Understanding the Service Lifecycle within a SOA: Design Time

by Quinton Wall

10/04/2006

Abstract

Service Oriented Architecture (SOA) presents an architecture approach that relies on decomposing business processes and lower level activities into standards-based services. These services may be fine grained, course grained, presentation-centric, data-centric, or any number of other permutations. The ability to effectively manage the lifecycle of services is fundamental to achieving success within a SOA initiative. These discussions shall be divided into two articles focusing on design-time and run-time aspects of the lifecycle, respectively. This first article covers the design-time phases in the service lifecycle.

Traditional Application Development

To understand the need for service lifecycle management, it is important to recognize some of the paradigm shifts that SOA introduces into an organization. Many authors and analysts (Groves 2005, Dwyer 2006) have indicated that SOA requires closer alignment of business and IT. This in itself is often a large shift within the organization's operations. Another is the shift from development and deployment of applications to discrete services.

With regard to IT projects, an application may be defined as a collection of tasks designed to address the needs of a user or category of users. For example, a foreign exchange application may contain tasks and functionality to support looking up trades, defining an instrument, and submitting the order to a trade blotter. These functions are designed to target a category of user loosely defined as foreign exchange traders. Traditionally, an application was developed under a project or initiative within the organization. Project managers and analysts would define needs and functionality required, determine the level of effort, and identify the resources required and some controlling factors such as cost, scope, and time. Here is where some of the concerns with traditional application development lifecycles begin.

Very often requirements are baked into these applications at functional definition time. Agile programming methodologies have attempted to address this concern by iterating the requirements definition throughout the build process. This approach has alleviated the concern to some extent, but the rules and processing behavior may still be embedded within the application. Service lifecycle discussions will not directly address this issue but the composition of services through a SOA can provide a way of decoupling this logic. In addition, due to the tight coupling within traditional applications and the need to combine many functions within a deliverable, run-time governance of tasks is often quite difficult. Many of the decisions relating to security, quality of service, and so on are made at design time rather

than at run-time, therefore greatly reducing the flexibility of the application to adapt to changing business needs.

Of greater concern to traditional application development projects are the long development and provisioning times. It is not unusual for applications to take six to twelve months before being placed into production. I have been involved in many projects that have taken a number of years before being production ready. Quicker time to market is a critical aspect of SOA's appeal. Regarding service lifecycle discussions, it is important from the perspective that traditional application development processes often resulted in attempts to squeeze as much functionality into a project scope as possible, often more than is realistic! The result of this additional scope is one of two outcomes: reduced quality of the release, or project delays. Neither of these two scenarios is ideal for business or IT. Gaining an understanding of service lifecycle management directly addresses this problem.

With the rise of popularity and interest in SOA as a method for decomposing applications into services which can be shared across organizations it is not surprising that SOA principles are being leveraged in an attempt to address many of the negative aspects of traditional application development lifecycle. To date SOA adoption has shown considerable benefits in addressing these concerns but it also introduces a number of new requirements around lifecycle management, service management and organizational awareness. This article attempts to identify issues and best practices around what I will call the Shared Service Lifecycle (SSLC).

The Shared Service Lifecycle

Many organizations and analysts have used the term business service lifecycle to describe the lifecycle associated with SOA services. I prefer to use the term shared service lifecycle, or SSLC. Utilizing the term business within the definition signifies that I am talking only about those services that represent business processes (these are often composite services), not foundation services such as those that provide information and access to physical data sources and security services. By substituting business with shared, you are better equipped to recognize that service lifecycle management is directly affected by its relationship with other services in the enterprise, some of these being composite business services.

Figure 1 depicts a typical iterative SSLC divided into design-time and run-time aspects. This article will focus on the design-time aspects with a subsequent article focusing on the run-time phases of the SSLC.

