zapthink white paper

SOA Performance

Scaling SOA Implementations Now and into the Future

ZapThink, LLC • 108 Woodlawn Road • Baltimore, MD 21210 • info@zapthink.com • www.zapthink.com

SOA PERFORMANCE

SCALING SOA IMPLEMENTATIONS NOW AND INTO THE FUTURE

August 2008

Analyst: Jason Bloomberg

Abstract

Many organizations look to Service-Oriented Architecture (SOA) to provide greater business agility in the face of evolving business requirements and complex, heterogeneous information technology (IT) environments. To achieve this agility, architects implement a Services abstraction that loosely couples business Services from the underlying implementations of those Services. Building such an abstraction layer is not without risks, however—inherent in building such an abstraction is the risk of sacrificing performance and scalability to achieve the organization's required agility.

As SOA becomes mainstream, though, enterprises simply cannot afford to trade away performance to achieve agility. As a result, architects must plan for performance up front, as part of their SOA planning process, and leverage a variety of techniques and solutions to achieve performance and scalability as well as business agility as the traffic to their Services continues to increase.

I. Performance Challenges Facing Successful SOA Adoption

For many industries, IT has become an essential part of their business. Companies rely upon IT to provide better value for customers, greater power to employees, and improved agility to the organization as a whole. Companies whose IT environments lack flexibility or cost effectiveness risk losing their competitive edge in the marketplace to firms that can creatively leverage their technology assets for competitive advantage.

This increasing reliance on IT comes in the face of burgeoning complexity and business change. Companies that adapt to an ever-changing business environment are able to compete more effectively and thrive in any business climate. Such companies are particularly adept in tough economic times, often finding opportunities in the midst of chaos.

Instead of being an enabler, however, IT is frequently a significant barrier to companies' ability to respond to change. To address this need for agility in the organization, many firms are turning to *Service-Oriented Architecture* (SOA) – a set of best practices for organizing and managing IT resources and people to build and support Services that abstract the underlying complexity of the IT environment, providing greater power and flexibility to the business.

Today, the majority of enterprises and government organizations are somewhere on their SOA roadmap. In fact, in many ways, SOA is now mainstream—enterprises have now widely adopted core SOA best practices, and organizations are looking to scale their SOA implementations to meet growing and changing business needs. To this end, architects are looking to their operations teams to support their SOA initiatives so that they can offer high performance as the usage of Services increases.

Architects are looking to their operations teams to support their SOA initiatives so that they can offer high performance as the usage of Services increases.

Herein exists a critical problem with SOA: as an architectural approach, SOA favors agility over performance, and yet the organizations who adopt SOA require both. Furthermore, it typically falls on the operations team to support the performance requirements, even though the additional layers of abstraction that SOA introduces impede performance and scalability. And finally, in too many cases, architects and operations personnel don't communicate or work together well, yielding architectures that cannot scale, or implementations that become inflexible as performance requirements increase.

The Challenge of Heterogeneity

The first, most fundamental reason that achieving high performance SOA implementations is such a challenge for many organizations is the technical challenge of heterogeneity. Today's enterprise IT environments are enormously complex, and it is that complexity more than any other cause that leads to the inflexibility that the business wishes to address. And yet, SOA does not actually

eliminate complexity—it *abstracts* the complexity, providing a flexible, simplified set of Services to the business that overlays the unavoidable technical complexity.

At the core, IT has always dealt with underlying complexity through the power of abstraction. In the world of IT, abstraction is a way to simplify the complexities of the technology with simple, yet powerful representations. Beneath the abstraction layer exists the complexity that IT deals with today. But due to the power of loose coupling, the Services available to the business provide whatever value the business requires from them. It's vitally important to SOA that we place such Services into the business context.

As with any abstraction, however, there is no magic here. To build such powerful Services requires sophisticated governance, management, and an overall focus on quality and performance. After all, while it's simple to talk about loose coupling—where it's possible to independently control and change Service providers and consumers without impacting the other—such loose coupling depends upon a high quality, high performance SOA infrastructure.

The first step in getting a handle on the problem of heterogeneity is providing sufficient visibility.

