SearchSOA.com E-Guide

# SOA Implementation

Service-Oriented architecture (SOA) allows organizations to quickly and efficiently respond to changes in the business environment and to leverage such change for competitive advantage. The SearchSOA.com E-Guide: SOA Implementation features expert insight into the key areas critical to the successful planning and implementation of an SOA. SearchSOA.com experts examine unique aspects to SOA implementation and offer an SOA implementation checklist in which practices need to be considered as part of typical strategic SOA planning efforts, effective approaches to SOA Governance and common mediation and orchestration requirements.



TechTarget
*The Technology Media ROI Experts*

SearchSOA.com E-Guide

# SOA Implementation

## Table of Contents:

# Business agility as an emergent property of SOA

By Jason Bloomberg

As students go through our Licensed ZapThink Architect (LZA) course, they experience a series of "aha" moments, as we systematically tear down their preconceptions about what service-oriented architecture (SOA) is -- and what it is not. But perhaps the biggest aha moment of all, however, is when they realize that implementing SOA isn't traditional systems engineering (TSE) at all, but rather a fundamentally different approach to dealing with complexity in the IT environment. Needless to say, this realization is an especially big wakeup call for people with TSE backgrounds!

The fundamental shift in thinking is this: TSE focuses on building big systems out of small components, where the behavior of the resulting system depends directly on the properties of the components. Essentially, TSE boils down to a "connecting things" way of thinking about distributed computing, where integration is the central activity, and what you end up with when you're done with all the integrating is at best what you expected to build.

SOA, on the other hand, calls for an entirely different approach. In the SOA context, we focus on building and maintaining the Business Service abstraction, which supports inherently unpredictable behavior as the business composes services to support fundamentally dynamic business processes. Essentially, with SOA we're building for change, while with TSE, we're building for stability. The problem with stability, of course, is it only takes the business so far -- if the organization requires business agility, then they're much better off implementing SOA.

## Business agility as an emergent property

Business agility, which ZapThink defines as being able to quickly and efficiently respond to changes in the business environment and to leverage such change for competitive advantage, is the primary strategic motivation for most SOA initiatives. What is it about SOA, however, that enables such agility? Unlike with TSE, where the properties of the resulting system depend upon the properties of the components that make up the system, the individual elements of a SOA implementation, including services, service compositions, policies, contracts, and the like, somehow contribute to the agility benefit, even though each of them don't exhibit business agility themselves.

We call properties that certain systems exhibit that their individual components do not emergent properties. In addition, we call systems that exhibit emergent properties complex systems. Complex systems theory is an exploding area of study today, because so many different systems in the world exhibit emergent properties. Emergent properties include everything from friction to traffic jams to the human mind, and the field of complex systems theory is gradually explaining many of the more subtle aspects of the world around us.

## Complex systems engineering: The key to implementing SOA

Explaining natural phenomena is one thing; building a complex system is quite another. We call such a practice Complex Systems Engineering (CSE). If you're looking to engineer a system, where your desired outcome is an emergent property like business agility, then TSE won't do -- you need to take a CSE approach. As a result, it is imperative that architects looking to implement SOA take such an approach, because business agility is a critically

important emergent property, and in many ways defines the success criteria for SOA initiatives. Leveraging complex systems best practices, therefore, may be able to give us some important insight into how to deliver on the business agility promise, and perhaps more importantly, how to avoid impediments that might prevent a SOA project from providing the required agility.

A fascinating paper from 2006 by David A. Fisher from Carnegie Mellon University's Software Engineering Institute provides a marvelous starting point for a discussion of SOA in the context of a particular type of complex system known as a system of systems. According to Fisher's paper, systems of systems are qualitatively different from traditional large-scale systems. Such systems of systems are far more complex, and involve independently managed and operated components. Furthermore, systems of systems depend upon other systems that are outside the control of their owners and users. As a result, TSE approaches and methods are often inadequate or inappropriate for systems of systems.

Systems of systems have unique characteristics that distinguish them from traditional monolithic systems, including higher levels of complexity, flexibility, and emergent behavior. Such characteristics result from the operational and managerial independence of their constituent parts, from independent evolution, and from the character of emergent effects, while traditional monolithic systems depend on central control, global visibility, hierarchical structures, and coordinated activities. As a result, such traditional systems are unable to exploit the advantages of emergent behavior like business agility.

## Is a SOA implementation a complex system?

Many of the characteristics of such systems of systems line up quite neatly with how ZapThink sees SOA. The core SOA principle of loose coupling in particular leads to the system of systems requirement of autonomous systems. In fact, complex systems theory recognizes that a result of tightly coupling systems is the fact that accidental failures of individual subsystems lead to cascading failures across the entire system. Furthermore, systems of systems depend upon interoperation among systems, rather than integration, where integration of subsystems leads to a unified system, while interoperation among systems is the combination of autonomous systems into a system of systems.

