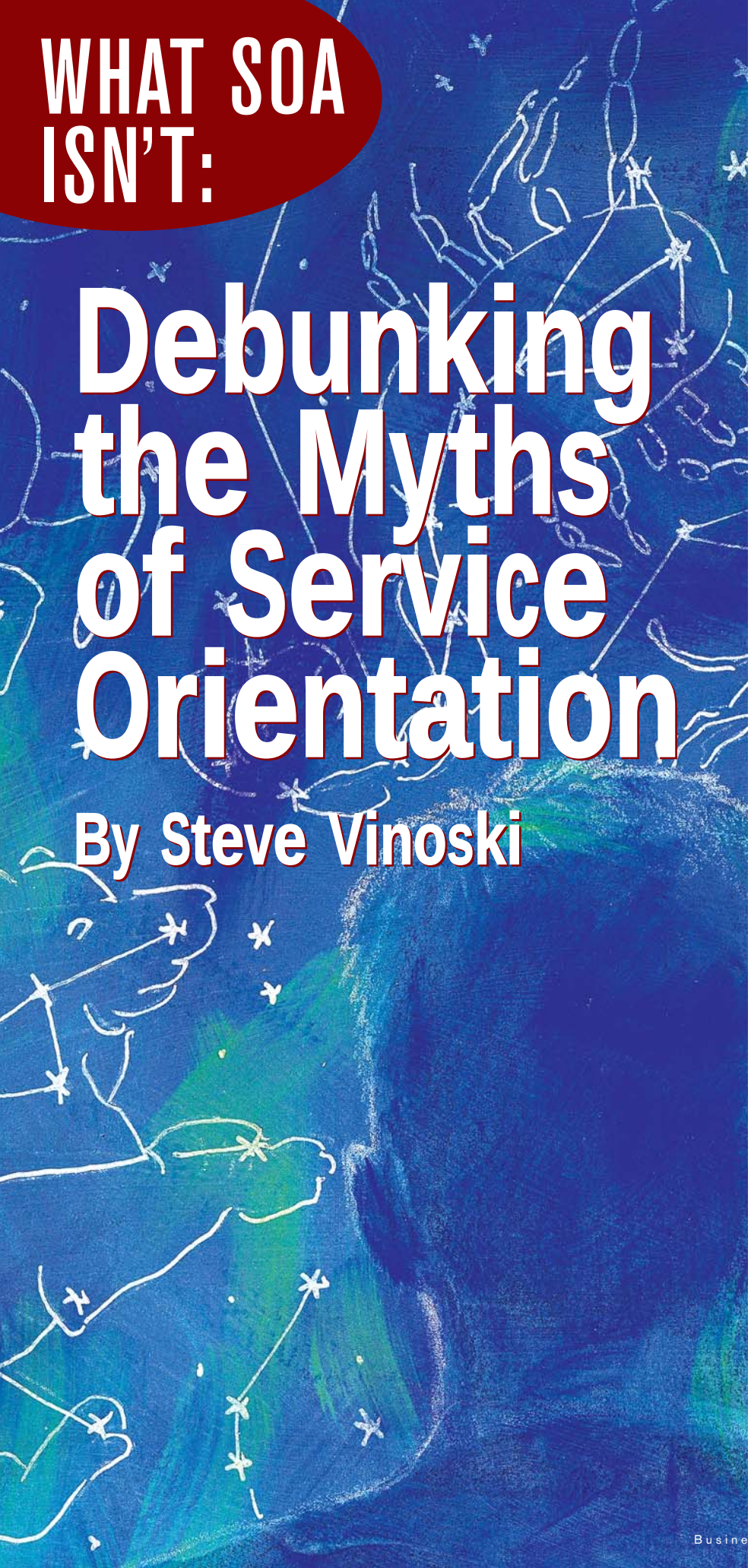
This article appeared in the
April 2005 issue of

business integration JOURNAL

Subscribe instantly at
www.bijonline.com

- Free in the U.S.
- $48 per year in Canada and Mexico
- $96 per year everywhere else



WHAT SOA ISN'T:

Debunking the Myths of Service Orientation

By Steve Vinoski

Every day, CIOs and enterprise architects face a variety of real-world IT challenges. The list is almost as long as it is familiar because these problems never seem to go away. The following questions a typical CIO might ask himself convey the essence of these never-ending challenges:

- How can I significantly reduce the cost of our enterprise IT infrastructure without compromising our competitive advantage?
- How can I ensure that our IT infrastructure becomes more responsive and business focused, while simultaneously becoming enduring and flexible?
- How can I strive for IT rationalization while simultaneously protecting previous investments and the strategic applications that run on them?
- Can I deploy a new architecture in a phased manner rather than a disruptive, risky "big bang" approach?
- Can I quantify the ROI for my initiatives and facilitate IT consolidation and innovation?

These aren't new questions and the usual answers are predictably lacking. Vendors are constantly touting the next great solution that will easily solve these problems. The latest silver bullet—Service-Oriented Architecture (SOA)—is generating considerable hype. Unfortunately, the messages coming from competing vendors are often contradictory and can be confusing. It's difficult to understand if and how SOA can benefit your business.

