

# The SOA Magazine

## Feature Article



### Watch Your SOA Blind Spots: A Checklist for Testing Web Services

by Mamoon Yunus and Rizwan Mallal

Published: May 27, 2007 (SOA Magazine Issue VIII: June 2007, Copyright © 2007)

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

*Abstract: Web services testing techniques have been around for some time. However, with the increased utilization of Web services within service-oriented solutions, the demands and complexities placed on Web services are being taken to a new level. This is in sharp contrast to traditional RPC applications and integration architectures wherein the role of Web services was typically limited to point-to-point data exchanges. Now, Web services find themselves being reused across multiple service compositions and in the midst of dynamic and sophisticated runtime service activities and chains. This article raises a series of testing issues and provides recommended techniques and remedies for establishing robust Web services-based SOA implementations.*

### Introduction

Web services – the most foundational technology within modern service-oriented architecture implementations – are self-contained, modular applications that one can describe, publish, locate, and invoke over a network. Web services are agnostic to operating systems, hardware platforms, communication protocols, and programming languages, and have blurred the boundaries between network devices, security products, applications, and other IT assets within the enterprise.

Almost every product or technology is now able to advertise its interface as a Web Services Description Language (WSDL) definition ready for XML and SOAP-based messaging. Using SOAP for system-to-system messaging and WSDL for interface description, IT professionals have gained unprecedented flexibility in integrating IT assets across corporate domains. It is this flexibility that can make developing and deploying robust, resilient, and reliable service-oriented solutions so challenging. SOA project teams are now responsible for adapting traditional testing techniques, selecting appropriate testing tools, and developing Web services domain expertise to make their SOA deployments deliver business value reliably and securely.

In this article, we describe common SOA "blind spots" that have emerged from recent experience with real-life deployments. Specifically, we will focus on testing-related issues in the following areas:

- Performance
- Security
- SOAP Attachments
- WSDL Definitions (and XML Schema Definitions)
- Interoperability

### Performance Blind Spots

## 1. Web Service are Independent

Modern service-oriented solutions are based on the constant utilization of reusable services. Service reusability encourages a service to become dependent on other services. This can lead to situations where SOA testers evaluate performance characteristics of a service while being unaware of "hidden" services that it may be depending on. Without knowledge of internal or external dependencies, testers can easily assume that the performance bottleneck lies within just the service they are testing, while the actual bottleneck may exist elsewhere.

*Remedy:* Service-oriented and Web services-based solutions do not operate in silos but are connected with other services. To trace the root cause of a bottleneck, the following techniques can be applied:

- Isolate the problem by identifying and directly testing each Web service within a chain or service activity by using specific test cases. This means consuming the WSDL definition of each Web service along that chain.
- Disable downstream Web services and substitute the disabled Web services with similar "test services" that have known performance characteristics. Using simulated downstream services, the behavior of the "line-of-sight" Web services can be determined more accurately.
- Monitor CPU and memory statistics of each service along a chain as the test transaction flows through this chain. This will give the tester a broad sense as to which service may be the cause of the delay.

*Challenges:* To isolate a Web service's bottleneck, access to downstream WSDL definitions are necessary for simulation purposes. Corporate, technical, legal, and governance obstacles may prevent access to these WSDL definitions and Service Level Agreements (SLAs) with downstream third party service providers may not include provisions for testing without incurring transaction costs.

## 2. Web Service and Web Site Performance Measurements are Essentially the Same

Web site testers focus on a number of performance characteristics including response times, transactions, fail/pass ratios, and throughput. For scalability testing, Web site testers typically ramp up the number of concurrent clients coming into a Web site and then study the resulting statistics. Testers keep an eye on the response times and determine whether the latency is acceptable. Independent of the latency, they look at the transaction fail/pass ratios that have acceptable percentage thresholds of the "503 - Service Unavailable" error.

In the Web services-based SOA world where interdependent services are the norm, fail/pass ratios should be measured beyond HTTP codes. With Web services, failures are returned within the message as SOAP faults. These faults can originate from downstream services and can propagate all the way to the calling client. Understanding when a Web service has hit its "performance knee" requires deeper content awareness than just examining HTTP return codes for success or failure conditions.

