

The SOA Magazine

Feature Article



SOA Infrastructure: Mediation and Orchestration

by Satadru Roy

Published: August 28, 2006 (SOA Magazine Issue I: September/October 2006, Copyright © 2006)

[Download this article as a PDF document.](#)

Abstract: Services are the foundational layer of a service-oriented architecture but building service interfaces and exposing them to service consumers without a supporting infrastructure can result in a brittle, hard-to-sustain technical environment that leads us back to the problems service-orientation was supposed to fix.

A services development platform such as a Web services runtime stack or an application server may no longer be enough to support the complex infrastructure requirements of a contemporary services ecosystem, such as mediation, orchestration, policy-based security enforcement, and services management. However, not all IT organizations have the need to meet all of these requirements. Careful analysis is necessary before decisions are made to invest in expensive infrastructure upgrades, especially as vendors continue to release a wide array of products often with confusing overlap in functionality and features, each claiming to act as an 'SOA-enabler'.

In a four-part series we will examine the most common SOA infrastructure requirements, their various degrees of complexity and how organizations can take an incremental approach towards SOA infrastructure software adoption. In the first article, we tackle two categories of SOA infrastructure requirements: mediation and orchestration. We'll look at a number of solutions that can help address these requirements and conclude by analyzing industry trends that IT decision makers should be aware of as they prepare a long-term SOA strategy.

Introduction

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 [REF-1].

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 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 [REF-2]) 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 [REF-2] 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.

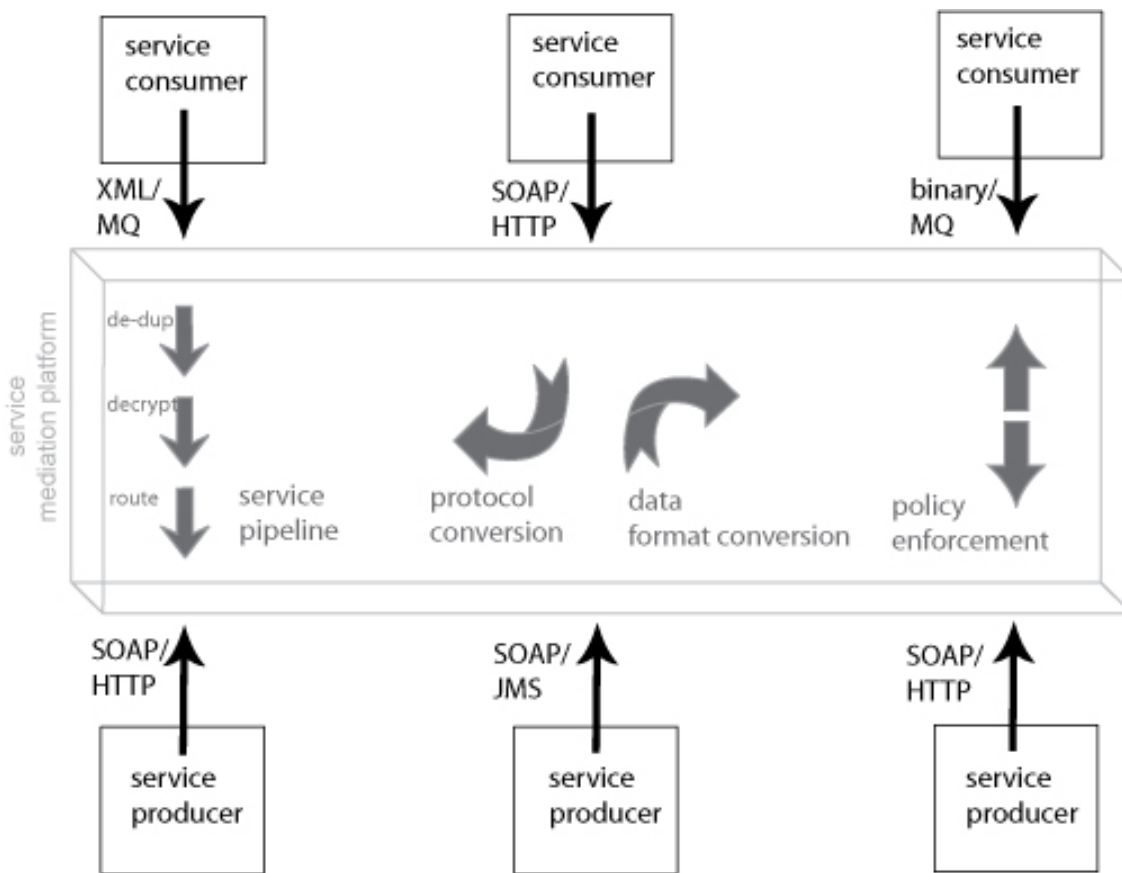


Figure 1: This diagram illustrates how the previously described requirements can be handled by a common mediation layer.