Migrating your enterprise to an SOA approach can be beneficial if you're realistic in your expectations of what SOA is and isn't. This article addresses some of the myths surrounding SOA by discussing a few popular, yet misunderstood trends that also share the acronym "SOA."

Is SOA New?

SOA doesn't stand for "State Of the Art." Rather, service orientation is firmly based on fundamental software engineering principles in use for decades. An indispensable element of service orientation is loose coupling between software services and applications. Good software developers have long understood the benefits of minimizing coupling between software modules while also maximizing the cohesiveness of each module. Loose coupling is even easier to achieve in today's distributed applications since the network provides a natu-

ral division between applications. When used properly, the distributed computing paradigm helps enforce these fundamental software engineering principles.

There are numerous examples of distributed computing technologies, including the Distributed Computing Environment (DCE), the Common Object Request Broker Architecture (CORBA), the Distributed Common Object Model (DCOM), and Enterprise JavaBeans (EJB), being used as the basis for successful service-oriented systems. However, this isn't to say that service orientation today is just a reinvention of these earlier technologies. In addition to being focused on distributed systems, these technologies were also partly focused on programming interfaces and infrastructure. This isn't the case for modern service orientation. Instead of requiring the same heavyweight distributed computing infrastructure under all applications that connect to each other to consume or supply services, modern service-oriented approaches focus solely on the messages that service consumers and providers exchange with each other.

Conjuring Up an Architecture

SOA doesn't stand for "Summon Our Architecture." If you don't understand the fundamentals that separate good software from bad, no amount of wishful thinking will conjure up a magical architecture that suddenly makes a real boy out of your enterprise Pinocchio.

Building good software is hard. No architectural approach is a substitute for understanding the fundamentals of software design and construction, or for understanding system requirements and being able to develop and deploy solutions that properly address those requirements. Too many developers and architects try to take shortcuts. The reduced quality typically resulting from those shortcuts is what gives the public such a poor impression of software. The enterprise computing systems these architects and developers create are composed of ad hoc point-to-point links between applications that were put there with little forethought. The result is typically a system that's hard to change and easily broken.

Realistically, it takes time to learn about and successfully apply the principles of service orientation to your enterprise. Be prepared to take several different views of your system:

- A "bottom-up" services view is important to help you determine which aspects of your infrastructure can most readily be service-enabled.
- A "top-down" view is necessary to help you determine which applications need to rely on which services, and to help you maintain the big picture view of what you're really trying to accomplish.
- A "message-focused" view of your system is necessary so you understand what messages your services accept and produce.

You need to:

- Determine what kind of services you need immediately to address specific problems.
- Understand the messages that services and their consuming applications will exchange.
- Know how services you create today will fit into the overall architecture of your enterprise.
- Invoke and rely on the principles and fundamentals of good software design and construction.
- Continue improving the quality of your enterprise services with each iteration.

Creating a flexible, extensible service-oriented system isn't easy. You'll go a long way toward ensuring the success of your IT improvement initiatives if you realize service orientation isn't magic and you take the time and effort to pursue and repeat these important steps.

Diversity and Heterogeneity

SOA doesn't stand for a "Same On All" approach or as many vendors have unfortunately suggested, "Scrap Older Applications."

Don't accept a vendor sales pitch that says you need to replace all your software with their software to get the benefits of SOA. That's completely false. Real-world computing systems are always diverse and heterogeneous. While many enterprises choose to embrace a common infrastructure for their projects going forward, they aren't prepared to invalidate the previous technology investments they've made by adopting a "rip-and-replace" strategy. Not only is this strategy unwise, it's nearly impossible to effectively achieve.

Few enterprises can move all their systems to new platforms in a "big bang" approach. They lack the time or budget to redevelop, retest, redeploy, and requalify all their critical applications. Moreover, the original developers of critical business services and the source code may not be available or easily found. For most large enterprises, a rip-and-replace strategy can't be implemented fast enough to keep up with advances in technology. By the time you finished ripping and replacing, the "new" technology would be old.

Rather than rip and replace, renovate your existing systems and applications. Doing this in a way that migrates everything toward service orientation requires a focus on extensibility and abstraction. Specifically, the resulting system must be based on abstractions that let service and consumer applications interact while their underlying implementations and communication details remain hidden from each other. These abstractions are critical to allowing your enterprise system to accommodate existing platforms and protocols while supporting the inclusion of new ones.

Enterprises also need to allow for extensibility in the area of Qualities of Service (QoS), specifically in areas such as security, management, fault tolerance, and transactions. Your existing systems will likely already be based on native capabilities in these areas, and

business integration journal takeaways

BUSINESS

- A successful Service-Oriented Architecture strategy can't be viewed as a silver bullet solution to the IT architecture challenges facing your business.
- Few enterprises are in a position to move all their systems to new platforms in a "Big Bang" approach, and they don't have the time or budget to redevelop, retest, redeploy, and requalify all their critical applications.

