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.

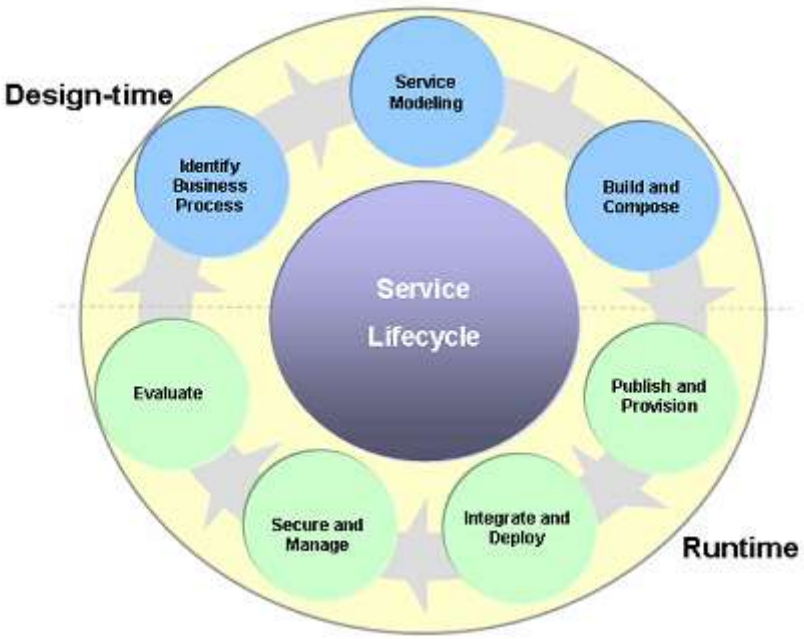

Figure 1: The Shared Service Lifecycle (SSLC)

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.

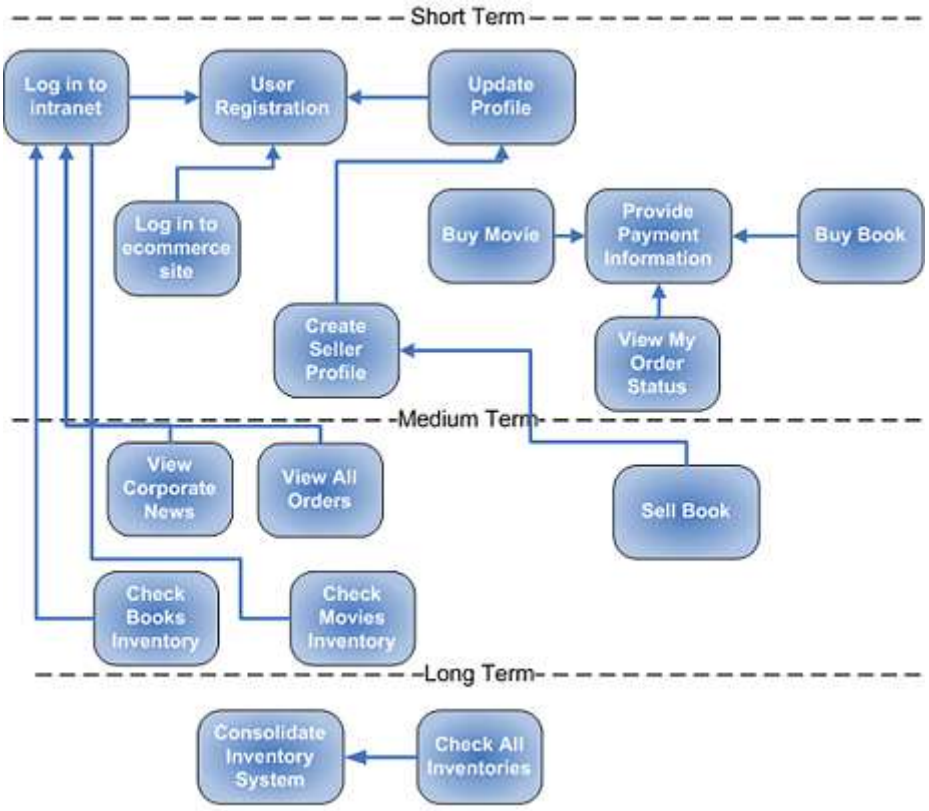

Figure 2: The catalog of needs

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.

