

Understanding the Service Lifecycle within a SOA: Run Time

by Quinton Wall
11/08/2006

Abstract

The ability to effectively manage the lifecycle of services is fundamental to achieving success within a SOA initiative. Design-time aspects of this type of management include such areas as service categorization, modeling methodology, and concepts related to building and composing services. This article focuses on the run-time aspects of the service lifecycle, which include publishing and provisioning the service, integrating the service into composite applications, deploying the service, monitoring and managing its usage, and evaluating the use of the service in a realistic setting, such as production. Refer to Understanding the Service Lifecycle within a SOA: Design Time for an introduction to this series and the design-time aspects indicated above.

Introduction

The Shared Service Lifecycle (SSLC) model shown in Figure 1 provides a consistent roadmap for discussions throughout this article. Where appropriate I dive into broader discussions to support the need for the run-time phases of the SSLC, and I provide best-practice advice on aspects such as service composition and handling change.

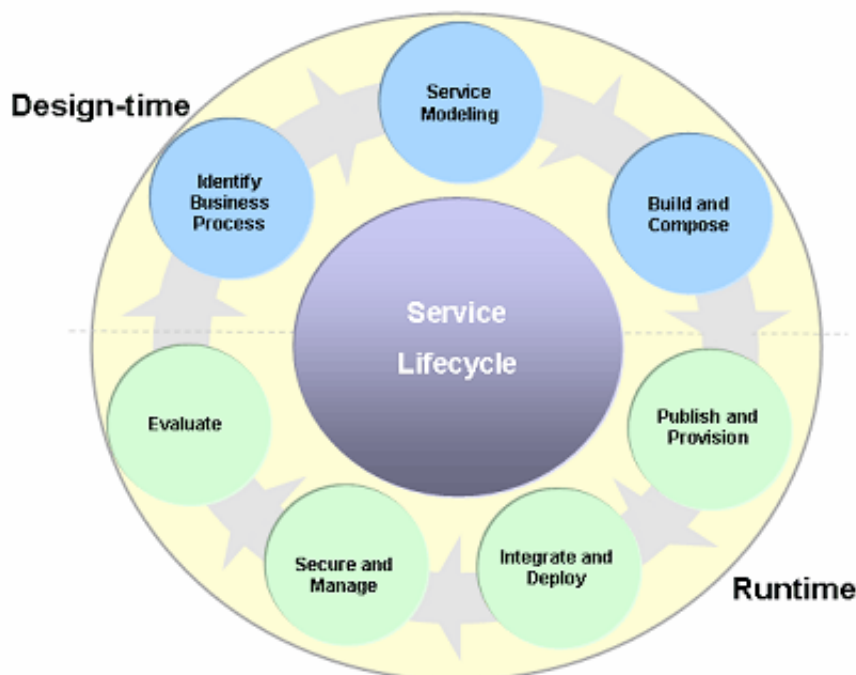


Figure 1: The Shared Service Lifecycle (SSLC)

Organizations undertaking SOA initiatives, big or small, often form groups tasked with service-enabling current or required business processes. These service engineering teams may be tasked with specific aspects or entire lifecycles associated with the SSLC. In the fictitious organization introduced in Part 1 of this article, the service engineering team, responsible for the entire SSLC, has modeled and built a number of services to support the organization's needs. The team must now expose these services as run-time offerings within the organization.

Before diving into discussions pertaining to the run-time aspects of the SSLC, let's look again at the fictitious organization that offers book and movies for sale over an ecommerce site. The service engineering team developed a catalog of needs to provide a roadmap to service creation, as depicted in Figure 2.