The first step in getting a handle on this problem of heterogeneity is providing sufficient *visibility*. As the IT environment becomes more complex, no one has a single, end-to-end view of any transaction. Given the increased complexity in SOA, plus the fact that most of that complexity is beneath the Service abstraction, end-to-end visibility, deep in the IT infrastructure, is critical to understanding what is going on in the IT environment.

The Challenge of Abstraction

Many people still confuse SOA with Web Services, even though the two concepts are quite different. While SOA is a set of best practices for organizing IT resources to better meet changing business needs, Web Services are nothing more than standards-based interfaces. Furthermore, many implementations of Web Services do not constitute SOA, and many of the Services that form the core of most SOA initiatives aren't Web Services—instead, they are *abstracted Services*.

An abstracted Service is a representation of a business capability or data that the organization can compose with other such Services to implement business processes. An abstracted Service is typically a business Service, but not necessarily, as there is a role for abstracted IT Services as well. However, all business Services should be abstracted Services. Such business Services are the core abstraction that underlies SOA. Abstracted Services should always be loosely coupled, although the specific coupling requirements can vary. Building loosely coupled abstracted Services thus becomes the core technical challenge of implementing SOA. The core challenge with abstracted Services in turn is how we make an abstracted Service actually work, when the tools at our disposal are the Service implementations and interfaces and all the infrastructure that goes along with them. It's one thing to talk about "representations of business capabilities," and quite another to implement something in software that actually works.

The first critical point to understanding abstracted Services is to understand that there is typically a many-to-many relationship between Service implementations and Service contracts. A Service implementation—the software that underlies a Service—may support multiple contracts, each of which could correspond to a particular Service interface, for, say, a particular type of consumer. Similarly, there might be several implementations that support a single contract, and hence a single Service interface, for the purposes of scalability or fault tolerance, for instance.

With abstracted Services, however, the relationship becomes what we might call "many-to-many-to-many": a particular abstracted Service might have several contracts that represent relationships with various consumers, while also representing multiple Service interfaces that themselves might each have one or more Service implementations. This approach might sound overly complex, but it's the key to loosely coupling the abstracted Service.

The abstraction layer intentionally hides the underlying implementation details, but those details are what makes the Services run.

Clearly, managing abstracted Services is a tall order. The abstraction layer intentionally hides the underlying implementation details, but those details are what makes the Services run. Furthermore, because of the many-to-many-to-many principle, a single abstracted Service may have several implementations running on different infrastructure or even in different geographic locations—and which implementation supports a particular Service might be different between one invocation of that Service and the next.

In some ways, in fact, abstracted Services suffer the same performance issues that transactional Web sites faced in the late 1990s. The user might see a single Web page, but many different Web servers might be responsible for serving that page to different users in different geographies, and those Web servers may in turn serve as the front end for a cluster of application servers and databases that may shift in configuration over time. Only now, instead of supporting an abstracted user interface, the infrastructure must support an arbitrary abstracted, contracted interface.

The Challenge of Performance

The challenge of Service performance begins with the fact that Service performance is a vague term. It can mean many different things to many different people. To operations and networking individuals, performance is a matter of guaranteeing up-time, managing utilization of resources, and keeping Services secure and running without glitches. To business analysts and project managers, Service performance is a matter of making sure that the processes are performing according to the specifications. To developers, Service performance is a matter of ensuring that the functional requirements are being met regardless of the usage of the Services. To the business, Service performance is a matter of meeting key performance and agility indicators. And so, to properly define Service performance, we have to look at the concept from all these perspectives.

The first thing that comes to mind when most think about Service performance is operational performance. Operational performance is how a Service, any Service, behaves in the IT environment. Operational behavior includes the uptime and availability of the Service as well as resource utilization and distribution of load across multiple Service implementations. In this regard, we can use traditional systems management approaches to monitor, measure, and manage Service operational performance.

However, SOA introduces a big twist in operational performance. Not only does the availability of a Service to respond to requests impact Service uptime, but the metadata that control Service behavior also play an important part. In addition, there might be multiple, distributed implementations of a Service throughout the network, and so managing Service operational performance becomes more like managing a Service "grid" than individual, discrete Services.

High operation-performance Services should ideally introduce no additional latency, usage load, or security gaps. However, thinking about consuming Services in a continually changing environment requires network administrators and other operations personnel to rethink the concepts of Service Level Agreement and Quality of Service as well. In this case, the word "service" in SLA

and QoS truly means a Service in the SOA context. So, rather than Service Level Agreement broadly applying to any kind of services, we're looking at Service Level Agreement as applied to Services in the SOA context.