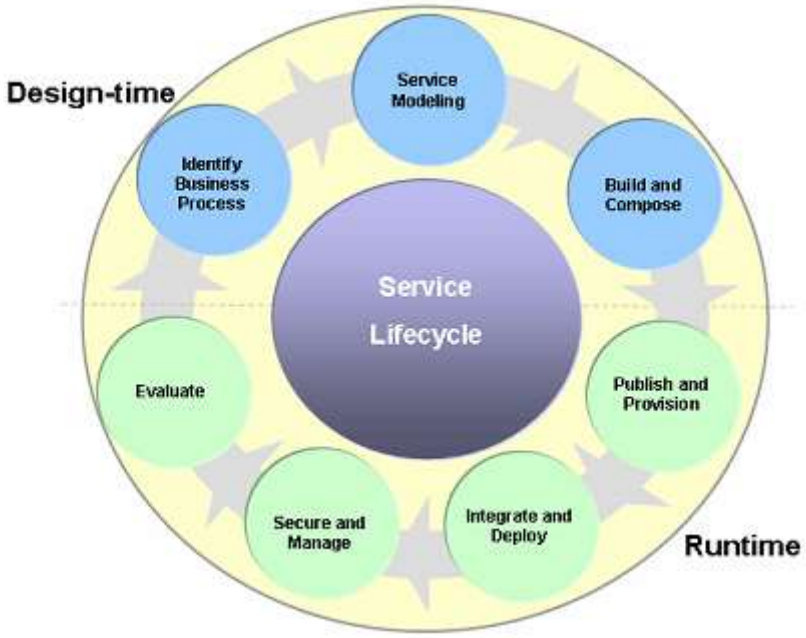

Figure 1: The design and run-time phases of a shared service lifecycle

Design-time Considerations in the SSLC

Let's now look at the design-time aspects of the shared service lifecycle. When I refer to design time I am focusing on the lifecycle of a service before it is put into production and made available for use. Many of the requirements of design-time modeling, such as developing a service modeling methodology, will not be covered within this article, but if there is enough interest I can write about this topic in future.

Identifying business processes

One of the core tenets of SOA is the alignment of business and IT and the establishment of the playing field. By identifying a business process that the organization feels will provide value through service orientation, the service engineering team (usually a mix of business, analysts, and IT personnel) can agree on a starting point for discussions.

Many organizations find it difficult to understand where to start with SOA and which business processes may be the most appropriate. A good approach is to start by defining a catalog of needs on a whiteboard. Divide the white board area into three swim lanes, each representing short-term needs (3 to 6 months – these are often more tactical in nature), medium-term needs (6 months – 18 months), and longer term needs (greater than 18 months – often strategic needs that may change as business demands change). After you have drawn your swim lanes, start adding needs for each of the areas. Try to avoid the tendency to think in terms of applications (for example, an e-commerce site); the farther out you look, the more likely it is you'll have only a high-level view of what you need to achieve. (For example, I need to consolidate my inventory systems.) At this stage of the lifecycle, you are looking at business processes that may become part of a business offering, such as the e-commerce site.

After completing this first run-through, the service engineering team may start looking for dependencies in an attempt to determine priority, expose opportunities for reuse, or identify dependencies between needs. Looking at the sample catalog of needs below, you can see that for this organization it makes sense to focus initially on the user registration process as it is depends on many other processes and can be reused across the e-commerce functions and the corporate intranet.

