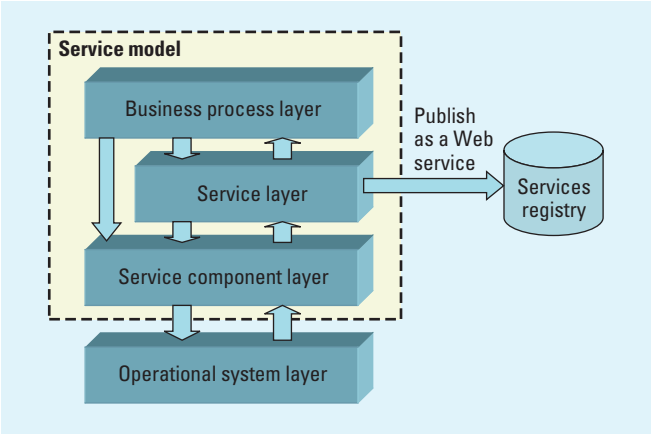

Figure 2. Multigranular integrated service model. The three interrelated, loosely coupled logical layers are realized in the Service-Oriented Reference Architecture.

decomposition, aggregation, transformation, and invocation. Consequently, solution architects must design component models for each service from scratch based on their personal experiences. In addition, the model doesn't provide architecture-level support for configuring and reconfiguring services and service components. Furthermore, a solution based on the current triangular service model developed using this approach might suffer from low reusability and might not provide adaptability for runtime evolutionary changes.

Many researchers and practitioners have explored methodologies for engineering services design and development and proposed various hybrid system integration methods.[3–5] However, such efforts still only provide high-level direction instead of detailed normative guidance.

Integrated Service Model

Figure 2 shows our proposed ISM. By grouping interaction patterns and configuration management according to semantic coherence, we've identified three interrelated but loosely coupled

logical layers to align with the high-level objectives defined in S3:[6] the *business process*, *service*, and *service component* layers. (IBM proposed S3 to guide IT architects in designing the overall architecture of an enterprise-level SOA solution.)

Our model decouples three perspectives of a service: the business process layer handles any business-logic-related service composition and decomposition; the service component layer handles any implementation-related service integration and invocation; and the service layer handles all interface-related (Web Service Description Language-related) service publication, location, and aggregation.

The business process layer provides facilities to decompose a business process into conceptual services that fulfill business functions.

As Figure 2 shows, the service model is built on top of an operational system layer,[6] which represents the following:

- packaged applications typically provided by individual service vendors (ISVs);
- customer applications developed in-house; or
- legacy systems typically developed using traditional technologies.

Each of these applications could traditionally only be used for one purpose and serve one specific user or user group. With the aid of a SOA, we can make such applications available as a service with standard interfaces so that other services or applications can discover and reuse them.

Business Process Layer

The business process layer leverages the service layer to manage services in the context of business workflows. This layer performs 3D process-level service handling: top down, bottom up, and horizontal. The top-down direction provides facilities to map business requirements into tasks comprising activity flows, each being realized by existing business processes, services, and service components. The bottom-up direction provides facilities to quickly compose and choreograph existing business processes, services, and service components into new business processes to fulfill customer requirements. Finally, the horizontal direction provides service-oriented collaboration control between business processes, services, and service components.

The business process layer doesn't focus on individual business process representations, which can be fulfilled by dedicated workflow description languages such as Business Process Execution Language for Web Services (BPEL4WS). Rather, this layer focuses on building SOA solutions using business processes—for example, the layer might take 10 existing business processes and aggregate them into three big processes, while monitoring and managing the collaboration between them.

To decompose a business process, we first divide it into smaller tasks and then map each one into service clusters (or conceptual services) that actual Web services will realize in the service layer. In other words, the business process layer provides facilities to decompose a business process into conceptual services that fulfill business functions, or service clusters.

Service Layer

The service layer extends the triangular SOA model into a comprehensive logical layer that enables and facilitates service registration, decomposition, discovery, binding, aggregation, and service lifecycle management.

The service layer leverages the concept of a Web service cluster,[7] which is a collection (category) of Web services serving a common business function. These Web services can be published by different service providers and differentiated from one another by specific features—for example, we might consider a generic shipping service a conceptual service cluster. Many service providers (such as the United Postal Service [UPS], the US Postal Service [USPS], and Federal Express [FedEx]) might exist for the same shipping purpose. In addition, one service provider—say, UPS—might also provide various shipping services with different timeframes and guarantees. For example, a UPS service can provide overnight, second-day, two-day, three-day, five-day, and one-week deliveries. Our model considers all these types of shipping service implementations

as a conceptual shipping service cluster. As a best practice, we always categorize services in this layer into service clusters based on some business function, such as reporting or purchase-order-management services.

A business process only cares about the level of service clusters, instead of individual services, because a selected service might be unavailable at invocation time. In this case, the SOA shall replace it with another available service in the same service cluster, making the switch transparent to users. Each potential service is kept in the logical service layer.