Secondly, functional performance is a critical part of the Service performance story as well. Knowing whether or not a Service is responding to requests or overloading infrastructure resources is not enough to know if a Service is performing. From a developer's and implementation-focused architect's perspective, a Service is performing if it provides the results expected by the consumer.

But even testing and validating the Services against continually changing business requirements and the metadata-controlled environment is not enough to guarantee the functional performance of a Service. High functional-performance services meet the agility requirement, meaning that we can find new uses for existing Services without requiring new development. And remember, metadata define a Service more so than does its implementation. As a result, organizations should measure the functional performance of its metadata more so than the implementations themselves to guarantee continuously high performing Services from a functional perspective.

Operational and functional performance, however, don't tell the whole Service performance story, because the power of SOA comes not from discrete, atomic Services, but rather from the composition of those Services to satisfy the requirements of business processes, which we call Service-Oriented Business Applications, or SOBAs. In this light, operational and functional performance matters only if the business process as a whole is performing for the enterprise.

A Service composition would be high performance not only if it operationally and functionally meets the process requirements, but also if the process itself can exhibit high performance characteristics. This principle means that architects and business analysts should be able to optimize SOBAs and choose among different process definitions to find the one that most optimally meets the business requirements. A high process-performing SOA has SOBAs that architects can easily reconfigure and recompose as well as measure through process-specific performance indicators.

A high process-performing SOA has Service-Oriented Business Applications that architects can easily reconfigure and recompose as well as measure through process-specific performance indicators.

Clearly, what makes one SOBA higher performance than another is not simply that it consists of Services, but rather that architects define and manage it in a way that allows the process to evolve along with continually changing business requirements. As a result, architects should create "Process-Level Agreements" that reinforce, but are separate from the SLAs defined in the operational and functional contexts above. An example of a Process-Level Agreement might be that users must be able dynamically reconfigure the process in the case of some exception within a certain amount of time. Other Process-Level Agreements can define user involvement in parts of the process or behavior of long-running, asynchronous processes.

When to Consider SOA Performance

There's clearly no way to layer on these various aspects of Service performance after the fact. Architects should consider SOA performance issues at a very early stage and communicate frequently with the operations teams about their SOA performance requirements. The operations team should have plenty of lead time to put SOA performance management processes and tools in place before the SOA environment is put into production.

It's also important to remember that many SOA platform vendors have not focused on scalability as a core feature of their solutions, opting instead to offer

feature/function enhancements. Eventually the number of Services and the traffic flowing to and from those Services will consume the available resources of the underlying middleware, servers, and data stores. Too many fine-grained Services exacerbate this problem, as they require more network traffic than course-grained Services.

Furthermore, SOA implementations should include distributed Service implementations. Thus, the more processing you can place at the Service endpoint, the better the implementation will perform. Many Services expose traditional legacy APIs, and therefore the adaptation layer between those APIs and the Services can easily be a performance bottleneck. It's also important to consider performance when selecting the technology for the composition layer. Many BPEL engines and other process tooling are notoriously poor performers, and can also become a bottleneck.

Finally, performance modeling and performance testing are essential parts of the SOA planning process. Modeling enables the team to predict behavior of Services under load, and performance testing offers the final validation. It's important that performance test plans include real-life scenarios, simulating the production environment as much as possible.

II. Scaling SOA Performance

Dr. Peter F. Drucker, the often quoted "father of modern management" once said, "you can't manage what you can't measure." This statement is just as true with SOA performance as it is with every other aspect of the business. What makes SOA performance unique is that performance brings together aspects of IT that IT personnel have previously considered as separate concepts. One performance criterion should not have any impact on any other, while at the same time it's important to be able to combine different aspects of Service performance together to achieve the overall behavior of the entire system.

In this manner, enterprise architects must consider Service performance from all the viewpoints in this paper. High performance SOA implementations are low latency, high availability, high quality, functionally relevant, configurable, business relevant, and profitable—and visibility is the key to performance. If you can bring all the aspects of Service performance together and make it all work, you'll make your SOA valuable and successful.