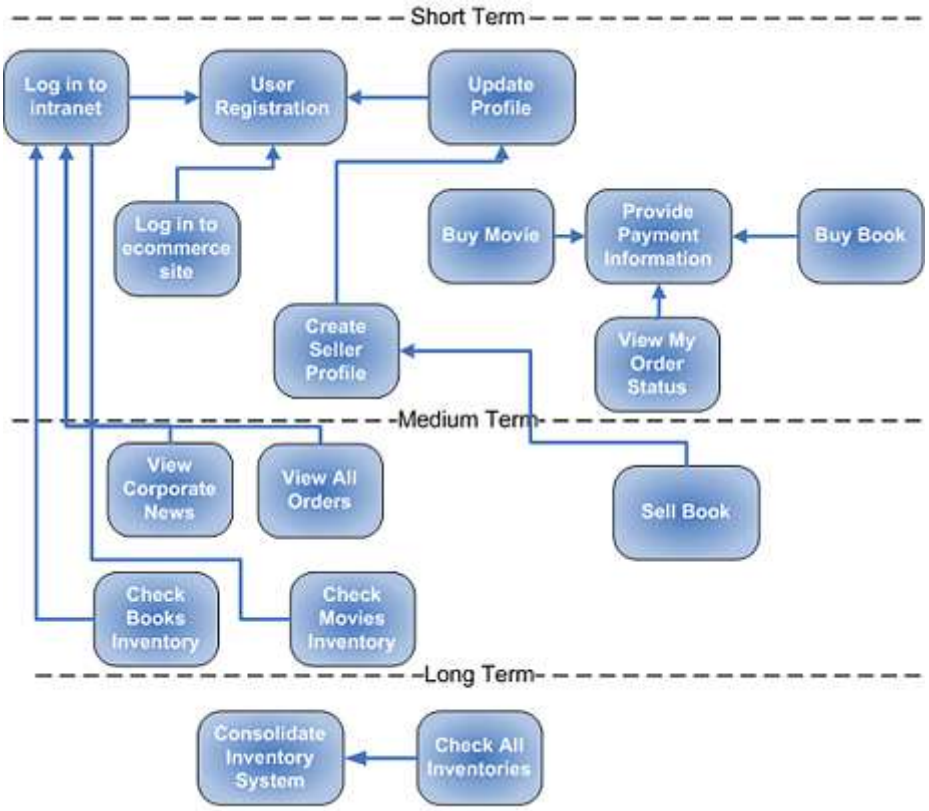

Figure 2: An example catalog of needs, which provides the service engineering team with a roadmap to achieve the to-be state of the organization

Depending on the maturity of the organization in relation to service design and development, choosing which services to develop first may naturally lead to building services that do not have many dependencies in an effort to gain experience. Although such thoughts are valid, it is important early in your organization's maturity to become familiar with service modeling techniques, such as strong contract and policy definition, that promote reuse. The service engineering team must be aware that the notion of reuse has been touted to business many times before, often with little success. Due to the shorter cycles of service development versus traditional application lifecycles, the service engineering team has the ability to demonstrate early success by focusing on building a series of foundation services from the short-term catalog that can be leveraged quickly across initiatives.

It is important to recognize, however, that the selection of initial services, in particular dependent services, should be balanced with the ability of the service engineering team. A new team will require time to become more experienced in the design phases of the SSLC. A dependent service identified through the services catalog may appear to be a good candidate due to the high level of reuse, but it may not be appropriate for a less mature team. If a service has upstream dependencies that span lines of business, provide enterprise quality functions, or are subject to strict quality of service regulations, it may not be an ideal initial candidate.

On the other hand, a service with a defined process and known end points that are controlled, mature, or small in scope and that is discrete enough to be built and rebuilt, if necessary, in a short amount time is a prime candidate for initial development. Such initial services should be able to be proofed out quickly to validate assumptions, methodologies, and processes. It takes experience and practice to get the design correct. Trial and error, especially in the formative stages of your SOA initiative, are an important mechanism to determine what works and what does not within your organization. Choosing stand-alone services without some dependencies early may limit the service engineering team's opportunity to learn important lessons in this formative stage.

Service design and modeling

The goal of the service design and modeling phase is to establish a consistent approach to defining service candidates based on the business process identified through the catalog of needs. This is where the "rubber hits the road," so to speak, with the service engineering team often white boarding the business process, decomposing the steps, and discussing current and future needs. In order for this to be effective, a consistent methodology for design should be established with common language defined that both business and IT understand.

A service design methodology provides the service engineering team a series of steps or activities that the team can use to decompose the business process to identify which aspects may make sense to be developed into a service based on service-oriented principles of design. This design methodology is something many organizations initially struggle with, especially in service granularity. Too fine-grained and you may get a proliferation of services that you cannot reuse; too coarse-grained and it may be difficult to get your hands around. Until the team feels comfortable with the modeling process, it should focus its activities on well-defined business processes that may not have large enterprise requirements, such as high throughput, long-running transactions.

