Engineering Software Quality in Service Oriented Architectures (SOA)

J Bocarsly
Division Manager
RTTS

Introduction

One of the hottest trends in business information is the move toward Service-Oriented Architecture (SOA) as the new paradigm for enterprise computing. What does SOA really mean? And, more important, how can the enterprise “engineer-in" SOA quality?

SOA can mean somewhat different things to different organizations, but the gist of it is that enterprise technology can be (re-)architected so that all major systems (databases, feeds, middleware, message queues, application servers, mainframe applications, etc.) can communicate using a “neutral” medium to provide *independent*, *generally consumable services* to the enterprise (without technology-specific protocols and bindings that characterize many proprietary products). Typically, SOA is implemented using SOAP Web Services (http://www.w3.org/TR/soap12-part1/), but this is not required by the basic concept. Figure 1 shows schematically how a typical SOA architecture might look.

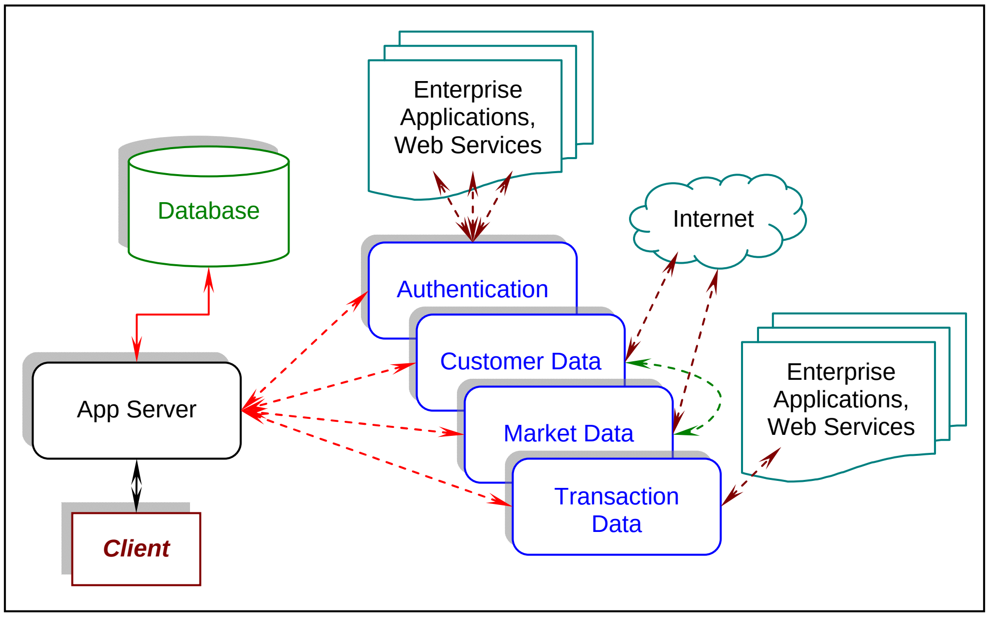

Figure 1. Schematic of an SOA Architecture.

In Figure 1, four SOA Web Services (Authentication, Customer Data, Market Data and Transaction Data) are all generally available for consumption within the enterprise. The services all offer a broadly consumable set of offerings, and each service’s inner structure and workings are invisible to the enterprise. They all operate with relative independence from one another. As shown, our application server uses all four, while two of the Web Services cross-communicate in a background process to enrich their data. Other applications and services in the enterprise and beyond can consume or feed any of the four services while our application consumes data from them simultaneously. The broad flexibility afforded by SOA tends to favor architectures that are topologically simpler than previous architectures. In Figure 1, this is illustrated by the need for only a single Authentication service used globally in the enterprise, as opposed an enterprise dependent on multiple proprietary authentication services.