*Remedy:* Error states need to be considered more comprehensively beyond just HTTP return codes. To identify fail/pass ratios, the following options should be considered:

- Use HTTP success and failure codes and SOAP faults appropriately. In certain deployments consumers may require only "200 - OK" HTTP responses even in the presence of a SOAP level failure. This is a non-standard approach. Also, be sure to adhere to specifications.
- Use performance tools that are Web services aware so that scalability testing is not restricted to HTTP codes only. SOAP faults should trigger failure counts during performance tests.
- Beyond HTTP codes and SOAP faults, business (semantic) data within a SOAP message may also indicate failure conditions. By using XPath statements, any data element in a SOAP body can be reviewed for business level errors. For example, during scalability testing, if a stock quote response returns a negative value, then an error has occurred. In this scenario, the HTTP codes and even the SOAP faults indicate a lack of errors.

*Challenges:* SOA implementers have to make sense of SOAP faults generated by a Web service and its downstream components. Without becoming an integral part of a Web service design and development process, SOA implementers can create unwanted blind spots that prevent them from seeing and identifying potential failure states. This inability to see Web services failure conditions results in meaningless performance characteristics and statistics reduced to simple Web site performance profiles.

Propagating excessive information in a SOAP fault, such as stack traces, is beneficial while diagnosing a system prior

to deployment; however detailed SOAP faults are a risk and can unnecessarily expose corporate infrastructure internals. Such an information leak should be avoided and only recoverable SOAP faults should be presented to consumers in runtime deployments. Finally, the business context of a SOAP response needs to be understood to successfully identify an error in the SOAP message that lacks HTTP errors or SOAP faults.

## Security Blind Spots

### 1. Static SOAP Messages are Sufficient

A key part of SOA security testing involves WS-Security validation. A tester has to ensure that target Web services that require message-level security (such as WS-Signatures, WS-Encryption or SAML) can consume such messages without any errors. A common blind spot in such tests occurs when the tester uses SOAP messages with static timestamps, nonces, and security headers. The specifications around timestamps and security elements require unique wire signatures for all messages. The static messages should be detected by the consuming servers as an expired message, or a replay attack.

*Remedy:* Regression suites should be developed that:

- Generate dynamic time stamps, nonces, and security tokens, as required by the WS-Security 1.1 specifications.
- Test whether the target Web services honor expired timestamps by sending stale messages and replay attack detection by sending duplicate nonces.
- Perform load testing where each SOAP request is unique. Taking a SOAP message with static timestamps, nonces, or security tokens should result in SOAP faults during a load test.

*Challenges:* A common practice is to morph existing Web application testing tools into Web services testing tools. Static WS-Security message are generated using a "cut-and-paste" scheme where the message is moved into the load testing tool environment. Understanding the nuances of WS-Security can be overwhelming, but developing this skill-set (and utilizing the right tools) is essential for building secure and scalable performance testing suites suitable for service-oriented solutions.

### 2. Positive Tests Cases are Sufficient

Within a typical testing process, most of the focus is generally on "positive" test cases. For the more determined testers who want to probe boundary conditions and perform negative tests, a common approach is to build some hand picked negative tests manually and make them part of the regression suite. Hand building negative test cases that test the boundary conditions of Web services is inefficient and usually ineffective and typically based on hit-and-miss heuristics. Manually generated negative tests may help identify a few quality issues but lack comprehensive coverage and can still leave behind several blind spots within the SOA deployment.

*Remedy:* Automate negative test case generation via automation tools. Such tools should provide sophisticated intelligence for test cases based on:

- The specifics of the target Web service interface (negative test cases can be derived from the WSDL definition).
- The ability to define success and failure criteria based on the SOAP responses. For example, if a Web service invoked with a negative test case does not return an HTTP error or a SOAP fault, the test case should result in a failure condition. Such behavior would indicate that the exception management of the target Web service needs to be enhanced.