Interoperation in systems of systems demands what Fisher calls a node-centric perspective, where each constituent (a service provider or consumer, say) views the system from its own individual perspective. For each node, a node-centric perspective describes the interactions with its immediate neighbors based upon available metadata. In this view, the behavior of the overall system of systems depends upon the interactions between service providers and consumers. An individual constituent node need have no knowledge of the details of other nodes that it doesn't interoperate with.

To achieve this interoperation, systems of systems should be interdependent with other systems beyond their boundaries, where it's impossible to predict the resulting outcomes of the architecture. Furthermore, requirements for systems of systems that interoperate are constantly changing and imprecisely known, as they typically are for SOA implementations. Interoperation also encourages distributed control, which aligns nicely with the business empowerment benefit of SOA.

It seems, therefore, that an architectural approach like SOA that leads to loosely coupled, autonomous, interoperable services in response to dynamic business requirements is a perfect example of a system of systems. Fisher, however, makes an interesting claim to the contrary: that SOA implementations, at least at the time he wrote his article, don't provide sufficient interoperation in systems of systems because they assume an absence of emergent behavior or at least fail to recognize and provide support for managing emergent behavior. The question of whether SOA leads to emergent behavior like business agility, therefore, depends upon whether complex systems theory applies to SOA at all.

## Taking a CSE approach to SOA

If business agility is an example of emergent behavior, and we expect SOA implementations to provide such agility, then how can we say that SOA implementations assume an absence of emergent behavior? This conceptual disconnect is at the heart of the aha moment our students experience in our course. Far too often, people assume that TSE is sufficient for implementing SOA, and TSE thinking excludes emergent behavior as a possible outcome. We see this misconception all the time, whenever an organization believes that they must purchase integration software like an Enterprise Service Bus (ESB) in order to implement SOA. TSE means connecting things, so the obvious place to start is with something that helps you connect things -- but that approach misses the boat entirely.

Fisher's second point is also well-taken. To put this point in another way, we would need to manage a SOA implementation's emergent behavior in order to achieve the benefits that systems of systems exhibit. ZapThink hammers home this point whenever we talk about business empowerment. If we mistakenly conclude that the sole point of SOA is to build all these flexible services and then simply turn them over to the business to compose willy-nilly, then it's true we'd never be able to achieve the business agility benefit, because such a haphazard lack of control would lead to immediate chaos. On the contrary, the key to the business empowerment benefit of SOA is governance -- a set of organizing principles that tempers the emergent properties of the architecture, thus providing the essential management of emergent properties that SOA requires for us to consider SOA implementations to be complex systems.

## The ZapThink take

There is an important insight here which is the essential point of this ZapFlash: for SOA to be successful, organizations must take a CSE approach to their implementation, including the implementation of SOA governance. Our natural tendency would be to take a TSE approach to governance, where we hammer out organization-wide policies and put in place a systematic governance infrastructure to enforce those policies. While such an approach might be integral to a traditional architecture that we've implemented with TSE approaches, taking such a heavy-handed approach to SOA governance will stifle the implementation's emergent properties, most notably business agility. ZapThink calls this situation the "big brother effect," where excess governance can lead to the failure of the SOA initiative.

Instead, it is essential to take a CSE approach to SOA governance. Represent policies as metadata, and leverage SOA governance infrastructure to manage and enforce those policies in the context of Services. As the SOA implementation matures, leverage SOA governance best practices to support overall IT governance, and in turn, corporate governance. This approach to SOA governance "in the broad" not only improves the organization's governance overall, it is also essential to promoting the emergent properties of the SOA implementation. Your SOA initiative depends on it.

***About the author:*** *Jason Bloomberg is Managing Partner at Service-Oriented Architecture industry analysis and advisory firm ZapThink LLC. He is the newest RIA and enterprise mashup expert at SearchSOA.com.*

# An SOA practices checklist for implementation roadmaps

By Nitin Gandhi

Although SOA is an accepted means of organizing automation logic in such a manner that it fosters reuse, growth and interoperability throughout the evolutionary cycles of an enterprise, it is not itself a solution to domain specific problems. By applying practices, we address the unique considerations that come into play when having to phase service orientation into enterprise domains. Planning this staged approach leads to a controlled and scheduled delivery of key design specifications, governance plans and, ultimately, actual services - all part of a master implementation roadmap.

For example, it is difficult to think of the technical service interface as a business interface. The propensity is to use Web services as just another technical extension to a particular automation environment. However, if an IT group comes up with a set of rules to design service interfaces as business interfaces and enforces this practice across projects, the probability of creating services that represent "truly" service-oriented automation logic is significantly increased.

This document attempts to provide a set of concrete practices and considerations for implementing service-oriented architecture. While a number of attempts have been made to define a global SOA roadmap, it is a difficult proposition because every organization is unique with different priorities. The approach proposed here is to first identify all practices that may need to be in place and then to define business needs, risks, strategic objectives and all the other factors that funnel into a customized roadmap.