The service layer performs both top-down and bottom-up service-level handling. In the top-down direction, the layer provides facilities to locate actual service interfaces for business processes; in the bottom-up direction, the layer provides facilities to expose service interfaces to the outside world (see the services registry in Figure 2).

The service layer handles the actual top-down mapping from business processes into real services. As we discussed earlier, our model decomposes a business process into service clusters. Then, for each service cluster identified, the service layer is responsible for

- finding an appropriate service provider,
- locating where the target service resides and accumulating other requirements such as access control, and
- binding to the target service interface.

From the bottom up, the service layer exposes Web service interfaces for service components. One service component can become available in different formats and service interfaces; Figure 3 shows one service component available in four service interfaces. In other words, one service component can implement multiple services defined in the service layer. Therefore, the number of services in the service layer can exceed the number of service components in the service component layer. In the service component layer, a wrapper that implements the interface defined by the service layer can use multiple service components.

The service layer can also perform some high-level service aggregation. Figure 3 illustrates two categories of services: an individual service refers to an atomic service that doesn't depend on other services, and a composite service depends on more than one individual service.

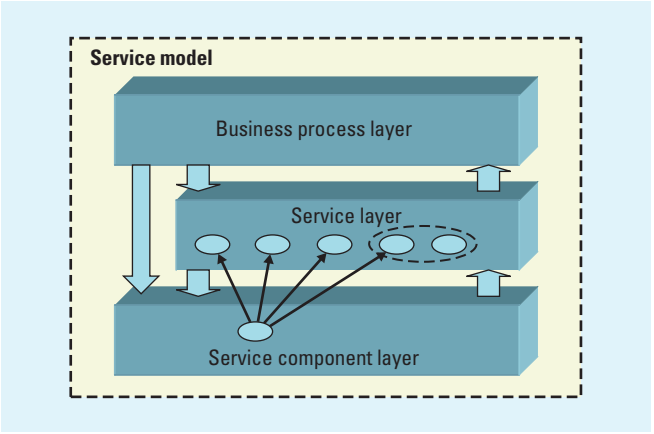

Figure 3. One-to-many relationship for a service component to multiple service interfaces. The number of services in the service layer can exceed the number of service components in the service component layer, so a single service component can implement multiple services. An individual oval indicates an individual service, and a dotted oval comprising several individual services indicates a composite service.

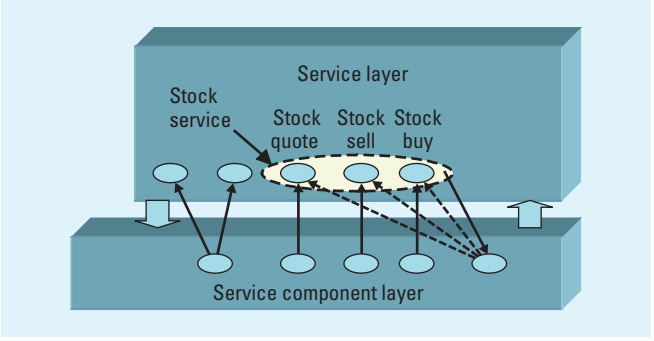

Figure 4. The composite service concept. In this example, a composite stock service consists of three individual services (quoting, selling, and buying).

Figure 4 illustrates the composite service concept using a composite stock service example. The three individual services include stock quoting, selling, and buying. Each service is wrapped by a corresponding service component in the service component layer and represents one stock-related activity—that is, quoting a stock price, selling a stock, or buying a stock. The stock service aggregates the three individual services into a composite service with a new service interface. This kind of service aggregation is based on interfaces (packaging) only, without business logic involved.

Figure 4 shows that no control flow exists between the three aggregated services. In fact, each

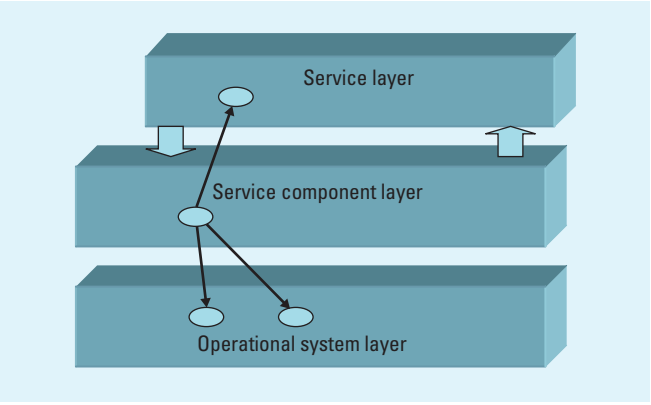

Figure 5. An individual service concept. Because only one service (interface) from the service layer is involved, this is an individual service even though it's eventually implemented by more than one legacy system.

of the three services could become a separate operation of the aggregated interface (such as a WSDL interface). Also, a composite service typically contains a special kind of service component, which we call a *technical service component*, in the service component layer to implement the interface aggregation. As Figure 4 shows, the composite stock service consists of a technical service component in the service component layer to realize or the three services (service interfaces) in the service layer.