Defining Performance in the SOA Context

To achieve the business value that organizations require from their SOA initiative, they must be able to achieve their flexibility and reuse goals. System flexibility leads to business agility, which can determine the difference between success and failure of the SOA effort. Meanwhile, Service reuse can increase productivity and reduce maintenance over time.

Many architects have designed their SOA implementations in a way that can impede performance.

While SOA promises these benefits, many architects have designed their SOA implementations in a way that can impede performance. Without the proper planning, flexibility comes at the expense of performance, leading architects to choose one over the other. But with proper planning, it is possible to achieve the required flexibility without sacrificing scalability and performance.

This SOA performance problem falls into two broad areas: ensuring sufficient performance of Service interfaces as well as of abstracted business Services. Service interfaces abstract existing systems, so ensuring their performance necessitates managing the performance of the components, applications, and systems that lie beneath the Services abstraction. As you might expect, dealing

with the performance of Service interfaces leverages well-established capacity planning and performance quality assurance techniques, including clustering, virtualization, and load testing. Today's architects are adept at making infrastructural decisions that ensure, for example, sufficient database performance, distribution of traffic onto a cluster of application servers, and the like.

While scaling Service interfaces is essential, the Services abstraction is at the heart of SOA. This abstraction which masks the complexity of the underlying technology implementation while presenting composable business Services to internal and external users. But every abstraction comes at a price, and the Services abstraction is no exception. Loose coupling, composability, agility, and the other benefits of SOA all introduce performance overhead. For limited sets of Services with small numbers of users, this performance hit may be minimal. For SOA implementations with large numbers of users, Services, or traffic, however, maintaining the necessary performance levels presents a substantial challenge, both to the architects who design the infrastructure as well as IT operations personnel who are responsible for keeping the lights on.

Performance Beneath the Services Abstraction

Runtime performance has always been a primary concern for the professional developer. Throughout IT's history there have been numerous techniques, approaches, and even architectures offered to specifically address performance and scalability. While the promise of SOA surpasses that of previous IT architectural models in terms of flexibility, adaptability, and strategic gain, it imposes performance demands. Therefore, with SOBA development now on the forefront, performance optimization is receiving more attention than ever.

Some of the reasons for this particular attention to performance include the increased emphasis on reuse and composability, leading to often unpredictable usage demands on Services. Furthermore, loosely coupled Services require a messaging-centric communications framework, which means that applications must now handle not only traditional processing logic, but also message transmission, validation, and parsing.

As organizations build out their SOA implementations, maintaining the Services abstraction in the face of high XML traffic levels becomes a key focus.

Furthermore, today's SOA implementations almost universally rely upon XML. As the use of Services grows, therefore, so too will the quantity of XML on the network, thus jeopardizing the behavior of the Services themselves. As organizations build out their SOA implementations, therefore, maintaining the Services abstraction in the face of high XML traffic levels becomes a key focus. Fail to maintain the abstraction, and the Services no longer meet the agile needs of the business, and the quality of the SOA implementation comes crashing down like a house of cards.

There are two primary approaches to achieving the performance that organizations require to maintain the quality of their SOA implementation in the face of such increases in traffic. By *scaling up* the infrastructure, it's possible to achieve high throughput—the number of messages of a given size that the system can process within a given time period, while reducing latency—how long a message takes to travel through the system. In addition, *scaling out* the system means leveraging parallel computing approaches to divide workloads across multiple intermediaries or other elements of the infrastructure at the same time.

There are other technical approaches for avoiding unnecessary bottlenecks that can impede performance. For example, while loosely coupled interfaces are generally desirable, there are places where tightly coupled connections are sufficient, for example, for legacy integrations that are unlikely to change over time. It's also a good idea to leverage hardware acceleration for processor-