Although technically not part of the modeling phase (but likely part of the modeling methodology), my experience has shown that it is important for organizations to invest time in defining service categorization guidelines. These guidelines should define what aspects of a service determine whether it may be a line-of-business (LOB) or application-level services versus an enterprise service with special needs. These guidelines may include throughput, quality of service (QoS), uptime, criticality to the business, and how many consumers are utilizing the service. In addition, these guidelines are very important to beginning to define organizational governance controls in relation to how services are funded, managed, and so on. Developing the guidelines can be a full-time job in itself but starting small and defining just what is required for current needs is a good approach. What's more, service categorization

may assist in grouping similar functions and identifying business owners for these functions. Remember that you can revisit the guidelines later as the need arises.

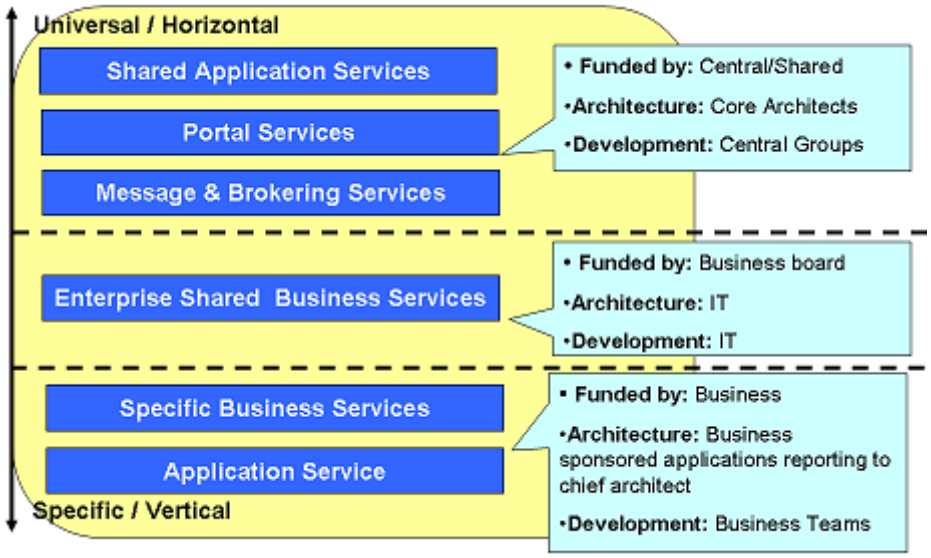

Figure 3: Service categorization and its relation to SOA governance; this categorization may assist in the definition of organizational governance controls of SOA assets.

Building on the services catalog example, the organization may have established categories of enterprise services and line-of-business services. Let's look at these in a little more detail.

Enterprise Services

Enterprise services have horizontal influence and may include:

- Security needs either at the perimeter or some form of centralized entitlement, for example, compliance rules for an industry.
- Auditing of activities. Remember that auditing may be an aspect of a particular function such as foreign exchange trading but not the process of placing the trade.
- Generic exception handling.
- The service requires 24x7 reliability and must be governed accordingly.
- The service demands high volume and/or low latency throughput.
- The service, based on context of use, may require a higher level of customer service or response time. For example; if the customer profile denotes that they are a gold-class customer, a service's contract may require a different SLA.
- If the service requires interaction across lines of businesses, there may be enterprise infrastructure requirements that need to be adhered to.
- The service interacts with enterprise data. This aspect may mean that the canonical model is owned by the enterprise but the implementation to the specific user data store is owned by the line of business. Experience and reality has shown me that more

often than not, a number of user data stores exist within organizations. Part of the SOA initiative may be to consolidate these aspects in the long term, but you need avoid a boil the ocean approach to your undertaking and leverage what is in place today while defining a plan for the future.

