

[▶ subscribe](#)[▶ contact us](#)[▶ submit an article](#)[▶ rational.com](#)[▶ issue contents](#)[▶ archives](#)[▶ mission statement](#)[▶ editorial staff](#)

▶ **The role of the service-oriented architect**

by **[Jason Bloomberg](#)**

Senior Analyst
ZapThink LLC

Web services have moved beyond the hype stage and are now a reality for many enterprises. Hundreds of companies have built Web services pilot projects, proving that this most recent evolution of distributed computing technology can reduce integration and development costs substantially. Forward-looking enterprises are now looking to take the next step and leverage the power of Web services strategically across the enterprise.



This next step means moving beyond simple point-to-point applications of Web services to a broad application of Web service technologies, both within the enterprise and increasingly among business partners. Making such a transition requires more than a simple change in programming practices. The broad application of Web services technologies requires an architectural change -- a move to loosely coupled, standards-based, service-oriented architectures. This new architectural approach requires a different perspective on the role of IT in the organization, and a new kind of enterprise professional: the service-oriented architect.

Just as Web services evolved from traditional distributed computing techniques, the practice of service-oriented architecture evolved from the practice of enterprise architecture. The potential rewards are enormous for enterprises that understand this evolution and make the move to such architectures. Finally, distributed computing technology promises to be flexible and nimble enough to respond to business needs and provide the business agility that companies have craved for so long, but which has always been just out of reach.

What is a service-oriented architecture?

Service-oriented architectures represent an approach to distributed

computing that treats software resources as services available on a network. Such architectures are nothing new; CORBA and DCOM are familiar examples.¹ However, these older examples of service orientation suffered from a few difficult problems. First, they were *tightly coupled*, which meant that both ends of each distributed computing link had to agree on the details of the API (application program interface). A code change to a COM (component object model) object, for example, required corresponding changes to the code that accessed that object. Second, such architectures were *proprietary*. Microsoft unabashedly controlled DCOM, and although CORBA was ostensibly a standards-based effort, in practice, implementing a CORBA architecture typically necessitated working with a single vendor's implementation of the specification.

Web services improve upon these DCOM and CORBA weaknesses. What's new about today's service-oriented architectures built with Web services is that they are *standards-based* and *loosely coupled*. The reliance upon universally accepted standards such as XML (extensible markup language) and SOAP (simple object access protocol) provides broad interoperability among different vendors' solutions, and loose coupling separates the participants in distributed computing interactions, so that modifying the interface of one participant in the exchange doesn't break the other. The combination of these two core principles means that companies can implement Web services without having any knowledge of the consumers of those services, and vice versa. The service-oriented architectures discussed in this article are the standards-based, loosely coupled kind, which we'll refer to as "SOAs."

The power and flexibility that SOAs potentially offer the enterprise are substantial. If an organization abstracts its IT infrastructure so that it presents its functionality in the form of coarse-grained² services that offer clear business value, then the consumers of those services (whether they are at the same company or one of that company's business partners) can access them independent of the underlying technology that supports them. Furthermore, if service consumers can discover and bind³ to available services while they are active, then the IT infrastructure behind those services can offer extraordinary flexibility to the businesses that invoke them.

Achieving these levels of power and flexibility, however, is a difficult task that requires a new approach to architecture: the *practice* of SOA -- in other words, what service-oriented architects must do to build SOAs.

The principles of SOA

Because SOA is a type of enterprise architecture, it begins with the needs of the enterprise. However, the difference between SOA and other approaches to enterprise architecture is in the business agility that SOA offers. *Business agility* is the ability of a company to respond quickly and efficiently to change, and to leverage change for competitive advantage. For the architect, building an architecture that enables business agility means creating an IT infrastructure to meet as-yet unknown business requirements -- a situation that throws traditional IT planning and design out the window.