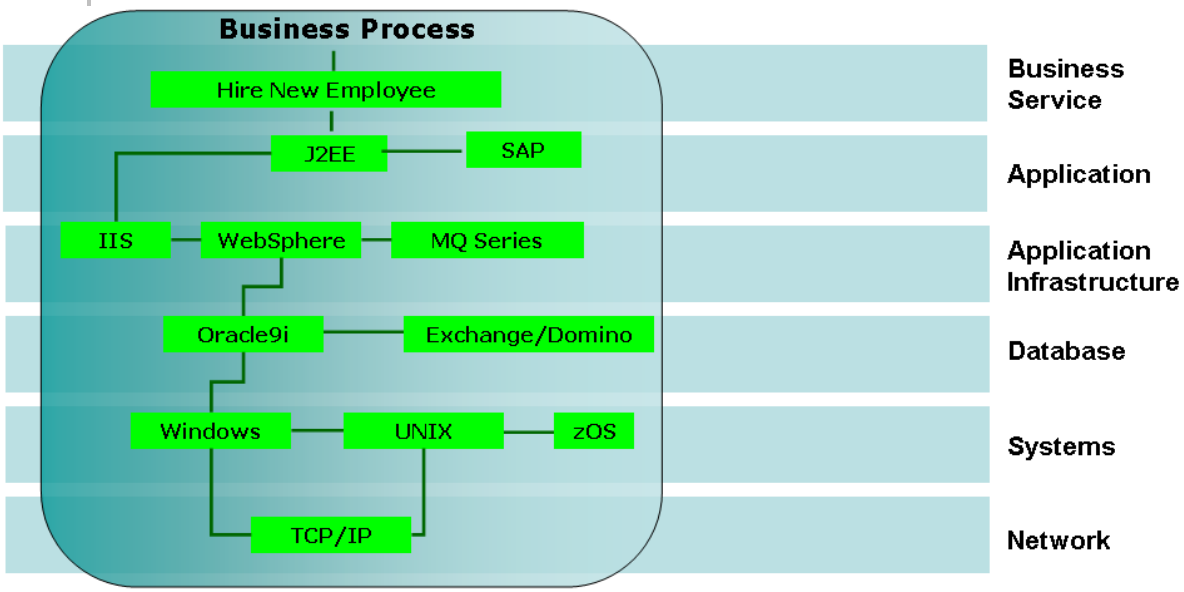
intensive XML operations like transformations and security operations that involve encryption or decryption.

While SOA performance management often concentrates on the Services layer, it's important not to manage the Services layer as though it were a technology silo, but rather in conjunction with the application infrastructure and back end systems below it.

Furthermore, while these technical approaches offer point solutions that can resolve specific bottlenecks, runtime visibility is critically important across all parts of the infrastructure to ensure scalability of the SOA implementation. Such IT visibility enables production-ready management of the infrastructure with cross-platform support, as well as end-to-end SOA transaction visibility.

Finally, while SOA performance management often concentrates on the Services layer, it's important not to manage the Services layer as though it were a technology silo, but rather in conjunction with the application infrastructure and back end systems below it, as shown in the figure below. The vertical nature of a transaction's path through the many different layers of infrastructure is the real challenge.

Service Performance Depends upon Layers of Infrastructure



Source: CA

Performance Above the Services Abstraction

While SOA manifestly relies upon Services, there is far more to properly architecting SOA than simply building a bunch of Services. Architects must consider the consumption of those Services as well, including the dynamic, business-driven composition of Services into SOBAs. Unfortunately, the very nature of SOBAs as flexible, continually changing, potentially *ad hoc* compositions presents complex performance challenges to architects and operations personnel alike.

In fact, there are several dimensions of SOBA performance that architects must consider as they plan their SOA:

- *Balance between use and reuse* – Some Services expect high use, meaning large volumes of traffic from specific consuming applications, while others expect high reuse, implying a range of different consumers that may use Services in different ways, for example, in different SOBAs. High usage is often (but not always) more predictable than high levels of

reuse, but the architect must plan for both, as well as the combination of the two.

- *Very large messages and granularity* – Addressing the performance issue of very large messages requires different infrastructure and different planning from Service interactions that exchange large volumes of messages. Sometimes this problem directly relates to the granularity of the Services, while other times it might concern SOAP attachments, encryption, or other features that increase the size of messages. But in any case, the architect must take the sizes of messages into account as part of the SOA performance plan.
- *Dynamic performance policies* – In some cases, the service level requirements of individual Services is part of the contract for each Service, but in other cases, the organization requires performance policies that they can apply to Services as part of their governance framework. In fact, being able to reconfigure performance policies may be a business requirement the architect must take into account when planning the SOA.
- *Service dependencies* – Service compositions come in many flavors: orchestrated flows, flexible choreographies, data virtualizations, and various combinations thereof. String several Services together where the output of one contributes to the input of the next, for example, and if one Service in the chain is too slow, the entire composition suffers. Multiply this bottleneck issue by the numerous ways that people can create SOBAs, and the architect has a complex task ahead.
- *Anticipate unplanned usage* – In spite of run time governance in place, inevitably in the production environment you are going to get developers who will either intentionally or unintentionally end up misusing Services in unanticipated ways that the original design may not have accounted for.