*Challenges:* Automating negative and positive test case generation requires sophisticated tools that are Web services aware. With a focus on curtailing costs, IT shops are likely to attempt developing ineffective and inefficient heuristics-based negative and positive tests cases by using existing Web site testing tools. Many SOA deployments, however, realize the required measure of ROI associated by investing in SOA tools that automate test generation.

## SOAP Attachment Blind Spots

### 1. Attachments are Incorruptible

SOAP is commonly used to transfer complex attachments such as MRI images, mutual fund prospectuses, and tax returns. Any binary content can be transferred as SOAP attachments based on MIME, DIME, and MTOM standards. When transferring attachments from a client to a SOAP server, the tester can get an HTTP "200 – OK" response status, yet content may be corrupted during the transfer. Relying on just HTTP codes for evaluating SOAP attachment transmissions can result in a blind spot that gives the SOA tester a false sense of success.

*Remedy:* To address SOAP attachment testing, consider the following options:

- Develop regression suites that pre-calculate the check-sum (MD5 or SHA-1) of a message before transmission. After the SOAP attachment is transmitted, the tester should retrieve the attachment and re-calculate the check-sum and make sure that the upload and download values match.
- Ask the Web services developer to return the check-sum value of the document after the upload is complete. The tester can then compare this value with the pre-calculated value. This test technique eliminates downloading the document; however it requires developers to add functionality to their code for calculating and returning check-sum values.
- Consider WS-Security. For sophisticated SOA deployments wherein the WS-Signatures technology-set is already established, the test client can sign the SOAP attachments and the Web services can verify the signature on the attachment. This signature verification process ensures the integrity of the transmission.

*Challenges:* The SOAP attachment process requires a degree of sophistication within an SOA deployment. Although it provides significant ROI through electronic transmission of large documents (compared to snail mail), regulatory mandates require attachments to be encrypted and signed. Such requirements subsequently have to be thoroughly tested by QA professionals. Investing in sophisticated tools that can handle various attachment standards, security nuances, and interoperability issues provides significant efficiency and accuracy in testing service-oriented solutions.

## 2. Size Does Not Matter

Well actually it does! SOAP attachments come in all sizes ranging from Kilobytes to Gigabytes. The attachment handling capabilities of Web services endpoints are limited depending on the vendor-specific memory handling schemes in use. In real life, attachments (such as a tax return by large corporation) can actually have a size of a Gigabyte or more, and may even go well beyond the two Gigabyte addressable memory limit of 32-bit architectures. Web services typically fail to handle SOAP attachments in this range.

*Remedy:* Here are options for addressing SOAP attachment size testing:

- Understand the business requirements for SOAP attachment sizes and build regression suites that identify the break points for message sizes. Ensure that each Web service behaves elegantly for messages that are larger than the established limit.
- Communicate the tested and established attachment size thresholds to trading partners.

*Challenges:* Generating large SOAP attachments can be challenging. Developing, maintaining, and running regression tests for large SOAP attachments requires a significant commitment in time, effort, and cost from the QA Team. Identifying tools that can generate large attachment loads and provide WS-Security for over-sized SOAP attachments is necessary for deploying reliable service-oriented solutions.

## WSDL Blind Spots

### 1. Testing within the LAN is Sufficient

Web services operations, input and output messages, and message data types are defined in detail within a WSDL document. Developers attempt to abstract common data structures (such as Business Address, Purchase Order Items, etc.) as reusable data "types" expressed as XML schema definitions. These schema definitions are then hosted externally so that they can be reused by WSDL definitions across the enterprise. Web Service developers simply import or include these schema definitions within their WSDL definitions through a URI.

One of the most common blind spots experienced by SOA testers is that they consume these WSDL definitions within their internal LAN test harness and successfully run their test suites. However, once the WSDL definition is handed to

an external trading partner, more often than not, the schema URIs are inaccessible through the firewall. This blind spot becomes more significant as reuse within an enterprise increases.