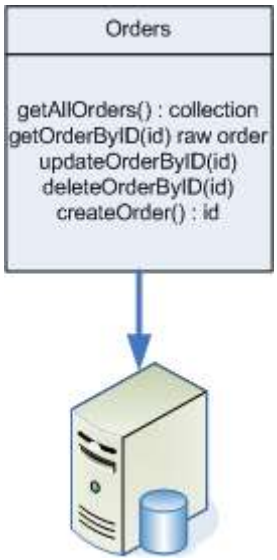

Figure 3: The Order Physical Service

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.

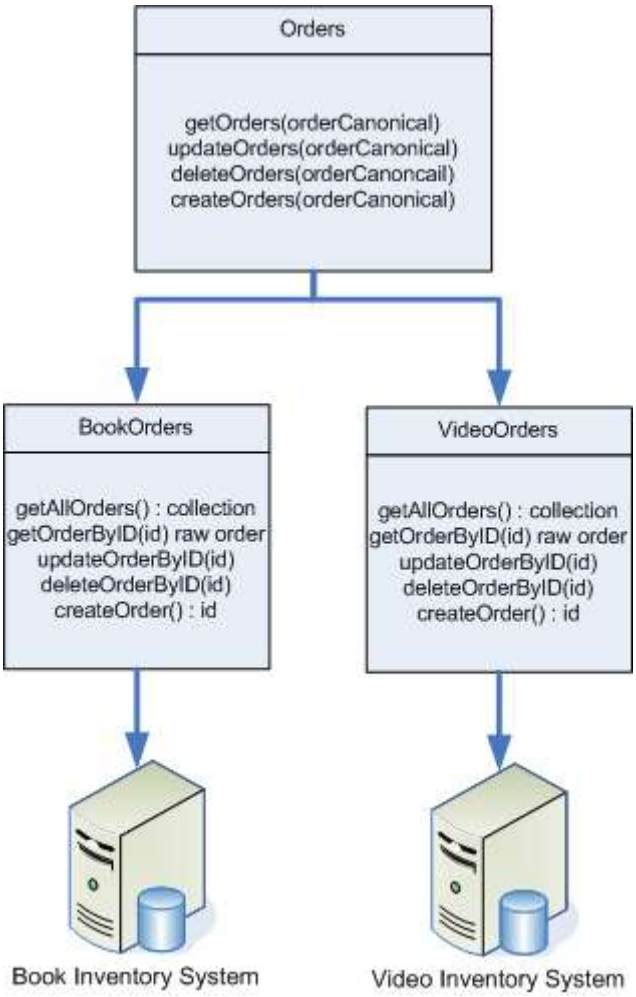

Figure 4: Canonical-based order service

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).

The Run-Time Phases of the SSLC

With a deeper understanding of the importance of layering services within a SOA, let's now focus specifically on the run-time phases of the SSLC. If managed correctly, these phases may provide a key ingredient to achieving organizational flexibility and agility powered by your SOA initiative. The following section provides detailed discussions on these aspects.

Publish and Provision

The Publish and Provision phase of the SSLC focuses on the understanding of what you need to do to release a service and establish boundaries for its use. In particular, this phase should establish the control points for SOA governance, and promote the acceleration of service reuse by defining such aspects as service interfaces and contracts.

When looking at publishing and provisioning services, it is important to have a strong understanding of what "pieces" of the service you are dealing with. At this stage, the implementation of the service is considered a static artifact of the design time and not the responsibility of the provisioning team. The provisioning team should focus its attention on the following aspects:

Service interface

The service interface provides a standards-based mechanism for consumers to access the functionality offered by the implementation according to the contract it offers. It is through the service interface that the service will be published for consumption.