## SOA Practices

Let's start with identifying some of the key areas in which practices need to be considered as part of typical strategic SOA planning efforts:

1. Service Monitoring
2. Exception Management
3. Version Management
4. Service Management and Deployment
5. Policy and Security Considerations
6. Service Level Agreements
7. Service Directory

Recommended practices and considerations in each category are described below:

**Service Monitoring**

| | |
|---|---|
| **Availability** | The active availability of a service is generally defined within its accompanying service level agreement (SLA). Common availability considerations and guarantees include:<br>•When will the service be active and available? (This may be expressed in a formal schedule or timetable.)<br>•Will all dependencies for this service also be available during the service's scheduled availability? (If not, certain service capabilities may not be fully available, even if the service is active.) |
| **Logging** | The supporting service infrastructure should store (log) data about each service's access and usage patterns. This data is useful for troubleshooting, monitoring, and security control. Formal review processes may need to be in place to ensure that logs are routinely checked. This is especially important to raise awareness of certain usage trends that may indicate a need for infrastructure changes (usually associated with scalability and security measures). |
| **Auditing** | Auditing Auditing is the analysis and reporting of logged information. The data collected at this stage should answer questions such as:<br>•How exactly is a service being used?<br>•Who is using the service?<br>•What do the common request and response messages look like? (In other words, what are the most typical content exchange scenarios.)<br>In addition to better understanding how a service is being used, these reports will also highlight the capabilities of the service not being utilized. |
| **Performance Metrics** | Performance metrics and statistics for each Web service are kept so that services can be monitored to avoid potential runtime issues, such as performance bottlenecks and fault counts. It is frequently required (and recommended) that automated notification mechanisms be attached to collected statistics so that action can be taken well before an infrastructure's limitations are tested. |
| **Debugging & Tracing** | The ability to optimize and track service activity during development and after deployment is important, especially for maintenance purposes. This focuses more on the service hosting platform's ability to provide front-end tools that allow for targeted investigation of specific activities (or sub-activities) as opposed to common general reporting features. |
| **Synthetic Transactions** | The ability to utilize of synthetic (dummy) transactions is helpful to simulate service usage scenarios during development, testing, and debugging phases. Synthetic transaction modules can be configured to send periodic service requests according to predefined settings. |

### Exception Management

| | |
|---|---|
| **Error Trapping** | Runtime errors needs to be recorded and reported as efficiently as possible. For example, the recorded response times and any logged timeouts need to be monitored to ensure that the service is kept inline with corresponding SLA requirements. |
| **Root Cause Analysis** | It has become widely accepted that SOAP faults are not that useful. An SOA infrastructure needs to be able to drill down into the real reasons as to why a service is failing. Often this reveals issues such as code faults and hardware failure. With the right tools, a true root cause analysis (one that goes beyond just the vendor runtime) can be completed. |
| **Notification Services** | A services infrastructure should be able to send out notifications when specific events are triggered. This notification mechanism needs to be configurable so that it can apply to individuals and broadcast groups. |

### Version Management

| | |
|---|---|
| **Data Contracts** | Web services exchange data using pre-defined data models based on XML Schema. Changes to a published schema can therefore break the data contract and jeopardize all existing service consumers. If data contracts do need to be changed, a separate version control practice may need to be in place. This is primarily in consideration of the fact that an XML data representation architecture will often need to be maintained and evolved separately from the service layers that utilize its schemas. |
| **Message & Operation Contracts** | Service consumers are tightly coupled to the service provider through the service interface. However, extending a service contract without affecting existing service consumers may still be possible. A formal process is recommended even when just adding operations to an established WSDL definition. |
| **Endpoints (Addresses)** | The service consumer is tied to the service endpoint (address). If the address of the service changes all active service consumers will be compromised. Therefore, a separate endpoint or address management practice may be required. |

### Service Delivery

## Policy and Security Considerations

| | |
|---|---|
| **Policies** | Given the absence of an industry-wide policy expression language (cross-vendor implementations of WS-Policy still seem a distant reality), documenting and attaching policies to the technical service contract in a standardized manner can prove difficult. A common approach is to define policies as part of the accompanying SLA. Though the policies may be required, the SLA can often just request that service consumers adhere to policy rules. The enforcement of this practice sometimes requires that the service consumer owner sign a contract (or the SLA document) guaranteeing that all policies will be honored. |
| **Internal Dependencies** | A service may be implemented using numerous proprietary components, databases, and other system resources. Changes to these underlying dependencies can raise all kinds of runtime exceptions. Therefore, internal practices may be required to ensure the integrity of each individual service-level technology archi- |
| **Claims** | The service provider may be secured using a common identity technology, such as active directory roles and users. Any security changes in the infrastructure can also impact all consumers of a service. Changes to established security management processes and practices therefore become important considerations and may require formal reviews by committees. |
| **Service Retirement** | Services do not live forever. When services are deprecated, service consumer owners must be notified in time to adapt their environments. Often this may require a graceful expiry period during which both old and new versions of a service contract co-exist for a pre-determined amount of time (sometimes this period can extend up to six months). |
| **Dependency Analysis** | Service consumers can be classified as "known" and "unknown" consumers. Keeping a list of all the consumers associated with each service can simplify some versioning issues by allowing for organized notification. |