To meet the needs of the agile enterprise, then, the practice of SOA has the following core principles:

- **The business drives the services, and the services drive the technology.** In essence, services act as a layer of abstraction between the business and the technology. The service-oriented architect must understand the dynamic relationships between the needs of the business and the available services on the one hand, as well as the technical underpinnings that offer the layer of abstraction required by the services on the other.
- **Business agility is the fundamental business requirement.** Instead of dealing with concrete requirements from business, SOA considers the next level of abstraction; the ability to respond to changing requirements is the new "meta-requirement." The entire architecture -- from the hardware on up -- must reflect the business agility requirement, because any bottleneck in the SOA can torpedo the flexibility of the entire IT environment.
- **A successful SOA is always in flux.** To visualize how an SOA is supposed to work, it's better to think of a living organism than of the traditional "building a house" metaphor that gave software architecture its name. The normal state of affairs is an IT environment that is undergoing constant change, so the work of the service-oriented architect is never done.

For the architect used to the notion of building a house, adapting to the idea of tending to a living, dynamic organism requires a new way of thinking. Fortunately, the foundations of SOA rely upon familiar architectural principles.

The foundations of SOA: Model-Driven Architecture and agile methodologies

There are two increasingly popular movements in IT -- one architectural, the other methodological -- and both have something to offer the service-oriented architect. The first movement is *Model-Driven Architecture* (MDA), championed by the Object Management Group (www.omg.org), the same body that looks after CORBA. The current incarnation of MDA states that architects should begin with a formal model of the system being built, ideally in UML (Unified Modeling Language, which is also shepherded by the OMG). MDA begins with a platform-independent model that completely represents the functional requirements and use cases of system users. From this platform-independent model architects can derive whatever platform-dependent models they need in order to specify the design of the system under construction. These platform-dependent models are so detailed that they can be used to automatically generate the implementation code itself.

One core strength of MDA is that designs for systems are fully specified; there is little leeway for misinterpretation when the systems are built, and the models can be used to generate working code. However, MDA in its

current version has some limitations. First, it assumes that the business requirements are fully specified before the model is built, which is not the case in the typical dynamic business environment. Second, MDA doesn't offer a feedback loop: If developers need to diverge from the models, there's no set way to keep the models up to date. As MDA matures, there's a good chance that these issues will be resolved, but for the time being, it's important for the architect to understand and allow for MDA's limitations.

The second foundation of SOA is the *agile methodology* (AM) movement, most notably represented by Extreme Programming (XP). Agile methodologies such as XP (extreme programming) provide a flexible, iterative process for building software systems in environments in which requirements are unknown or in flux. XP requires that a user representative work closely with the development team, helping to write tests that guide the developers' daily work. All members of the team pitch in on the design, which is kept as minimal and informal as possible. The goal of agile methodologies is to build just what the user wants, avoiding extraneous work on artifacts such as formal models. AM's core strength lies in its agility -- the ability to deal with changing requirements. AM's main weakness is its limited scalability. Although XP works well for small teams and projects of moderate size, as the project scope grows, it becomes more difficult for team members to have a solid grasp of all aspects of the project without a concrete, comprehensive plan to work from.

On the surface, MDA and AM appear to be contradictory. MDA assumes fixed requirements, whereas AM tackles changing requirements with an iterative approach; MDA centers on formal models, whereas AM eschews them. However, at the risk of being iconoclastic, we're going to take certain elements from each of these approaches and put them together into a coherent architectural practice. To get us started, Figure 1 shows how MDA and AM fit into SOA.

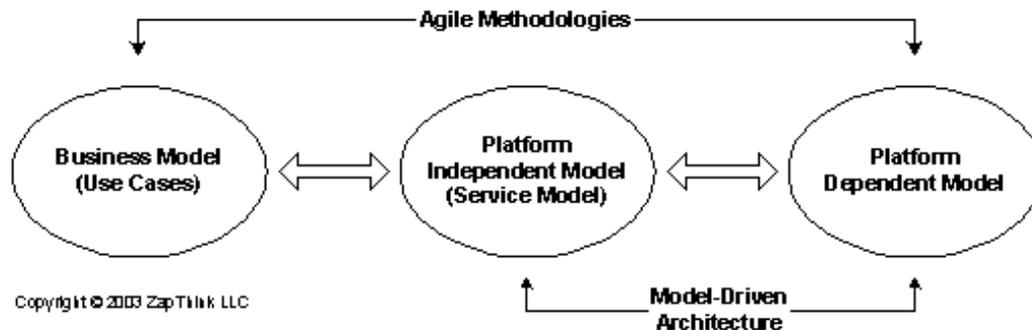


Figure 1: The SOA meta-model

The ovals in Figure 1 represent the three levels of abstraction in an SOA, following the first principle of SOA from above: The business drives the services, and the services drive the technology. In other words, the three ovals represent a model of models, or a "meta-model." AM relates the business model directly to the implementation, as represented in the platform-dependent model. MDA, on the other hand, does not separate

the business model from the platform-independent model, but rather takes the platform-independent model as its starting point. SOA must connect these models, or levels of abstraction, into a single architectural approach. We will make this connection by looking at the five-view approach to architecture.