A composite service must comprise more than one service (interface), or it would be an individual service. Figure 5 shows a service implemented by a service component in the service component layer, which is in turn implemented by two legacy systems in the operational system layer. Because only one service (interface) from the service layer is involved, we consider this an individual service.

Hence, a composite service must include a composite interface and an implementation through a technical service component, which in turn invokes or realizes more than one service in the service layer and/or another technical component, such as a legacy system.

We use the term "aggregation" here instead of "composition" as in the business process layer. On one hand, service composition refers to integrating services into a business process with business logic. With service composition, business flow exists between services, which can be represented using a business flow description language such as BPEL. On the other hand, service aggregation refers to turning services into individual operations in a new service interface without adding any business logic between them. Note that our model doesn't define business flows between the aggregated services. In short, service aggregation is merely a new way to release existing services.

In general, a given organization has two distinct groups of services: external and internal. The first group, also known as common business services, refers to a set of business-aligned services fulfilling an organization's enterprise processes and goals. These services can be tied back to business processes or exposed to other lines of business or to the outside world of partners and the SOA ecosystem. The second group contains those services that address IT integration and infrastructure problems. Typically, one organization doesn't necessarily apply the same rigor in identifying and exposing this type of service. Although internal services are designed to meet certain requirements, they might not directly represent an important aspect of the SOA value proposition from a visibility perspective.

Service Component Layer

The service component layer provides code containers that implement services dealing with how to actually implement services. A service component might rely on one or more packaged applications, customer applications, or legacy systems; it can also invoke services in the service layer or business processes in the business process layer to implement the method signatures defined in the service layer. For example, a service component can be implemented in a Java class, an Enterprise JavaBean (EJB), a .NET component, and so on, or it might include the implementations of multiple methods, with some methods exposed as services.

The service layer is in charge of binding to an actual service, but it doesn't handle invocation adaptation. The services component layer handles service invocation,[1] including input-method signature transformation and output-method signature adaptation.

Enhancing service invocation and automating it remain challenging. Web services can only be accessed through their service interfaces defined in the standard WSDL. Because the current form of WSDL specifications only exposes limited information for Web services interfaces, parameter adaptations and interpretations are

typically required prior to and after actual service invocations. A WSDL service method signature only defines the method name and the parameters' data types. This information is usually too generic and inadequate for a program to properly invoke the target Web service, and there's no semantic information available to help correctly construct input parameters.

Organizations can start with our ISM, customize it, and then apply it to develop reusable, flexible, and extensible SOA services for one or more business lines. This model is especially suitable for software architects responsible for designing software architectures for business-driven SOA solutions. They could quickly configure and customize this model into an architectural proposal for their customers based on specific business requirements. Moreover, users can directly deliver the customized service model to a corresponding development team as architecture templates and normative guidance for the actual solution development. Currently, our ISM has become an integral part of the IBM SOMA Modeling Environment (SOMA-ME),[8] which various industry projects are using as the core tool to conduct SOA solution services. IT

Acknowledgments

This major work of this research was conducted while Jia Zhang was an academic visitor at IBM T.J. Watson Research Center in 2006.

References

1. L.-J. Zhang, J. Zhang, and H. Cai, *Services Computing*, Springer & Tsinghua Univ. Press, 2007.
2. C. Ferris and J. Farrell, "What Are Web Services?" Comm. *ACM*, vol. 46, no. 6, 2003, pp. 31–35.
3. J. Hutchinson et al., "Migrating to SOAs by Way of Hybrid Systems," *IT Professional*, vol. 10, no. 1, 2008, pp. 34–42.
4. Y.L. Blevec et al., "Service-Oriented Computing: Bringing Business Systems to the Web," *IT Professional*, vol. 9, no. 3, 2007, pp. 19–24.
5. B. Benatallah et al., "Service Mosaic: A Model-Driven Framework for Web Services Life-Cycle Management," *IEEE Internet Computing*, vol. 10, no. 5, 2006, pp. 55–63.
6. A. Arsanjani et al., "S3: A Service-Oriented Reference Architecture," *IT Professional*, vol. 9, no. 3, 2007, pp. 10–17.
7. L.-J. Zhang and B. Li, "Requirements Driven Dynamic Services Composition for Web Services and Grid Solutions," *J. Grid Computing*, vol. 2, no. 2, 2004, pp. 121–140.
8. L.-J. Zhang et al., "SOMA-ME: A Platform for the Model-Driven Design of SOA Solutions," *IBM Systems J.*, vol. 47, no. 3, 2008, pp. 397–413.

Liang-Jie Zhang *is a research staff member and program manager of application architectures and realization at the IBM T.J. Watson Research Center. His technical interests include services computing, Internet media, and software engineering. Zhang has a PhD in pattern recognition and intelligent control from Tsinghua University. Contact him at zhanglj@ieee.org.*

Jia Zhang *is an assistant professor in the Department of Computer Science at Northern Illinois University. Her technical interests center around services computing. Zhang has a PhD in computer science from the University of Illinois, Chicago. She is a member of the IEEE. Contact her at jiazhang@cs.niu.edu.*