Service contract

The contract should contain both functional metadata (how to interact with the service) and non-functional metadata (what conditions and restrictions apply to consumers looking to consume the service). By ensuring that the service implementation is free of logic such as conditions and restrictions to consumption, the service is more likely to be able to be composed in a myriad of ways, most of which have not been considered by the service engineering team or business at this point. It is important to realize that service contracts may be changed at run-time without the need to recompile code in the implementation; the trick is to leverage this flexibility efficiently.

Metadata and policies

In my experience it is the metadata and policies defined to support a service that truly enable service reuse and composition. Management of metadata is the secret sauce in establishing a truly flexible SOA environment. By defining a series of policies, the run-time control of a single service or a category of service may be easily managed external to implementations. The policies define not only which rules apply to which consumers but also to the service itself with regard to who can provision what and when. The added benefit of metadata-driven policies is that they can be enforced under dynamic conditions such as the context of the users session, and message payload.

In focusing on runtime requirements of the SSLC, control and governance begin to play an increasingly important part to successfully managing your SOA initiative. As a result, services should be published to some form of registry that can support configuration and change management aspects of a services lifecycle. This registry may also integrate with an enterprise repository, which may also house additional related information. In particular, control points should be established that, through tooling, should notify existing consumers of changes to a service, and potential governance policies should be established to provide a window of time for consumers to migrate to a new version of a service, if required. All of this information may reside in the repository and may be presented to the service consumer through the registry.

Integrate and Deploy

The SSLC and SOA promote flexibility and agility as central tenets of success. Service reuse is one method for accomplishing these goals. As identified in the Build and Compose phase of the SSLC, services may be consumed as part of new service implementation. It sounds obvious but this statement is important to ensuring that a design-time composite service does not inhibit flexibility through the Integrate and Deploy phase of the SSLC. The reliance on service implementations to dynamically assemble services to support business needs may cause issues with dependencies and version mismatches when changes are introduced at later phases. It is therefore suggested that you leverage dynamic tools such as Business Process Modeling (BPM) to facilitate roundtripping of service linkages rather than embedding them into the service

implementation. Another approach may be to utilize proxy services in your service implementation rather than use true business service end points. By exposing proxy services through an Enterprise Service Bus, the physical service may change independent of consuming services. (Of course, if the service interface changes upstream, consumers should be notified through an established governance process.)

Unfortunately, dynamic assembly of services at run-time does not just happen. The ability to integrate and deploy seamlessly requires a decoupled service environment that targets the need to reduce point-to-point relationships between services or intelligent end points—these denoting that a service implementation contains coded logic inhibiting reuse of the service in a way that has not been explicitly planned (coded) for. In addition, the service engineering team must recognize that services do change over time and that new versions shall be released while others are retired. The integration and deployment phase should focus on support for multiple versions of services that may need to coexist for a specified period of time. Overlapping such a need is the governance control mentioned previously to determine appropriate actions to limit service sprawl focusing the organization into a kind of primordial soup of uncontrolled producers, or, as a colleague of mine so eloquently put it, "You need to eat your IT vegetables and not just your IT candy."

In order to support such dynamism within your SOA initiative, you need to look at mechanisms of configuration-based routing that, instead of hard coding end points, have the ability to interrogate the payload or context of a service request and route it appropriately in a configuration-based approach. Further, the environment needs to support seamless change control that provides accurate auditing of change information (remember, you reap a lot of the financial benefits of SOA through cost avoidance in aspects such as compliance!), provide the ability to aggregate configurations into manageable groups, provide real-time changes and simultaneous session support for transparent change control, and, of course, provide the ability to roll back when problems occur.

Finally, the Integrate and Deploy phase of the SSLC should support the notion of role-based operational views. These views should allow delegated configuration of composite services and not just limit their visibility and change management functionality to the service contracts defined in the Publish and Provision phase. By purely focusing on individual services and the ability to manage change at this level, the organization may not realize the full benefits of composite applications and be in danger of reverting back to some of the practices of traditional application development resulting in systems that are difficult to monitor, and provide concerns related to break-fix-style regression testing.