The 4+1 view approach to SOA

Enterprise architects find their profession both challenging and gratifying because they must consider IT from many perspectives. Philippe Kruchten distilled these perspectives into the 4+1 view model of architecture, which we apply to SOA in Figure 2.

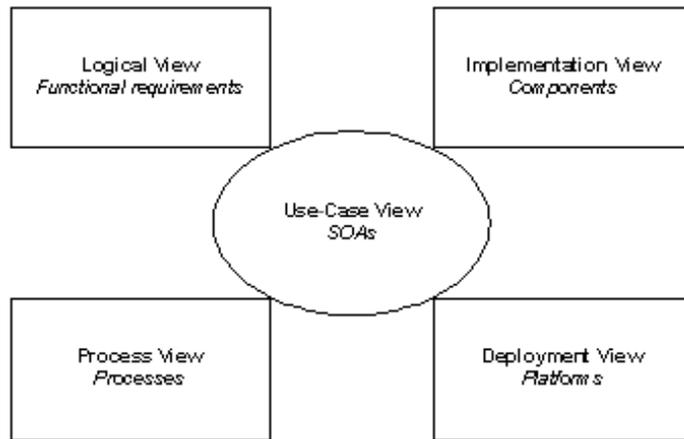


Figure 2: The 4+1 view approach to SOA

The four rectangles in Figure 2 represent different ways of looking at an architecture, or the perspectives of different stakeholders. The fifth view, the *use-case view*, overlaps with the other views and plays a special role with regard to the architecture. The *deployment view* maps software to the underlying platforms and associated hardware, the way that systems specialists view the architecture. The *implementation view* describes the organization of the software code, and is the view favored by programmers. Business analysts work with the *process view*, which addresses the software's runtime issues. Finally, the *logical view* represents the users' functional requirements. In the case of SOA, the service-oriented architect must be able to connect the users to the services, and the services to the underlying technology, following the threads of use cases in the *use-case view*.

To show how the service-oriented architect must work with each of these views, let's put them in the context of the SOA meta-model, as shown in Figure 3.