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. For more details I recommend you refer to [REF-3].)

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 [REF-3]) illustrates the relationship between common service layers.

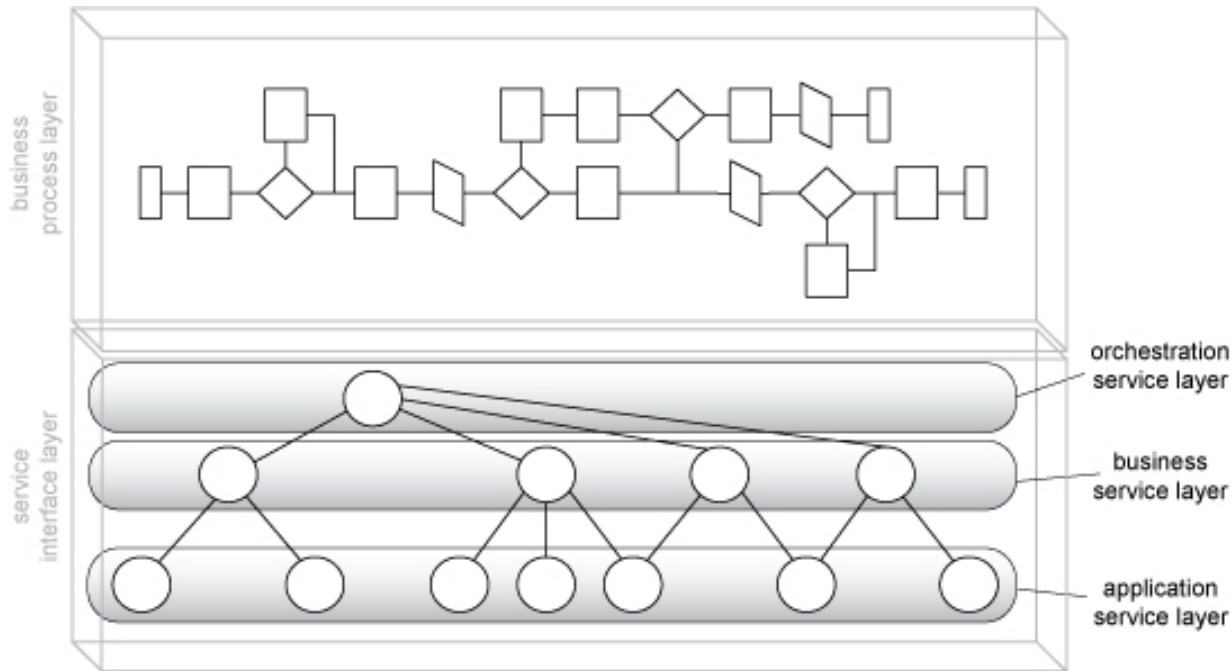


Figure 2: The three common service layers that are implemented through the use of service models.

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 [REF-4,5] 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.

References

[REF-1] Design Patterns – Elements of Reusable Object-Oriented Software by Gamma, Helm, Johnson and Vlissides, Addison Wesley

[REF-2] Enterprise Integration Patterns – Designing, building, and Deploying Messaging Solutions by Hohpe, Woolf, Addison Wesley

[REF-3] "Service-Oriented Architecture: Concepts, Technology, and Design", Thomas Erl; Prentice Hall, 2005; ISBN-0-13-185858-0, available at <http://www.soabooks.com>

[REF-4] The Mule Framework, available at <http://mule.codehaus.org>

[REF-5] The ServiceMix Framework, available at <http://servicemix.org>

[REF-6] ESB roundup part one and two, available at <http://www.infoq.com>

[REF-7] Enterprise Service Bus by David Chappell, Oreilly

[REF-8] The BPEL specifications, available at <http://www.oasis-open.org/apps/org/workgroup/wsbpel>

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL



[home page](#) [past issues](#) [official book site](#) [legal disclaimer](#)

Copyright © 2006 SOA Systems Inc. All Rights Reserved