Secure and Manage

Initiatives within your organization can be successful only if they can be accurately managed to respond to the needs and demands of the business. In a SOA capacity this entails providing enhanced quality of service to the right customers through an understanding of the context in which the services are being used. I often think of the economic models that identify specialization as a way to increase the production probability curve within an economy. Without going into economic theory too deeply here, the ability to enable a shared service

infrastructure to understand usage needs in both a proactive and reactive approach can provide a higher level of customer service or leveragability. In economic terms this may relate to an understanding of the aggregate demand of a service.

The Secure and Management phase of the SSLC should allow the organization to manage security constraints through the policy rather than the service implementation. The service policy can dictate such aspects as transport-level security with regard to the protocols the service can be called by as well as leverage WS-Security standards to ensure interoperability. Of course, lower-level data security can be managed through abstracted data services or some form of distributed entitlement engine. Specifically for the SSLC, this security is provided through management of the policy and associated contracts rather than service implementation.

At any stage in the SSLC it is important to have visibility into the process in real time. The Secure and Manage stage in the SSLC focuses on managing services to business requirements based on current reality usage. This may include SLA management on performance and error events. The ability to manage this information is an important aspect to adhering to governance control points established within your industry or organization. For example, let's consider that for each book to be purchased from the fictitious organization, you need to ensure that the order is responded to within a certain timeframe and any books of adult nature are sold only to people with whom some form of age verification has been conducted. In this case, a service policy may be implemented that confirms age verification has passed—possibly through the introspection of current customer canonicals. (It is important to note that, in a highly leveraged environment, a single service implementation can and should provide multiple service policies and contracts to determine its use.) Continuing the example of the purchase of an adult book, two policies are actually in effect here: first, the age verification if the book is of an adult nature, and second, a SLA that requires orders to be shipped within a particular timeframe. Any breach of these SLAs may be reported as an exception or violation of policy and likely reported in some manner. A mature SOA organization may have clearly defined governance processes that clearly identify control points and exception routines, all of which are likely to be recorded in an enterprise repository such as AquaLogic Enterprise Repository.

With SLAs and business policies established, the organization may leverage enterprise dashboard-style functionality to provide immediate and centralized feedback to users who may require visibility into the operational state of the organization. Through such visibility, the Secure and Manage phase should provide the run-time flexibility to enhance quality of service delivered to the consumer if the situation dictates it. One such example is the ability of the system to "dial-down" the service if an end point is unavailable or route the request to a lower-cost channel such as IVR (Integrated Voice Recognition) rather than costly CSR (Customer Sales Representatives) for the majority of standard requests.

Evaluate

Just like fiscal and monetary policy of governments, planning and preparation are subject to change when actions are put into practice. The final phase of the SSLC involves detailed

analysis of how services are being used in run-time based on actual usage. The intention of such analysis and evaluation is to allow an organization to better govern and manage service usage behavior based on actual production influences.

The evaluation phase focuses on taking production services and assessing their usage based on the service categorization guidelines established as part of the service modeling methodology presented in Part 1 of this series. Experience has shown that a service developed initially with a particular function or level of use in mind may actually be much more heavily used than originally anticipated. Such run-time changes are part of the desired organic growth of the SOA within an organization, but they must be managed effectively to avoid detrimental effects upon SLAs and SOA effectiveness. During the evaluation process, the service engineering team should determine whether the service is being utilized as envisioned or needs to be recategorized to maintain or improve desired levels of service. This recategorization may change how a service is governed going forward and require a change in support or funding structure. This is especially true if a horizontal service, one that is initially designed for line-of-business usage, begins to be utilized across the enterprise. Such a service may be recategorized as an enterprise service and governed accordingly.

Another important aspect of the evaluation phase is to leverage the service infrastructure to collect metrics for demonstrating ROI and success of the SOA initiative. Both business and IT benefits may be identified through the gathering of SOA metrics. Determining ROI and cost benefits of SOA is a broad topic and beyond the scope of this article (but is being addressed in a further article I am writing). For the sake of brevity, one of the metrics that the SSLC is concerned with is the level of reuse of services. In particular, metrics that measure how often services are being reused verses being created are important when evaluating the future success of the SOA program. For metrics to be truly effective, they must be captured from day one and able to be compared to past metrics information such as time to market, reuse, and reduction in the break-fix cycle, of previous IT initiatives.