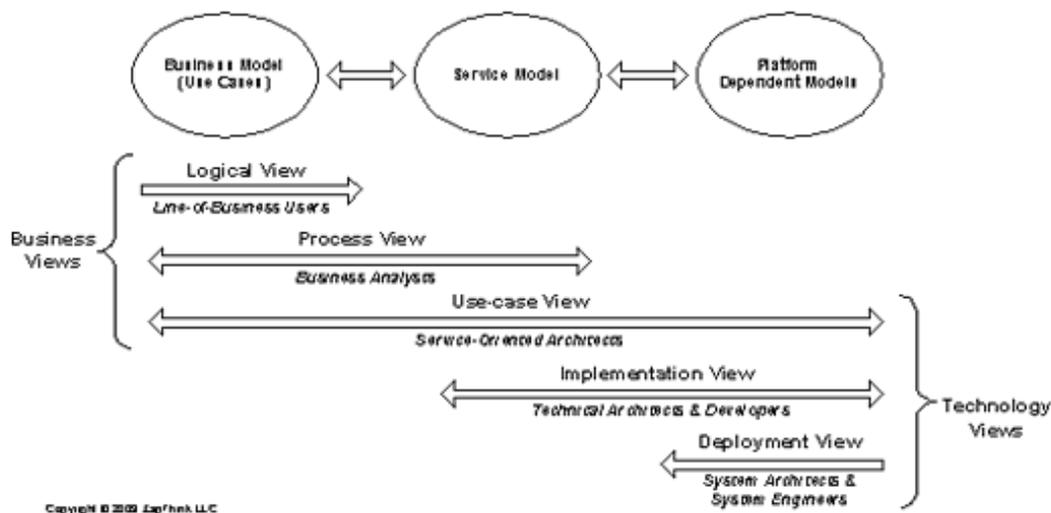


Figure 3: The practice of SOA

Figure 3 shows the two overlapping realms of SOA: the business realm represented by the business model and service model ovals on the left, and the technology realm represented by the service model and platform-dependent model ovals on the right. The service model thus becomes the point of contact between the business and technology realms, and acts as the conduit for communication across the enterprise. Business users who use the logical and process views work with coarse-grained business services, orchestrating them into processes as needed, depending on the fluctuating requirements of the business. Technologists, on the other hand, work to build and maintain the abstraction layer between the services and the underlying technology. The central model, representing the services themselves, acts as the axis around which the business moves.

The SOA meta-model inherits the platform-independent and platform-dependent models from MDA, but adds AM's interaction with the user as well as an iterative, agile feedback loop, represented by the two-way arrows between the ovals. Likewise, the meta-model solves AM's scalability issue by introducing the intermediate level of abstraction provided by the central service model. Users can thus deal with the day-to-day concerns of the business, reflecting any changing requirements in the service model. The technologists then can respond to these changing requirements quickly and effectively, because the underlying technology is model-driven.

What's different about the practice of SOA and more traditional approaches to enterprise architecture is the way it enables agility. Remember that the third principle of SOA states that SOAs are always in flux. This environment of constant change is the cornerstone of the practice of SOA shown in Figure 3. The stakeholders shown in this figure continue to effect changes throughout the architecture on an as-needed basis. The line between design time and runtime blurs, as the technologists respond to changing business requirements as a normal part of day-to-day operations.

The missing pieces

This article provides a broad, high-level framework for the service-oriented architect, showing how elements of MDA and AM provide tools for creating and maintaining SOAs. Nevertheless, many pieces are still missing from the SOA puzzle -- pieces that software vendors and professional services organizations must provide. In the business realm, vendors must offer service-oriented business process, workflow, and service orchestration tools and services. Modeling tools must be available that can adequately reflect business services in an agile, platform-independent way. The technologists must have tools that can generate code from models, and update those models when the code changes. And finally, vendors must offer SOA enablement software that enables service-oriented architects to build and maintain the level of abstraction between the services and the underlying technology in a reliable and scalable way. Fortunately for today's enterprises, such products will be coming to market soon.

Another missing piece is a full representation of the SOA meta-model in terms of MDA. Today's incarnation of MDA presupposes fixed business requirements, but the business model in Figure 1 represents a flexible, business-oriented interface to the SOA that enables business users to interact with the architecture. In other words, the MDA does not have the iterative feedback loop to the user that AM provides to SOA. This limitation, however, is with the current version of MDA, not with the principles underlying it. There is an additional, higher level of abstraction for MDA known as the *Meta-Object Facility* (MOF). MOF is a model for building meta-models like the SOA meta-model -- a meta-meta-model, if you will. Theoretically, the MOF provides the flexibility necessary for tools vendors to implement meta-models like the SOA meta-model in software.

The most important missing piece, however, is the top-down approach to SOA outlined in this article. Most of today's thinking about Web services is bottom up: "Here's how to build Web services; now let's use them for integration." That approach is a great first step, because Web services can dramatically lower the cost of integration, which is a story today's technology managers love to hear. As the economy improves and IT finally pulls out of the doldrums, however, companies will increasingly look to IT to offer strategic value to the organization. Service-oriented architectures provide the framework that will enable IT to offer this value in the form of business agility.

[Editor's Note: An earlier version of this article appeared in *Application Development Trends* magazine.]

Notes

¹ For a comparison of SOAs like CORBA and DCOM to architectures built with Web services, see http://www.therationaledge.com/content/sep_01/f_webServices_jb.html.

² See the article at <http://www.zapthink.com/flashes/09162002Flash.html> for a definition and discussion of coarse granularity.

3The article at

http://www.therationaledge.com/content/sep_01/f_webServices_jb.html

explains service discovery and binding.



For more information on the products or services discussed in this article, please click [here](#) and follow the instructions provided.

Thank you!

Copyright **Rational Software 2003** | [Privacy/Legal Information](#)