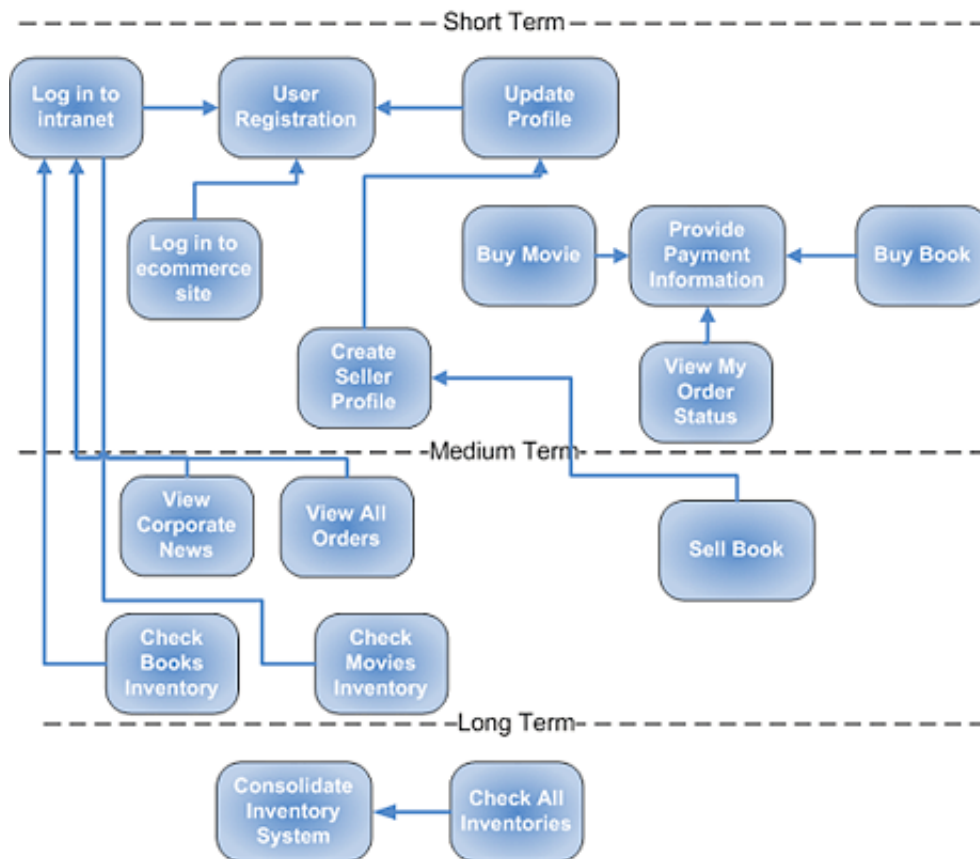


Figure 2: The catalog of needs

Using the catalog of needs, you can begin to identify dependencies and the value of planning for your SOA initiatives. Just as in the build phases of the SSLC, run-time aspects use this catalog to target effort; then, the effort required can be further defined and dependencies understood by examining how applications can be composed through layering of services. Once the engineering team has acquired an understanding of the organization's needs, it can leverage those services created in the design-time phase of the SSLC to focus on the run-time requirements of putting these services into production. The remainder of this article focuses on aspects of the SSLC and associated activities for successfully managing the run-time phase of the SOA Shared Service Lifecycle.

Composite Applications and Layering of Services

Earlier, I described the differences between traditional application development and SOA practices as a way of breaking out of the limitations of monolithic applications and shortening development/release/test cycles. Such agility can be obtained only by focusing on the notion of composite applications. By definition, composition is the creation of new functionality to meet business needs by assembling some combination of existing and newly created services. Within a mature SOA environment, complete business applications may be assembled through the use of existing services composed to quickly meet the business needs. To ensure flexibility in this provisioning process, the service engineering team must pay careful attention during design-time to avoid coding logic into service implementations. In addition, to ensure this agility is not lost at provisioning time, the run-time phases of the SSLC should focus on the need to ensure that service contracts, policies, and metadata do not inhibit future needs. One such approach to achieving this run-time flexibility is to design and provision services to define a number of layers that may form some logical or conceptual layering of the system.

Within a SOA initiative, composition is an integral part of achieving business flexibility through the ability to leverage existing assets in higher-order functions. My experience has shown that many organizations develop services at such a granular level that the proliferation of many small specific services are difficult to compose into broader logical services. By defining a layered approach to service definition and construction, the service engineering team is more likely to achieve the right mix of granular and course-grained services required to meet the business demands today and going forward.

To demonstrate the concepts of layered services, consider the e-commerce and corporate intranet example defined in the catalog of needs (see Figure 2 above), which identifies a long-term need to consolidate the organization's inventory systems and associated processes. The BEA SOA Domain Whitepaper (PDF) identifies a number of logical layers that may exist within your Service Orientated environment and form part of your SOA Reference Architecture. These layers—information and access, shared business services, composite services, and presentation services—are a great way of defining separation of responsibility. You can decompose these layers further by taking a more granular view of service needs. Such a perspective identifies the need to group services into the categories of physical, canonical, logical, and application services. Let's look at these in more detail.