Finally, the evaluation phase of the SSLC should be leveraged to drive the next round of service candidates in conjunction with the catalog of needs. Usage metrics are direct evidence of whether the dependencies identified within the catalog of needs accurately represents reality. In addition, service usage and composition may identify additional relationships that warrant developing specific, governable, and measurable services to support. Through evaluation these relationship services are likely to become evident much more easily than purely through the design-time phases of the SSLC.

New Challenges of SOA

Many papers and articles identity the benefits of a SOA for organizations. However, any new technology has new challenges associated with it. In particular, SOA SSLC and related governance processes must be prepared for these challenges. The following section identifies a handful of challenges, some good and some bad, but all important for the service engineering team to have a good understanding of them.

Change

The preceding section described the run-time phases of the SSLC and how they relate to an organization's SOA initiative. Just as important as the SSLC is one constant factor that requires special attention: change—an inevitable and necessary part of business evolution. Having an accurate understanding of the SSLC is a great way to be prepared for change and allows a consistent methodology for implementing a SOA initiative. Managing change within SOA and the SSLC introduces new challenges to managing change. In particular, introducing services into your organization introduces a number of new abstractions that require careful consideration. Such concepts as service contracts and policies require a different way of thinking about how functionality is developed. This takes time and practice to get right until the shared services team can promote some form of consistent adoption across the organization.

SOA also introduces the likelihood that the organization may have multiple versions of a service implementation in production at any given time. This idea, foreign to many organizations, introduces governance and support challenges pertaining to how long to support prior versions, support for consumer applications, the ability to change versions without detrimental upstream effects, and the ability to notify consumers of impending changes. I have worked with a number of organizations that have addressed these issues purely through governance to "force" consumers to migrate to the latest service version. Although this is definitely one solution, a more mature organization may provide a window of opportunity for consumers to migrate to the latest version if required, offer support for previous versions though a service registry and also leverage the flexibility of service contracts to provide seamless change control, transformation of requests, and so on. You need to remember that SOA is about promoting the agility of organizations. Requiring application teams to upgrade to each point release of a service may be considered detrimental to such agility.

Packaged applications

Another new challenge that the success of a SOA initiative within the organization may come from a very unexpected source; your existing packaged applications. Most vendors these days provide web service interfaces to functionality within their products which are enabled by default. Depending on the size of these packaged applications within your organization, they may introduce hundreds of services, which may not adhere to your established service guidelines. The influx of unstructured services is likely to wreak havoc on the success of your SOA initiative. These services may not adhere to SLA requirements, may be managed by lines of business verses enterprise groups, and may also cause concern from a compliance perspective. This is especially true if the services exposed are published from core enterprise systems such as CRMs, Data Warehouses, and ERPs. It is strongly suggested that the organization establishes an aspect of the governance model to specifically deal with what services should be exposed and managed accordingly. One such approach may be to provide additional service policies or contracts to control package applications through the Publish and Provision phase of the SSLC. In conjunction with some prep work at the Secure and Manage phase, this approach may alleviate a proliferation of ungovernable services within your organization.

Service contracts

When you consider service contracts as part of implementing a service, most of the time you don't really give it much thought except for the fact that contracts help determine service usage. This is true but often people fail to recognize that a service contract is a formal, binding agreement that may restrict future flexibility. By definition, a contract is a binding agreement between two or more parties to behave in a particular way. Like it or not, by establishing a service control, the producer and consumer are governed by its stipulated rules.

Let's jump back to our catalog of needs that determined that we wanted to develop services to accept as an input a generic orders canonical. This canonical schema forms part of the binding contract that you expose. Any consumer must provide this canonical as an input parameter to the service. Careful attention must be paid to ensure that the schema does not allow too much flexibility resulting in many permutations of how a service may be called. The result of very flexible contracts is that the service becomes brittle and prone to difficulty when attempting to debug or maintain the service.