Runtime SOA governance focusing on policy enforcement looks good at the architecture stage, but in a production setting can create additional overhead and complexity.

It's also important to remember that runtime SOA governance focusing on policy enforcement looks good at the architecture stage, but in a production setting can create additional overhead and complexity. In addition, the SOA runtime governance platform itself creates another potential point of failure. Sometimes this type of functionality belongs in the underlying SOA infrastructure rather than in the management system, and in other cases, it's essential that the management take place via lightweight agents that do not impede performance.

Tackling the SOA Performance Problem

Dealing with performance bottlenecks is like playing whack-a-mole: defeat one and another immediately pops up. Even worse, implementing SOA just increases the number of moles you have to whack. It's essential, therefore, for the architect to plan for performance bottlenecks at different levels, both above and beneath the Services abstraction. In other words, the architect must craft a performance plan that should take advantage of some combination of the following approaches:

- *Service and infrastructure virtualization* – Various virtualization techniques can provide cost-effective approaches to dealing with variable performance issues, essentially by abstracting a specific part of the infrastructure. Virtualization is especially useful for dealing with unexpected spikes in demand, but the complexity of virtualizing heterogeneous resources can often limit such approaches' effectiveness.

- *Combining judicious loose coupling with strategic tight coupling* – While a simplistic view of SOA might suggest that loose coupling is always better than tight coupling, the fact of the matter is that loose coupling introduces overhead, while tight coupling can smooth out bottlenecks. The architect's challenge, therefore, is in identifying those situations where the business requires some level of loose coupling, and then providing only as much as it needs, for example, by implementing transactionality in compiled code and distributing that code for high concurrency parallel processing underneath the Services abstraction.
- *Content-based routing* – A Service-oriented approach to load balancing leverages registry-based location independence and intermediary-based routing to route Service requests to the appropriate Service instances in order to satisfy performance requirements. This dynamic routing approach to scalability and fault tolerance is unlikely to be as performant as tightly coupled clustering, but works well in heterogeneous environments, and can leverage the declarative policy management capabilities of the registry.
- *Performance as part of the SOA governance framework* – The broadest, most agile approach to SOA performance is to plan for it as part of the governance framework for the SOA implementation. Based on the performance constraints of the technology, the architect should craft policies that will maintain the required performance levels while empowering users as much as is practical. For example, if the architecture team can distinguish particular types of SOBAs that the infrastructure can support from others that it cannot, then the team should be able to craft policies that will appropriately limit SOBA creation, and thus create predictable limits for overall performance.

While including performance policies in the SOA governance framework is an important best practice, it's also important to remember that SOA runtime governance is a very distinct product category from SOA performance management. Customers should consider a SOA management solution that can manage every layer of an SOA implementation and is focused on production readiness. Runtime SOA governance tools, in contrast, often focus on the runtime behavior of the Service interfaces, while not addressing the underlying performance challenges that can limit the scalability of the SOA implementation.

III. SOA Management and Continuous Quality

The essence of SOA, of course, is building for change. It is not sufficient to simply scale a particular implementation of SOA, but instead, organizations must realize that they must maintain performance even as changing business requirements lead to changes in the SOA implementation—and ensuring that systems meet business requirements is fundamentally the definition of quality.

Quality means far more than simply reducing defects. Fundamentally, quality means building something that meets the requirements of its users, now and into the future. Being defect-free is a necessary, but by no means sufficient criterion for a quality product. Software quality is no different. While many software quality assurance efforts focus on eliminating bugs, the bug-hunting process is only the starting point for software quality.

The real challenge with software quality, as with any other quality effort, is in guaranteeing that the software meets the requirements set out for it, because only by providing this guarantee can IT establish trust. In an ideal world, quality

assurance (QA) personnel would simply take the requirements document, use it to build a test plan, and run tests against that plan. Once the project passes all the tests, it's ready to go live. But in the real world, requirements continue to evolve, both during projects as well as once the projects are complete. And there's nothing worse than evolving requirements for throwing a wrench in the most carefully laid QA plans.