## Service Directory

| | |
|---|---|
| **Methodology** | A methodology practice (such as Thomas Erl's mainstream SOA methodology) defines how a service is moved through lifecycle stages and also establishes any new phases required specifically in support of SOA. In addition to shaping service logic in preparation for implementation, these formal approaches also need to provide governance processes in support of post-implementation management and evolution. |
| **Standardized Service Delivery Lifecycles** | Requirements gathering, business analysis, service modeling, design, and other delivery lifecycle phases need to be aligned with the service-orientation paradigm. This does not necessarily supersede object-oriented approaches, but the service implementation (and especially its contract) must reflect the design characteristics required to achieve the strategic goals behind service-oriented computing. |

| | |
|---|---|
| **Identity Store** | An identity store is a repository of users, their credentials and preferences. In order to access a secure service the service consumer must be validated against these identities. Various options exist for implementing an identity store including Active Directory or even a regular database. Ideally, an identity store is a standardized and centralized part of the overall infrastructure. Practices need to be in place to ensure it is consistently used and maintained. |
| **Authentication & Authorization** | The ability to verify the identity and permissions of service consumers raises a number of considerations, including how security claims from service consumers are validated and processed. |
| **Exchange Policy & Contracts** | This practice should address how policies and contracts are exchanged between a service provider, its consumers, and the owners of its consumers. |
| **Internet Perimeter Security** | The issues associated with securing Internet firewalls and Internet facing servers and ports are amplified when moving toward a technology environment consisting of a combination of internal and external Web services. Practices need to be in place to ensure that acceptable measures of security are always maintained, but also to provide a level of adaptability in response to unforeseen external access requirements that may be raised by newly published services. |
| **Usage Control & Metering** | Understanding the usage of a service may be important for a number of reasons. For example, we may need to bill the service consumer on a per usage basis or we may want to study service usage and load patterns. |
| **Transport Security** | Here we deal with how Web services traffic is secured on the wire. Of course this usually involves the positioning of SSL, but several WS-* specifications can also enter the discussion. |
| **Load Balancing** | Services with heavy or unpredictable volume loads often need to be dynamically load balanced across multiple servers. Various types of load balancing algorithms exist and services should be assessed individually to ensure that the most appropriate type of load balancing is always chosen. Sometimes, based on increased reuse or recomposition, services need to have their load balancing algorithms "upgraded" in response to new usage demands. |
| **Geo Clustering** | This consideration addresses issues that can arise from the physical proximity of clustered services within an enterprise. For example, service consumers may need to bind to services that are geographically closest to minimize server hops and latency issues. |
| **Web Services Interoperability** | While Web services are designed to interoperate, enforcing WS-I Basic Profiles is often the only way to guarantee interoperability, especially across disparate platforms and organizational boundaries. When building services with some of the new WS-* technologies and especially within vendor-diverse enterprises, extensions or custom variations of the WS-I Basic Profiles may need to be created (although their usage will likely be limited to internal, controlled environments). |

### Service Level Agreements

| | |
|---|---|
| **Quality** | A dedicated practice is often needed to ensure that a service provider continually meets the quality contract promised to its consumers. Carrying out this practice may require a separate "quality assuror" and "quality broker" role. |
| **Billing** | Whether a service is consumed within an enterprise or outside of the organization, a billing structure is often required to guarantee that the service owner (a department or the organization itself) is compensated. Usually, billing is based on a licensing agreement that ties into a per consumer or even a per usage payment model. |
| **Configuration Management** | As each service is promoted through various environments it needs to be configured and tested. As the number of services increase, managing the overall configuration management effort can be complex and will likely require formal processes. Configuration management of services can be much more challenging than traditional efforts. Therefore, new practices will likely need to be developed and configuration management implications may need to be expressed in the SLA. |

## The SOA implementation checklist template

| | |
|---|---|
| **Awareness & Discovery** | The service consumer must have knowledge of the existence of the service provider and the location of its contract. This awareness increases the overall reuse potential within an enterprise. UDDI exposes the required meta information in an industry standard manner. (Note that awareness alone does not enable a service consumer to invoke a service.) |
| **Publish Process** | Once a Web service is implemented, it will need to be registered in a local service directory or registry. Formal processes need to be in place to ensure that the registration and the documentation of associated metadata are performed in a consistent manner. |
| **Subscription** | It is ideal to allow service consumer owners to subscribe to service directory records. If changes to the service contract are added or should there be outages or other events associated with the implemented service (such as a change to billing rates), consumer program owners can then be easily notified. |
| **Service Owner Contact Information** | Service consumer owners or designers of potential service consumer programs should have the ability to contact the owner or custodian of a service for questions and recommendations. Especially in larger organizations, it is best if this contact information is published alongside the service directory records. |

| | |
|---|---|
| **Documentation** | The service registry should contain documentation to help owners of potential service consumers better interpret the service's capabilities. This information may exist in separately published documents or as annotations to the technical service contracts. |
| **Rating** | It can be beneficial for quality assurance purposes to allow service consumer owners to rate services and provide feedback. Typically, ratings are in response to guarantees and promises made in an SLA as measured against a service's actual runtime performance. |

There is no fixed order in which all of the previously listed practices need to be carried out. Each organization will have its own preferred sequence based on how their transition or migration plans are structured. To provide some guidance for going through these individual practices, a sample Excel template is provided. You can use this as a checklist to ensure that all considerations are eventually taken into account: SOA Implementation Template.xls.