Line-of-Business Services

Such services have vertical influence and may include:

- Specific business functions such as purchase order (PO) or new hire processing.
- Presentation services with specific UI, look and feel, or wizards often used to provide visual representation for a specific business function.
- Information and access services to support CRUD (Create, Read, Update, Delete) activities for a line of business.
- Application services such as sales tracking or forecasting based on specific line-of-business data.

This categorization is by no means complete but should provide an indication of where an organization may begin their categorization efforts.

By examining the categories above you may be able to place some of your service candidates from the catalog of needs you defined previously into governable groupings and identify a number of canonical structures previously not apparent:

Enterprise Service	Line-of-Business Service
	Log into the Intranet (intranet infrastructure is typically owned by IT or a particular LOB)
Update Profile (Canonical Profile)	Update Profile (Service)
	Log into eCommerce site
Seller Profile Canonical	Create Seller Profile
Inventory Item Canonical	Buy Movie
Inventory Item Canonical	Buy Book
	View My Order Status
Payment Canonical	Provide Payment Information
Inventory Item Canonical	Sell book
	View Corporate News
Inventory Canonical	Check Movie Inventory
Inventory Canonical	Check Book Inventory
Check All Inventories	
Consolidate Inventory System (long term are often planned initiatives vs. actual services)	

The point of the service lifecycle is to move towards solving business needs, and not get weighed down in a detailed taxonomy exercise. The evaluate phase of the SSLC is intended to support this reassessment based on actual usage and context. I like to think of the movie Field of Dreams with Kevin Costner in which he hears the voice repeatedly saying If you build it, they will come. This is not dissimilar to exposing services in the enterprise. Something defined for a certain usage level at a given point of time may be utilized in a completely different way in

reality, often a way not considered at initial design time. Guidelines should assist in the recategorization phase.

At this stage in the process I am talking about the notion of service candidates versus service implementations. Erl (2004) suggests that candidates are potential services that may or may not be realized into the eventual design. The design process is intended to identify the inputs to future phases of design and development. It is important for the service engineering team to understanding what already exists within the enterprise verses what needs to be developed. Tooling that supports discovery of services, such as a UDDI-compliant registry, is an important component to promoting service reuse and understanding what already exists that may be leveraged.

Finally, during the modeling phase and with your understanding of the fact that you are attempting to define service candidates, the service engineering team should continue to design through the established methodology independent of realities of technical architecture and constraints of the physical environment. The intention of the service design and modeling phase is to define a desired future state. The build and compose phase of the SSLC will subject service candidates to organizational constraints in an effort to define final service implementations.

Build and compose

The build and compose phase of the service lifecycle focuses on the tasks required to develop new services and also leverage what already exists within the organization in an effort to develop new functionality more quickly and cheaply. This approach should translate into a reduction in time to market, one of the key financial benefits to SOA.

At this stage the service candidates that have been identified in the service modeling and design phase are rationalized into service operations with the infrastructure and environmental reality mapped to them. As mentioned in the modeling phase, it is important to identify the goals of the SOA initiative. Achieving these goals may not be feasible due to current restrictions or limitations in your environment but may promote some healthy discussion and potentially some form of cost benefit analysis to determine how to achieve that desired future state. For now, however, the organization needs to move ahead so your candidates must be realistic in your organizational ecosystem.

With an understanding of which service operations and implementations are realistic, you can build on any opportunities for reuse and composition identified in the previous phase. To fully leverage SOA, the notion of composition is very important to business agility. The development environment and service infrastructure tooling must facilitate design-time discovery of services and provide the ability to compose these services together to complete a business process.

Without such tooling, the success of your SOA initiative may be limited. As initial services become available to line-of-business teams and other engineering groups, opportunities for composition may be realized. In this case, while cataloging needs you identified initial dependencies. These dependencies should be interpreted as direct opportunities for building composable services and should provide the tangible benefit of reuse. I have hinted at

composition throughout this article but the importance of such activities is directly relevant to the build and compose phase of the SSLC.