SOA, naturally, facilitates environments of continually changing business requirements. SOA leverages a metadata-driven Service abstraction to provide greater power and flexibility to business users, with the clear purpose of enabling IT to respond to changing requirements in an agile manner. This core agility benefit of SOA collapses, however, if the Services or the applications that consume and compose them are of poor quality, behave in an unpredictable manner, or fail to scale.

The core agility benefit of SOA collapses if the Services or the applications that consume and compose them are of poor quality, behave in an unpredictable manner, or fail to scale.

The SOA story touches upon many challenges, including loose coupling, trust, governance, and management, to name a few. But in the final analysis, SOA quality becomes the primary organizing principle for actually getting this stuff to work—what does it actually take to enable IT to meet the changing needs of the business. SOA quality becomes the thread that enables the business to trust technology to provide the agility it requires to meet requirements now and into the future.

Full Lifecycle SOA Quality: Supporting the Requirement for Agility

Perhaps the greatest SOA quality challenge, therefore, involves maintaining quality throughout the Service lifecycle, especially once the SOA implementation is in place. The problem is, the more mature the SOA implementation is—that is, the better the Service abstraction maintains an agile separation between business users and the underlying IT capabilities—the more impractical traditional QA approaches are likely to be. In many of today's IT shops, there are separate, identical QA and production environments. QA personnel can load any new or changed code into the QA environment, and test it to their heart's content before giving it the thumbs up for promotion of changes to the production environment.

In a mature SOA environment, it's practically impossible to maintain a useful duplicate of the running system, because Services, configurations, and associated metadata continually change. As a result, maintaining a parallel QA environment rapidly becomes an exercise in futility. The solution to this quality conundrum is to test new and changed Services and Service configurations in the live, production environment. The only way to ensure that all aspects of the new configuration continues to meet the requirements set out for it is to run test messages through production Services. Now, saying you should test in production is tantamount to proposing rewiring your house with the power on—it's possible, but you have to be especially careful, know what you're doing, and plan ahead. In the case of SOA, planning ahead means that Services (as well as Service consumers) must be able to support a testing mode.

Furthermore, in a complex SOA environment it's impossible to test every possible Service combination at every usage level. Test thoroughly before deployment, but also acknowledge the fact that SOA implementation by their nature will be dynamic and unpredictable. Therefore, real-time visibility in the production environment is critical for ensuring quality.

Traditional performance testing typically applies simulated loads before deployment to a test environment that closely mimics the production environment. After all, such traffic loads can bring systems to their knees, so nobody would want to run them in a live environment. Performance testing for

SOA must therefore take a more subtle approach than the "pound on it until it fails" technique. Instead, SOA performance testing relies upon the use of configurable policies that indicate whether a Service is in live or test mode, combined with SOA management that proactively monitors live performance and takes preventative steps should Services cross pre-set warning thresholds.

There are a few important notes about running the QA process through a production system. First, the QA process is policy-driven. As a result, the testing process is itself Service-oriented, which goes a long way to satisfying the agility requirement of SOA. Second, it's resilient. Even if a fully tested Service still fails in production, the QA process responds in a way that minimizes the business impact, and thus maximizes the loose coupling of the Services. But most importantly, SOA quality requires planning ahead. Architects must plan for test modes and deprecation policy support as an essential part of designing Services, and furthermore, plan ahead for performance testing in the context of an agile environment.

IV. The ZapThink Take

Analyzing SOA performance highlights the fact that SOA is more evolutionary than revolutionary. Architects must still know how to use every capacity planning and performance enhancement tool in their toolbelt, only now they're able to add a few new tools to the mix. In fact, there's no way we'd be able to figure out how to scale Services if we hadn't already worked out how to scale traditional Web applications.

It's also important to note that SOA performance is about more than ensuring that Services perform as required, just as SOA is about more than building Services. SOA best practices also cover the consumption of Services within SOBAs as well as at the user interface. Furthermore, dealing with SOA performance requires an Enterprise Architecture approach to SOA. Those bottleneck moles in the whack-a-mole game can appear anywhere in the enterprise, at any level of abstraction. The fact that SOA hides the complexity of the infrastructure from the user only exacerbates the need for an enterprise perspective, because high quality, high performance SOA requires high performance from every part of the enterprise.