This template provides the following supplementary columns:

1. Phases

Allocate a practice to a phase based on your organization's priorities and risks. By default, four phases are represented in the template. Some practices may intuitively fall into earlier phases. For example, it is often desirable to implement some or all practices related to security as early in the lifecycle as possible.

2. Products

There are two aspects to introducing any practice, namely technology and process. The technology factor may be either a product or some custom development software. Process, on the other hand, requires internalizing the technology so that it is used consistently and correctly. The process is represented by the practice. This column simply provides a means of referencing the technology associated with either applying or carrying out the practice.

3. Custom Development

Some practices can be better implemented through custom development efforts. For example, authenticating a service consumer against an existing membership repository will almost always require some extent of custom programming.

## Conclusion

As part of our exploration of SOA roadmap planning we've accumulated a list of 37 recommended practices and considerations. When putting together a serious SOA transition plan, you will always come up with additional items for this list. Every organization has unique requirements that relate to the distinct nature of their existing technical environments. Many of these unique requirements will originate from the legacy systems that already comprise a

fixed part of the infrastructure and impose very specific constraints on the SOA project plan. Either way, starting the process of charting your SOA roadmap with a simple checklist, such as the one provided here, will ensure that you address fundamental considerations, which is the first step to establishing formal practices.

***About the author:*** *Nitin Gandhi is an independent software consultant. In the past, Nitin has held senior | consulting roles on several large projects in the insurance, financial services, government, and healthcare sectors. He is also a member of several OASIS Technical Committees and an author and book reviewer for Prentice Hall.*

# SOA infrastructure: Mediation and orchestration

By Satadru Roy

There seems to be no end to the hype surrounding SOA. Vendors and analysts are only too happy to fuel the media buzz by touting their own products and research recommendations. Some claim their products go a long way towards enabling SOA while others maintain that SOA is an architectural pattern, albeit one also better implemented with their offerings.

No matter what claims are made in the marketplace, it is important to realize that any one SOA infrastructure product represents a means to an end, not necessarily an end in itself. Platform software can provide the necessary backbone for an enterprise-wide services fabric, but the choice of the solution should be solely dictated by the requirements of the organization.

One such software product is the now ubiquitous enterprise service bus (ESB). Few recent innovations have generated as much excitement and confusion. All ESB vendors claim that their enterprise service bus implementations provide a key piece of the service-oriented technology architecture. However, most still struggle to agree on what actually constitutes an ESB, let alone what capabilities the common ESB should have.

Many other vendors are jumping on the ESB bandwagon. Traditional EAI products are being rebranded as ESBs and pure-play messaging vendors are doing the same by implementing a SOAP and WS-* stack on top of their existing solutions. Some vendors are even offering not one, but two sets of ESBs!

Imagine, amidst all of this, that you are the IT architect charged with the responsibility of 'SOA-enabling' your applications. What should you do? It's simple really, just go back to your business requirements as they can be fulfilled by service mediation and orchestration.

## Service mediation

The concept of mediation is nothing new. In traditional object-oriented literature, a mediator is a well-known pattern that promotes loose coupling by keeping objects from referring to each other explicitly and lets you vary their implementation independently.

Replace the word objects in the above sentence with services and you will have a good starting definition of service mediation. However, service mediation in an SOA also goes much further than merely keeping services from referring to each other (although that in itself is a good start). In the world of services, where the emphasis is more on technology-neutral, XML-based message interactions, a number of things become possible once you put a mediator in between a service producer and a service consumer. For example:

## Transport protocol conversion

Often, the service provider offers a service based on a certain transport protocol (such as HTTP) whereas the service consumer is only capable of communicating over a different protocol (let's say MQ). Alternatively, perhaps a

synchronous-asynchronous bridge needs to be built connecting a service provider and consumers over protocols such as HTTP and Java Messaging Service (JMS). Technically speaking, JMS is not a transport protocol, but rather a vendor-neutral set of messaging APIs. As illustrated in Figure 1, asynchronous Web service implementations often expose "SOAP services over JMS," but under the covers the transport protocol used is vendor-specific, such as MQ or Tibco RV.

This is quite common in environments where some legacy assets may have been service-enabled, but other applications cannot consume those services because of a transport protocol mismatch. Rather than building yet another protocol adapter or implementing a synchronous-asynchronous bridge on top of the service provider implementation, it is better to let a mediator handle these differences and do the necessary translation. The service developers can then focus on building the service logic, letting the infrastructure software handle mediation responsibilities.

## Data format conversion

Even within the same organization the business entity definitions may vary from one organizational unit to another. The finance department may have a specific customer structure that could be different from the definition of a customer from a billing perspective. In this situation a customer-related service in billing cannot consume a service from finance without having to account for the data model differences.