*Remedy:* Below are two techniques for addressing this issue:

- Test the WSDL definition both internally and externally by having tests repeated from the WAN outside the corporate firewall. This closely simulates the external trading partner experience and ensures that the Web services do not break when handed to external consumers.
- Flatten the WSDL definition when handing it over to external trading partners. Embed the schema content inline so that all schema information is packaged together with the WSDL definition. With this approach, external URIs are no longer required for accessing schema definitions. WSDL definition flattening may be the only viable option where firewall restrictions prevent access to schema definitions hosted within the corporate network.

*Challenges:* WSDL definition complexity continues to increase and with an eye towards reuse, companies are abstracting schema definitions and still making them accessible via URI imports.

## 2. WSDL is for Developers

WSDL definitions are the single most important contract between consumers and producers within a Web services-based SOA. The WSDL definition is typically generated or authored by developers with the intent of enabling consumers to integrate. The quality of a WSDL definition may therefore vary.

When generating WSDL code, a WSDL definition is as good as its WSDL generator and the parameters that the developer provides for the generation process. For top-down approaches where the WSDL definition is built by the developer before coding the Web services, hand merging multiple WSDL operations, schemas, and bindings can be challenging when an organization has insufficient WSDL and XML schema expertise on-hand.

In both hand-authored and auto-generated WSDL definitions, the developer may perform simple unit tests and fail to identify issues with the WSDL definition. For example, a number of IDEs provide default namespaces (namespace=http://tempuri.org) that the developer may not change. This may cause namespace collisions when intermediaries attempt to aggregate and merge WSDL definitions from different internal sources. QA testers have as much, if not more responsibility, in understanding and ensuring the quality of WSDL definitions and their associated XML schema definitions.

*Remedy:* To address WSDL quality issues, consider this:

- Dive into the WSDL definitions and understand how to identify WSDL quality.
- Enable a WSDL quality assessment process that provides feedback to development teams on the quality of their WSDL definitions.

*Challenges:* QA professionals should become an integral part of the Web services development process early on. A SOA project's WSDL definitions should be well understood by the QA team and since WSDL can capture business specific information, the QA team should also understand the business objectives in the initial phases of the project.

## Interoperability Blind Spots

### 1. All Web Services Talk to Each Other

Interoperability assessment is the single most important part of a Web services deployment. A service is developed for consumers independent of language, operating systems, or hardware that the consumers run on. Typical Web service consumers are written in Java, .NET, and PHP. QA testers are responsible for ensuring that the published WSDL definitions are consumable via the languages used by the consuming applications.

*Remedy:* To address SOAP interoperability testing, the following options should be pursued:

- Adopt SOA testing products that run interoperability tests for WSI-Basic Profile and WSI-Basic Security Profile compliance. Such tests enforce requirements that enable cross-language interoperability. As an example, WSI-Basic Profile mandates that the SOAP message be of type document/literal and not rpc/encoded.

- Establish a test harness that includes Web service consumers in popular languages such as Java, .NET, and PHP and across popular parsers such as AXIS and XFire.
- Test across different versions of a language framework. For example, .NET WSE 2.0 uses AES-128 for bulk encryption. WSE 3.0, however, defaults to AES-256.

*Challenges:* Building a comprehensive testing platform for SOA is challenging especially when wide proliferation is desired. As more complex operations are added to the mix (such as WS-Signatures, WS-Encryption, and attachment handling) the interoperability problem exacerbates because configuration options are now available that need to be tested. Investing the upfront time and effort in building automation suites based on commercial SOA testing products is essential when facing the challenges of deploying a highly interoperable, service-oriented enterprise.

## Conclusion

The promise of an agile, Web service-based SOA lies in fostering and growing reusability across distributed environments. The ease by which Web services can be developed ends up putting a significant burden on SOA testers to ensure that Web services are and will remain robust, reliable, secure, and scalable. Through collaboration with development teams, an increased understanding of Web services technologies, and comprehensive testing tools, an SOA tester can ensure that SOA-specific blind spots are reduced and perhaps even eliminated.

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL



[Home](#) [Past Issues](#) [Contributors](#) [What is SOA?](#) [SOA Books](#) [Legal](#) [RSS](#)

Copyright © 2006-2007 SOA Systems Inc.

All Rights Reserved