Consider the catalog of needs example: An initiative called Consolidate Inventory System has been identified in the long-term goals. At first review this task may be interpreted as physically retiring old inventory systems and consolidating the repository into one master source. Although this may indeed be the case, if a cost benefit analysis indicates it is more cost effective to retire old systems, the activity may also indicate a less obtrusive approach. The services engineering team may produce a series of logical data services hiding the physical end points to the consumer. Such an approach of building a ubiquitous data-access layer would directly leverage existing check inventory X services developed in the mid-term catalog of needs through composition. The consolidate inventory systems initiative may then require little more than decision logic based on a canonical representation of an inventory document to determine which end points need to be modified. This sort of distributed CRUD logic should be provided through the Service Infrastructure Tooling. An example of this may be the BEA AquaLogic Data Services Platform.

Very often services originate at the line-of-business level rather than through enterprise initiatives as typically this is where project funding and need is driven. As a result, the "*if you build it, they will come*" scenario may result in design-time discovered services that may not be good candidates for reuse. They may not provide adequate performance or consistent schemas. Although they are available in the enterprise, they should remain as application-level services. As a result, the organization must begin to establish a governance process to control enterprise visibility to services. Very often, service registries are leveraged to provide control mechanisms and processes to ensure service quality. Many of these aspects must be addressed in the publish and provision phase of the service lifecycle.

Finally, to provide rapid development, experience has shown that standardizing on tooling enables the organization to leverage learning and reuse across SOA initiatives. This is not to say everyone must use the same IDE or one specific tool; rather, any tool that is used must function in a similar fashion, must support standards, and must promote a reduction in the learning curve if a developer is required to support other projects with different tools. In addition, these tools must support the ability to easily capture metrics with regard to reuse and time-to-market for services developed. Capturing metrics through the service lifecycle provides invaluable information to the organization when demonstrating the success of a SOA initiative.

The BEA Domain Model

As with many methodologies, an underlying theme need to be established that unifies all other activities. In the case of BEA and SOA this is BEA's Domain Model (requires registration). A number of articles on Dev2Dev describe the importance of understanding each of the aspects of SOA (see Successfully Planning for SOA by David Groves for a great introduction). The Shared Service Lifecycle leverages this model and provides tangible control points along the way. In the case of the design time phases as identified in this article, the influence of the

domain model is represented through the need to define projects and applications in alignment with the catalog of needs through an architectural approach.

This approach often begins with a vision and is initially implemented through foundation services or building blocks. Although not as critical in the design phase as it may be in the run-time aspects of the SSLC, governance begins to exhibit a degree of influence across the process, in particular when determining initial service realizations.

The second part of this article series will demonstrate the importance of measuring costs and benefits of deployed services and an increased focus on how services are governed at run-time. In addition, both the design-time and run-time phases of the SSLC require close alignment with the business strategies and processes. This was evident in the need to identify and design business processes that become our service candidates and may be composed into reusable services to achieve business agility.

Summary

By further understanding design time needs with regard to shared service lifecycle, organizations looking to SOA to promote reuse and increase business flexibility may recognize that establishing fundamentals early, such as methodology, categorization guidelines, and development tools, is crucial to early and continued success. By beginning to break the traditional application development paradigms and focus on business processes as the blueprint for moving forward, service engineering teams can provide closer alignment to business needs in a timely and efficient manner.

The second part of this article will focus on the run-time aspects of the shared service lifecycle.

References

- SOA Compass: Business Value, Planning and Enterprise Roadmap, Bieberstein et al (2005); available at Amazon.com or other retailers
- Business Benefits of Implementing SOA by Tom Dwyer (The Yankee Group, 2005)
- Service-Oriented Architecture: Concepts, Technology & Design, by Thomas Erl (2004); available at Amazon.com or other retailers
- W3C (2004) Web Service Management: Service Life Cycle

Quinton Wall is a Sr. Product Marketing Manager for Integration at BEA where he is responsible for articulating the strategic vision and direction of the products such as WebLogic Integration and AquaLogic Integration.