You can try and force everyone to adopt a common definition (as with the proposed Canonical Data Model), but it may be impractical to do so in a large organization. What is preferable is to let a mediation component handle the transformation between one format and another. It may even be possible to let service interfaces deal only with a canonical data model, but service consumers will then need to be built with mediation components to transform their data format to the one expected by the providers.

## Service policy enforcement

Having a mediator handle the interaction between the service provider and the consumer allows it to easily intercept XML traffic between the two and take necessary steps to ensure policy compliance -- in the form of appropriate auditing, logging and security monitoring, for example.

Once again, delegating these cross-cutting concerns to a common mediation component frees up the service developer from having to fulfill their requirements in the service level. Of course, if this is the only mediation requirement you are faced with it may be more cost-effective for you to implement a cross-cutting solution of your own based on technologies such as aspect-oriented programming (AOP).

## Service processor pipeline

The pipes and filters architectural pattern describes how general-purpose message pre-processors and post processors can be built to enhance a message processing cycle. A mediation platform can compose such filters in a declarative fashion for a service invocation allowing us to vary the invocation sequence of these pre and post processing components by changing the configuration of the filters.

Examples of these pre-processors include service routing based on message content, context or business rules, message enrichment, message encryption/decryption, message de-duplication, and so on (see Figure 1). The key value-add of mediation here is not so much in being able to provide such processors which developers have to build anyway but to facilitate an arbitrary composition of these processors in a declarative fashion.

## Service invocation and dispatch

Remember the part about loosely coupling the service provider and the consumer? If the service consumer interacts directly with the service provider it may be difficult to change the interface (not the implementation, which should be independent of the interface anyway) of the service provider without impacting the service consumer. In other words, the service consumer forever remains tightly coupled with the service provider.

However, if the service consumer interacts only with the go-between mediator and that mediator service does not change its interface the mediator can transparently dispatch the service invocation to different versions of the service, possibly even with different service interfaces. Obviously, the mediation (the necessary translation) in this case still needs to be built, but the important point is the independence achieved between service consumer and provider.

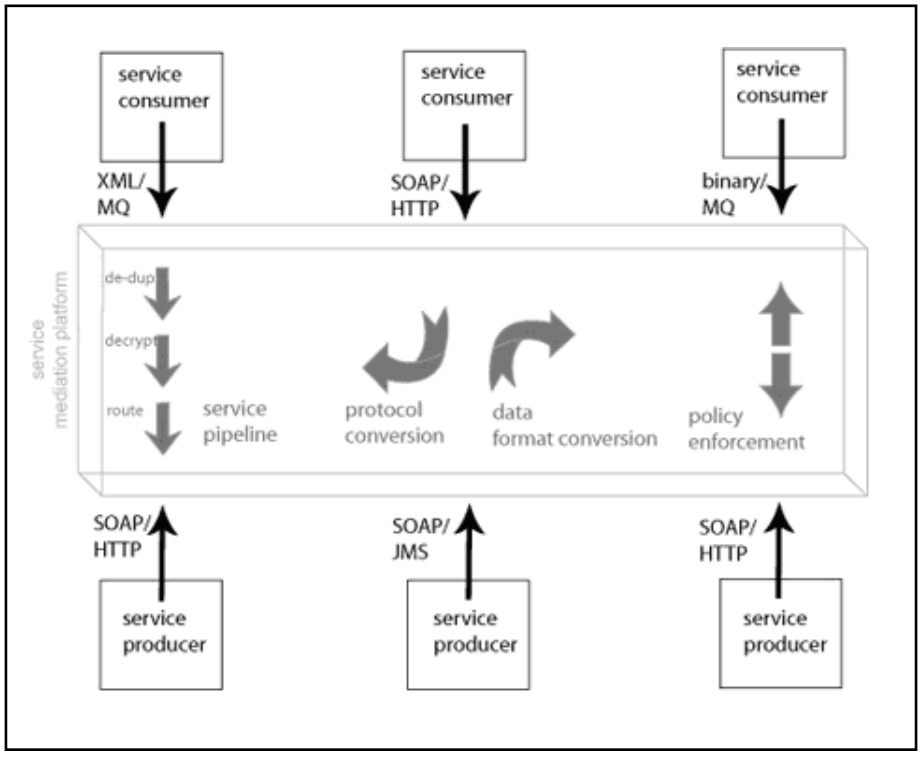


*Figure 1: This diagram illustrates how the previously described requirements can be handled by a common mediation layer.*

The enterprise service bus can be viewed as a mediation platform supporting some or all of the capabilities we just described. In fact, you will likely find even more features in some of the higher end ESB products.

## Service orchestration

Unlike service mediation, which essentially establishes a broker component as part of contemporary middleware, orchestration provides capabilities associated with the composition and coordinated execution of services. Orchestration technology is also middleware-based and can establish a highly centralized part of the architecture that governs the design of business process definitions, as well as the execution of business process logic.