TECHNOLOGY

- Building good software is hard and no architectural approach is a substitute for understanding the fundamentals of software design and construction, or for understanding system requirements and being able to develop and deploy solutions that properly address those requirements.
- No single application protocol, not even SOAP, can solve all enterprise integration problems.

their native mechanisms must remain intact and be integrated with other such mechanisms wherever necessary. Like extensibility in the area of platform and protocols, extensible QoS helps to ensure the integrity of your established applications as you service-enable them and extend them into new areas.

A final example of extensibility relates to the concept of "middleware dark matter." Just as dark matter invisibly governs the behavior of the universe, enterprises are often powered by "invisible" home-grown, handcrafted solutions that are based not on recognizable technologies such as J2EE or .NET, but rather on simple yet effective approaches involving open source Web servers such as Apache and scripting languages such as Perl and Python. Such middleware dark matter systems support more strategic applications and business processes than many might realize, so your service orientation strategy must be sure to include these applications and processes. If you select a highly extensible enabling technology for your services, chances are good that you'll be able to appropriately service-enable the "dark matter" in your enterprise, enhancing its value.

Beyond the Application Server

Application servers have become a popular application development and deployment platform, but this doesn't mean that SOA stands for "Services On Appservers." Few organizations are prepared for the time, cost, and effort it takes to re-architect their existing applications to be deployed on an application server. This approach also has other potential problems.

When "application server fever" takes hold and you spend a lot of time and money moving applications to an application server environment, it becomes easy for your developers to assume that all service and consumer applications are based on that application server. As soon as they make that assumption, it becomes all too easy for them to allow implementation details to leak into the messages exchanged between consumers and services. When that happens, your infrastructure becomes brittle and inflexible, making it harder to incorporate the newer technologies that will arrive in the future.

Application servers can be good solutions for some problems, but for many integration projects, they're simply too heavyweight for what needs to be accomplished. Their footprints can be too large and they may cause unnecessary message conversions when, for example, XML messages are converted into and out of Java or C# objects as they flow through the system. You might also see that using an application server technology presents an impedance mismatch between the systems being integrated, such as with existing middleware approaches based on C or C++. Finally, they can also cause a poor service design approach by forcing you to think first about Java or C# objects and annotating them with special programming language keywords to turn them into services, rather than first designing the services and then treating C# and Java as merely an implementation detail. When you focus on the messages first, you can choose whatever implementation approach you deem suitable for that particular service rather than being forced to take a "one size fits all" Java or C# application server approach.

Coming Clean With SOAP

Finally, given the popularity of Web services, some might try to convince you that SOA stands for "SOAP-Only Applications." SOAP certainly has its place in a service-oriented enterprise. It has proved to be a good choice for coarse-grained integration of disparate systems. However, no single application protocol can solve all enterprise integration problems.

Strategic legacy applications are often the primary systems that need to be service-enabled in the enterprise. Often, when bringing these applications into the service-oriented enterprise, they cannot be modified in any way. The cost of doing so might simply be too high. In such cases, the only way to make them "speak" SOAP is to front them with SOAP gateways or bridges.

Gateways and bridges have at least two drawbacks. First, they add more moving parts to the overall system, thus making the system more difficult to debug and manage. Second, they're generally slow because they have to convert between the protocols and data formats they're bridging.

A better approach is to ensure your enterprise system can handle multiple protocols and data formats so consumer applications can speak whatever native protocol and format the service it is connecting to speaks. Writing applications that are aware they're based on SOAP locks that application out of being able to handle other protocols and can be a step in the wrong direction.

Should We Redefine SOA?

SOA stands for Service-Oriented Architecture. But is that even appropriate? After all, a software architecture is supposed to provide somewhat detailed and formal constraints that tell you how the various components of a system should be connected. What we call SOA isn't really an architecture but more of a set of guidelines and good practices. This might be why people have inflated expectations regarding what service orientation can do for them. They expect architectural guidance, but don't receive it. So even the term SOA is a misnomer.

Rather than "service-oriented architecture," consider using the term "service-oriented approach" or simply "service orientation."

Conclusion

Service orientation is an approach that focuses on the building blocks that regularly recur as solutions within a given application domain. With service orientation, you focus on the messages that those building blocks, or services, consume and emit, and you ensure all service and consumer application implementation details remain completely hidden. Service orientation also requires a focus on extensibility and abstraction given that communication protocols and data formats are a dime a dozen, subject to frequent change, and ultimately irrelevant details when it comes to solving your business problems.

Today's application servers and SOAP will be tomorrow's legacy. Have you designed your service-oriented enterprise to handle that when the time comes? It will be here sooner than you think. **bij**

Acknowledgements: *Thanks to my IONA colleagues Robert Morton and Jamie Osborne for their help with this article.*

About the Author

Steve Vinoski is chief engineer of Product Innovation for IONA Technologies in Waltham, MA. He is also an IONA Fellow. He joined IONA in December 1996 to start the company's U.S.-based engineering organization and to lead the development of IONA's next generation Adaptive Runtime Technology (ART), a highly flexible and high-performance distributed computing engine that underlies the company's products.
e-Mail: vinoski@iona.com
Website: www.iona.com/hyplan/vinoski