On the positive side, service contracts provide a great way of ensuring that your services can be leveraged quickly and efficiently. By designing service implementations to free of business logic and run-time decisions the service is more likely to be able to be leveraged as part of a composite service. Such benefits can be further achieved by recognizing that services can have multiple contracts that determine how they are used at run-time. This concept is similar to polymorphism in object orientated (OO) programming. Best practices in service design allow independent evolution of needs at the contract level, thereby keeping the service implementing simple and clean. The ideas of a service having multiple contracts can be extended further when it becomes clear that policies and contracts may have many-to-many relationships themselves and that multiple services may implement the same contract. Suddenly, the run-time operation of a system can be managed in real time by leveraging the power of a services repository to tweak usage as needed. True loose coupling requires the presence of many-to-many relationships within your services infrastructure. This is a powerful concept that, in my opinion, is still in its infancy within SOA initiatives.

Unfortunately, the full benefits of SOA gained through the leveraging of contracts and policies at run-time includes some technical limitations. In particular, WSDL cannot currently support all functional and non-functional requirements needed by an organization. In addition, the WS-Policy specification falls short in that it provides only an interoperable container for exchanging policy information but does not provide the support to specify that policy behavior. In time, I am sure these limitations will change as organizations continue to mature with their SOA adoption and techniques.

Metadata

In my experience, in the long run the success of an organization's SOA initiative is dependent on how it manages and leverages metadata. Although many organizations will benefit from SOA initiatives by reduced time to market, service reuse, and so on, I believe that the harnessing of metadata within the organization will prove to be the difference between short-term success and long-term transformation. Metadata, the description of services and how consumers of those services interact with them, provides the flexibility to focus on the

creation of shared services through composition to meet the business needs on a continual basis.

Having an understanding of metadata helps the organization avoid hard coding governance and process into tools and implementations and allows dynamic discovery of resources. The organization must be able to leverage the SOA infrastructure for more than just service details. A metadata repository must be established that supports all SOA artifacts and allows the description of a process within the environment and the context of the operation. This ability to manage metadata effectively will have a direct correlation with the ability to establish appropriate and measurable control points throughout the governance process.

Summary

To harness the full benefits of a SOA initiative within your organization, it is critical to adopt the notion that a service in run-time is a corporate asset that can be inventoried and tracked. This will help not only in building your shared service catalog and reducing time to market through reuse but also as a continued justification of SOA spend. If you cannot track and measure the benefit or cost avoidance through your SOA platform, future cost benefit decisions may be limited to subjective information.

Through accurate run-time governance of shared services, the organization shall improve the effectiveness of its SOA initiative. Unlike traditional software development, SOA focuses on building reusable atomic services that facilitate flexibility through contracts and policies. Through the successful design of services and consistent modeling methodologies, run-time aspects of service orientation should be more focused on the service infrastructure present in your organization.

In addition to solid design and run-time practices, organizations should recognize that SOA introduces new challenges that must be managed effectively through a consistent governance model and an embracing mentality. By embracing environmental certainties, such as change, your SOA initiative should provide organic growth matched only by the organization's desire to remain competitive and be considered a leader in the industry. After, all change is about innovation, and innovation is about thinking differently from the rest. If managed correctly, SOA can be a critical component to this change.

Finally, as an architectural paradigm, SOA is aimed at promoting reuse and allowing the business to bring new offerings to the market more quickly. As a methodology, SOA requires a detailed understanding of how an organization can rapidly adapt to change. The run-time aspects of the SSLC are directly related to building organizational flexibility. This article has attempted to provide insight into how, through careful analysis of current needs and service-orientated design practices, organizations may achieve this goal.

References

- The ROI of SOA - Ronald Schmelzer (ZapThink, 2006, Document ID: ZTZN-1187. Available with registration)

- Grabbling with SOA Change and Version Management - Ronald Schmelzer (ZapThink, 2006, Document ID: ZAPFLASH-2006519. Available with registration)
- 10 Rules for Service Design - John McDowell (FtpOnline, 2005)
- Service Orientated Architecture: Concepts, Technology, and Design - Thomas Erl 2004
- SOA Data Strategy - Author Unknown (Sys-con) 2006
- The Rise of the Software Architect in a SOA World - Miko Matsumara (DevX, 2005)

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.