## Service layers

Before we describe orchestration in detail it is important to understand the concept of service layers. Note that this discussion serves only as an introduction to service layers and domain abstraction.

Service layering is a separation of different categories of services (called service models) based on the problem domain they operate in. Business services, for example, focus solely on business logic whereas application or utility services are more technology-oriented. Examples include services that facilitate solving system-level problems such as security and auditing. Figure 2 (reprinted with permission from "Service-Oriented Architecture: Concepts, Technology, and Design" by Thomas Erl) illustrates the relationship between common service layers.

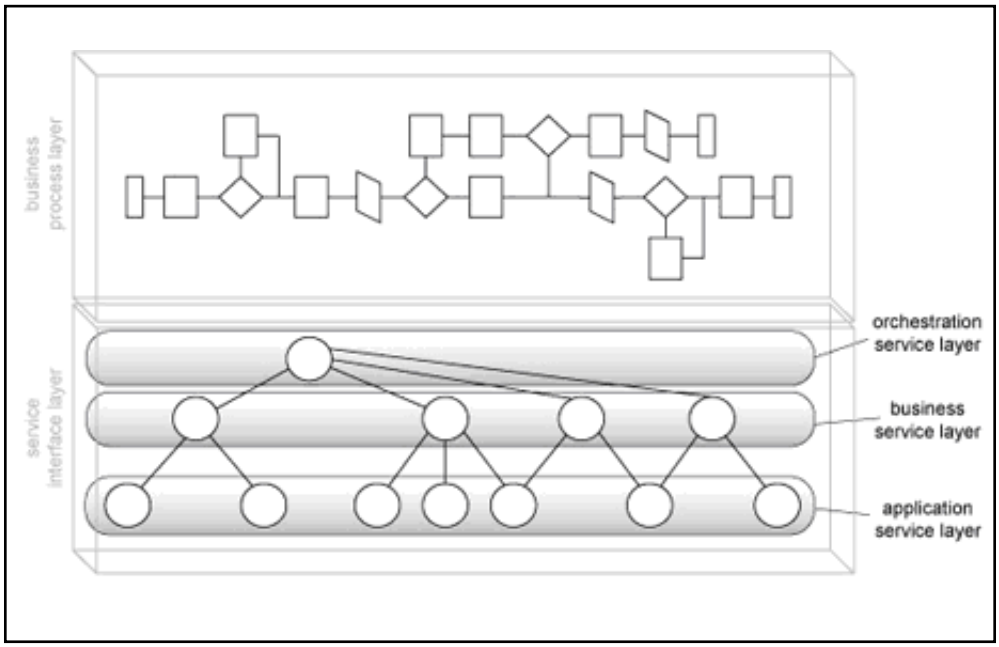


*Figure 2: The three common service layers that are implemented through the use of service models.*

Business service implementations are likely to make use of utility services, and possibly other business services as well – this is known as service composition. One of the fundamental principles of service design is that a service should be built such that it can participate in a composition, if necessary.

Service orchestration is an extension of the service composition model where a parent service layer performs an orchestration of many business and utility services by controlling workflow logic and invocation sequences.

For example, a mortgage solution may need to make use of credit score services offered by external credit agencies. The calls to the credit agencies can happen in parallel, but the results of the calls may need to be aggregated and analyzed to arrive at a decision. Such complex workflow logic often involves timeouts and exceptions and is typically modeled as business process definitions.

## Service orchestration and BPEL

The key factor in determining whether you need a true service orchestration platform is figuring out if your service and business process interactions require a complex pattern of invocations as described above. If they do, then you might want to consider implementing them in a service orchestration platform such as a Business Process Execution Language (BPEL) engine.

BPEL has emerged as the most promising standards for Web service orchestration and even though it is often mentioned in the context of modeling executable business processes, the service orchestration aspects of a BPEL engine cannot be overemphasized. It is important to keep in mind that BPEL has limited capabilities in terms of modeling real-world complex workflows and as such it may be better to focus on BPEL more as a service orchestration vehicle than a full-blown workflow modeling tool.

## BPEL and ESBs

Some ESB products come packaged with a BPEL implementation while others do not. The ones that don't may provide a graphical wizard based on service mediation flow constructs to model simple short running service flows. If you are tempted to model all your short running, synchronous service flows and interactions via such tools think long and hard about the true mediation requirements you have. If you can't come up with many, then a code-based implementation, rather than investing in expensive middleware, may be a more prudent choice.

## Additional orchestration capabilities

Provided below is a brief list of other features and capabilities provided by commercial orchestration products:

- Management of synchronous, stateless (short-running) flows with complex interaction patterns, such as parallel invocations and time-bound executions with possible cancellations.

- Management of long running, stateful, asynchronous flows where processes may need to wait for events to fire. For asynchronous invocations, callbacks with correlated request-response patterns may also need to be supported.

- Parallel service invocations with support for AND-join and OR-join conditions.

Orchestration technology has been around since the days of EAI. Therefore, it has matured and become relatively commonplace. However, when assessing an orchestration product for use within a service-oriented technology architecture, it is well worth investigating the extent of support it provides for contemporary Web services technologies and the common principles of service-orientation.