The fact that SOA hides the complexity of the infrastructure from the user only exacerbates the need for an enterprise perspective, because high quality, high performance SOA requires high performance from every part of the enterprise.

It's also critically important for IT operations teams responsible for SOA performance to know what steps they should take, especially when they have no access to architects or if it's too late to re-architect the system. Operations personnel should be ready to consider approaches like virtualization of Services and XML acceleration, as such approaches don't necessarily require an architecture team to accomplish. The bottom line: organizations require a passive SOA performance management solution that provides visibility into the Services layer as well as all the complex infrastructure layers beneath.

Finally, adequate consideration of SOA performance helps address one of today's must fundamental fears about SOA: will the implementation work when the deployment team finally throws the switch? SOA is architecture after all, and architecture without implementation is nothing but wishful thinking. Without working software, SOA efforts are for naught. Get SOA performance right, and the bottom line is that the SOA implementation will work as advertised.

Copyright, Trademark Notice, and Statement of Opinion

About ZapThink, LLC

ZapThink is an Enterprise Architecture (EA) strategy advisory firm. As a recognized authority and master of Service-Oriented Architecture (SOA) and EA, ZapThink provides its audience of IT practitioners, consultants, and technology vendors with practical advice, guidance, education, and mentorship solutions that assist companies in leveraging SOA to meet their business needs and presenting viable SOA solutions to the market. We provide this audience a clear roadmap for standards-based, loosely coupled distributed computing – a vision of IT meeting the needs of the agile business.

ZapThink provides IT practitioners strategic insight and practical guidance for addressing critical agility and change management issues leveraging the latest EA and SOA best practices. ZapThink helps these customers put EA and SOA into practice in a rational, well-paced, and best practices-driven manner and helps to validate or recover architecture initiatives that may be heading down an unknown or incorrect path. ZapThink assists with solution vendor, technology, and consultant selection based on in-depth, objective evaluation of the capabilities, strengths, and applicability of the solutions to meet customer needs as they relate to EA initiatives and as they map against emerging best practices. ZapThink enhances its customer's skills by providing education, credentialing, and training to EAs to develop their skills as architects.

ZapThink helps to augment consulting firms' EA offerings and intellectual property by providing guidance on emerging best practices and access to information that supports those practices. ZapThink provides frameworks for product-based consulting based on ZapThink insight and research, such as SOA Implementation Roadmap guidance, Governance Framework development, and SOA Assessments, and provides a means to endorse and validate consulting firm offerings. ZapThink also accelerates consulting firms' efforts to attract, retain, and enhance the skills of EA and SOA talent by providing education and skills development

For solutions vendors, ZapThink provides retained advisory for guidance on product strategy, as well as marketing, visibility, and third-party endorsement benefits through its marketing activities, lead generation activities, and subscription services. ZapThink enables vendors to leverage ZapThink knowledge to transform their offerings in a cost-effective manner.

ZapThink's Managing Partners are widely regarded as the "go to advisors" and leading experts on SOA, EA, and Enterprise Web 2.0 by vendors, end-users, and the press. Respected for their candid, insightful opinions, they are in great demand as speakers, and have presented at conferences and industry events around the world. They are among the most quoted experts in the IT industry.

ZapThink was founded in October 2000 and is headquartered in Baltimore, Maryland. Its customers include Global 1000 firms and government organizations, as well as many emerging businesses. Its Managing Partners have worked at such firms as IDC, Saga Software, Mercator Software, marchFIRST, and ChannelWave, and have sat on the working group committees for standards bodies such as RosettaNet, UDDI, and ebXML.

Call, email, or visit the ZapThink Web site to learn more about how ZapThink can help you to better understand how SOA will impact your business or organization.

ZAPTHINK CONTACT:
ZapThink, LLC
108 Woodlawn Road
Baltimore, MD 21210
Phone: +1 (781) 207 0203
Fax: +1 (815) 301 3171
info@zapthink.com

ZapThink, LLC • 108 Woodlawn Road • Baltimore, MD 21210 • info@zapthink.com • www.zapthink.com