Physical services

The Information and Access services may represent functions that retrieve data in its raw form. Looking at the catalog of needs, one of the needs is the ability to read orders. A physical service may be defined similar to that shown in Figure 3.

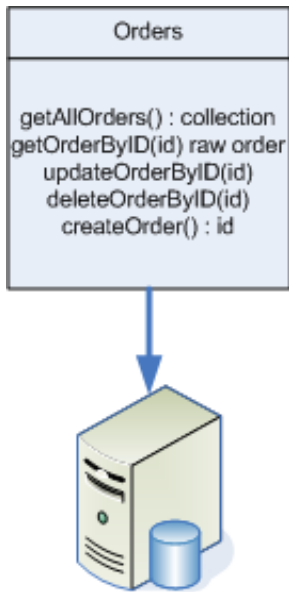


Figure 3: The Order Physical Service

The physical service defined above is likely to form part of the information and access layer of an organization's reference architecture. Such a service should support fine-grained CRUD (create, read, update, delete) operations on the specific data source(s).

Canonical services

Canonical services may define a standard view of information for the organization. Quite often, such a view may leverage industry-standard formats such as SWIFT or other standards specific to the industry your organization is related to. By establishing a common lingua franca, a clear producer layer begins to form. This standardized layer, independent of the physical services, can be leveraged as the genesis of shared services or composite applications.

Taking a look at the physical service defined above, you see that it has been modeled to return a raw representation of an order. The service engineering team must now implement a conceptual layer to return a standardized canonical model. At initial glance, you may not see the need for a canonical model of an order if the orders service returns everything you need. Taking another look at the catalog of needs developed previously by the service engineering team indicates that the fictitious organization has more than one inventory system for books and video, and there is a desire to consolidate these systems in the long term. By defining a standard canonical that represents both a book and a video at an abstracted level, you begin to achieve this transparency in a very flexible manner.

This order service, represented in Figure 4, is defined to accept a standard canonical defined as an xml schema for interacting with the multiple inventory systems based on the xml document passed in.

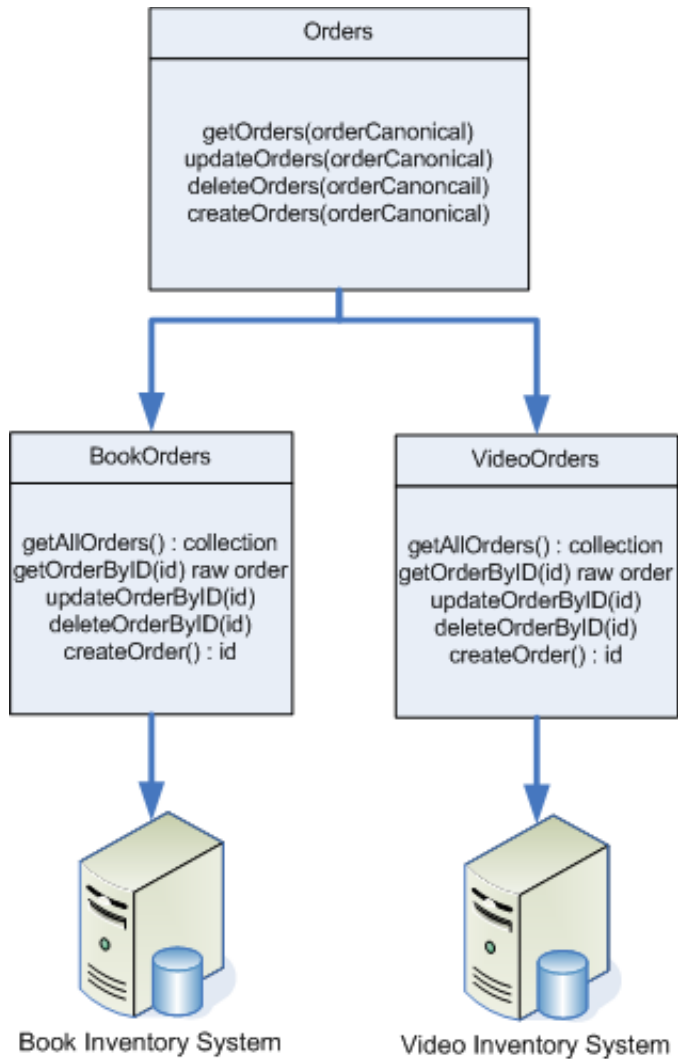


Figure 4: Canonical-based order service

The service implementation handles the logic for which physical inventory system is required based on the context of the message.