## Future trends

Even as commercial ESBs acquire more sophistication, some open source mediation frameworks are gaining traction and are solutions you might want to consider, especially if you are building using the Java Enterprise Edition (JEE) platform. The most popular of these frameworks offer mediation functionality such as protocol conversion, data transformation and service routing. However, keep in mind that these may not provide the rich user interface offered by the commercial products.

The other interesting trend is the very recent emergence of what can be characterized as hardware ESB appliances. These are primarily positioned as "edge-of-the-enterprise" XML security devices, but they also offer strong mediation capabilities. Furthermore, because the implementation consists of actual dedicated hardware, these devices can offer superior processing performance. However, it remains to be seen if they gain widespread adoption or even supplant the software ESB solutions. We will look at these appliance solutions in more detail in a future article when we discuss SOA security infrastructure software.

## Conclusion

In this article we covered common mediation and orchestration requirements and also addressed how ESB and BPEL implementations can help satisfy these requirements. In our next instalment we will look at additional types of SOA infrastructure requirements and explore how they are currently being incorporated within the overall SOA technology landscape.

***About the author:*** *Satadru Roy is a Technology Architect who has worked for several Fortune 500 companies in the manufacturing, automobile, and telecommunications industries. Satadru has extensive background in CORBA, Java, JEE, and Web services technologies. His main interests and areas of specialization are middleware and integration technologies (such as TP monitors, integration brokers, workflow engines, etc.).*

# Resources from Oracle

« **White Paper:** **Oracle WebLogic Server - A Solid Foundation for SOA**

« **White Paper:** **Oracle IT Modernization Series Modernization: The Path to SOA**

« **White Paper:** **Oracle WebLogic Server - A Solid Foundation for SOA**

**Find an Oracle Partner near you:**

**Geographic Coverage:**
•AR •AZ •CO •ID •KS
•MO •MT •ND •NE
•NM •SD •WY

### About Beloit Solutions Group

Beloit Solutions Group is a client-focused provider of information technology based business solutions that focus on increasing productivity and reducing costs. We believe in evolving, not revolutionizing our client systems, structure, and assets to maximize return on investment. Beloit focuses on IT Strategy and Planning, Business Process Management, Service Oriented Architecture, Application Performance Management. Portals/Corporate Communication and Identity/Access Management. By staying current on the latest innovations to solve tough business problems and using a flexible approach to our methodology, we are able to design and implement successful strategies for our clients.
www.beloitsg.com

**Geographic Coverage:**
•AL •CA •CT •FL
•GA •MA •ME •MS
•NC •NH •NV •NY
•OR •RI •SC •TN
•UT •VT•WA

### About Idhasoft

Idhasoft is a software products and IT services provider serving enterprise customers. We offer solutions to industry segments in the United States, Europe and Asia. We provide services to industry segments including Banking and Financial Services, Healthcare, Insurance, Manufacturing, Public Sector, Retail and Telecommunications. With a reputation for unparalleled service, Idhasoft offers a comprehensive range of technology solutions. From software application development, strategic IT consulting and recruitment process outsourcing to enterprise-wide implementation of third-party vendors, Idhasoft is second to none as the preeminent, complete IT partner. With expertise in robust technologies, such as Oracle and SAP, Idhasoft excels at customer satisfaction, always completing projects ahead of schedule and under budget.
www.idhasoft.com


**Geographic Coverage:**
•IA •IL •IN •KY
•MI •MN •OH •WI

## About PSC Group, LLC

PSC Group, LLC is an information-technology and professional services consulting firm specializing in providing business process solutions to clients that view IT as a strategic resource. We partner with IBM, Microsoft, Oracle, and other leading providers (including Open Source), to deliver Agile PBM, back-end application and ERP integration, middleware, portal, and collaboration-based technologies and architectural services for the Services, Insurance, Manufacturing, Distribution, and Financial Services sectors.
www.psclistens.com


**Geographic Coverage:**
•LA •OK •TX

## About TSA

TSA was founded as Technical & Scientific Application, Inc. in 1986 to serve the Oil & Gas and engineering community as an exclusive supplier and a repair facility for Hewlett-Packard equipment. As a trusted HP Oracle - Solution Elite Partner, TSA is a full service business partner, providing the complete product offering of HP equipment and business solutions. Our goal at TSA is to leverage our 23 years experience and our strong partnership with Hewlett-Packard and Oracle to develop long-term relationships with our clients by delivering innovative and reliable IT solutions.
http://www.tsa.com


**Geographic Coverage:**
•DE •MD •NJ •PA
•VA •WV

**About Turnberry Solutions**

Turnberry Solutions is an IT Professional Services firm with a proven track record of delivering large-scale Portal, BPM and Software Integration projects. Over the past decade, we have assembled a team of distinguished engineers and project managers who provide services in the following areas:

- Business and IT Strategy
- Requirements Analysis and Project Management
- Software Architecture, Design and Development
- Quality Assurance and Operational Support

http://www.turnberrysolutions.com