Logical services

One of the issues that arises when establishing standard canonical structures within the organization is that these structures need to support a very wide data footprint. As a result, consumers of these services may be concerned by degraded performance due to excessive information being returned from such a generic service when all the consumer really wants is a small subset of the result. Establishing logical services provides a more granular level of interaction without sacrificing the benefits of building on a canonical structure. Further, upstream applications may require logical groupings of information such as a single view of the customer. Standard logical services provide composite functionality and immediate benefits to reuse and reduction in time to market. Current tooling, such as BEA's AquaLogic Data Services Platform, also provides functionality to *compile-out* these layers at execution; this results in highly optimized queries without sacrificing design-time or run-time flexibility. The example in Figure 5 defines a customer profile service that provides a logical grouping of information for consumption.

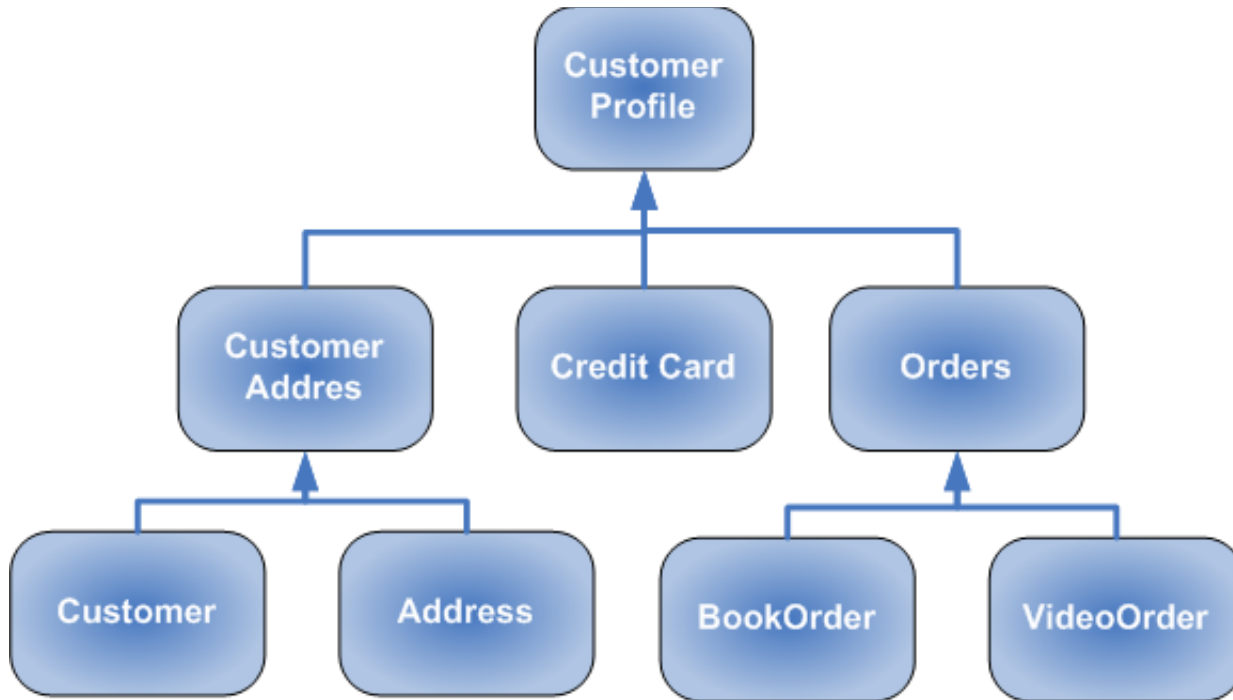


Figure 5: Customer profile logical service

This customer profile logical service depicts the ability to compose services throughout the catalog of needs in order to provide standard application structures for consumption by downstream services quickly and with little need to write new functionality.

Application services

Finally, the service engineering team may develop a number of application services that are intended to be consumed directly by applications in a line-of-business dependent fashion. The catalog of needs has two specific audiences that may be interested in a customer profile service: the actual customer who may want to update information, check orders, and so on; and the intranet user who may be interested in running reports on all customers who ordered a particular book, for example. As a result, the service engineering team may produce two application-specific services. These services may be exposed through presentation services such as portlets, as defined in the SOA Domain Whitepaper (PDF).

Pages: [1](#), [2](#), [3](#)

[Next Page »](#)