To formalize a consensus definition of an SOA architecture, we can turn to Thomas Erl’s catalog of

SOA characteristics (http://www.serviceorientation.org/p0.asp):

- Service reusability – Logic is divided into services with the intention of promoting reuse.
- Service contract – Services adhere to a communications agreement, as defined collectively by one or more service description documents.
- Service loose coupling – Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.
- Service abstraction – Beyond what is described in the service contract, services hide logic from the outside world.
- Service composability – Collections of services can be coordinated and assembled to form composite services.
- Service autonomy – Services have control over the logic they encapsulate.
- Service statelessness – Services minimize retaining information specific to an activity.
- Service discoverability – Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

For the quality engineer tasked by the enterprise to evaluate and monitor the quality of the architecture, what do these characteristics mean? In terms of designing a testing strategy to meet enterprise needs both pre- and post-deployment, the list above reduces to three major considerations:

- Loose coupling of SOA components
- SOA components tend to be stateless (i.e. the current status of any transaction is generally *not* preserved by SOA components)
- SOA components provide a layer of abstraction (reusable, autonomous), hiding technologies and components behind the SOA interface

Therefore, to engineer SOA software quality, test implementations will need to:

- Respect the loose coupling of services, and test at the SOA component level
- Respect the stateless nature of SOA components and be aware of repositories of transaction state
- Break through the abstracted layer on an as-needed basis to vet “hidden” components behind the SOA interface

SOA quality efforts will also have to reflect other trends in the general technology space:

- Internationalized SOA Services
- Functional/Regression testing *and* Performance testing
- How quality should be an inherent part of the SOA stack

Finally, SOA is inherently technology-neutral, and may be implemented with any interoperability standard (RPC, DCOM, ORB or SOAP Web Services); because the SOAP Web Services appears to be the dominant direction, we’ll focus on it in our discussion. However, the discussion below generally applies to SOA regardless of the interoperability technology.

Loosely Coupled Testing

One of the basic strategies for SOA testing is to take the loose coupling of services at face value, and

design automated test suites that address each service in the architecture in an individual, or decoupled, mode. In practice, this means that each service, based on its documented interface, needs to have each of its method calls vetted for both proper data responses and appropriate performance. Treating each service as a separate problem in software quality will provide dividends to the development effort, as the library of automated suites will provide an efficient path to issue localization. A significant portion of any development effort is spent debugging during system testing, when the main question is: exactly where in the architecture *is* the problem? As architectures trend towards greater complexity (and SOA will only enable this trend), localization of issues, once they are observed at one entry point, will require increasing amounts of effort. To the extent that the task of issue localization within an architecture can be automated (and this applies to both data integrity and performance issues), development cycles can be shortened and costs contained.

The Stateless Nature

While the recommended design for SOA components is a stateless one (meaning that the status of a transaction, or where the transaction is on the business coordinate at any time, is *not* maintained), that does not mean that SOA-supported transactions cannot be stateful. If the entire Web service infrastructure is implemented as a set of stateless transactions, then the software quality effort simply needs to mirror the architecture and ignore state considerations. However, state may reside in specific locales in an SOA architecture; for example, a specific service within the architecture may be assigned the task of persisting state. Another place where state may reside is in the client applications that consume the stateless services of the SOA architecture. Either way, the preservation of state by any one component for the whole means that the entire SOA architecture behaves in a coordinated fashion ("the architecture is the computer")[1]. Test automation must be cognizant of SOA architectural behaviors – automation code will have to persist transaction state, either because it is mimicking the behavior of a client application or because it uses its' own record of transaction state as a baseline against which to check the architecture's record of the state, or both. The message here is that automation code may have to be stateful if state is relevant to the transactional nature of the SOA architecture. The time-line for the development of automation code will therefore have to accommodate the additional complexity of incorporating transaction state.

Breaking Through Abstractions

Testing loosely coupled SOA components at the component level resonates well with the basic concept of the SOA architecture, as noted above. But quality engineering may demand more than that: it may demand that testing go behind the SOA "curtain", and verify the behavior of the SOA service against the underlying technologies and components. The simplest example of this would be a lookup service in which a Web service exists as an interface to a database. Requests to the Web service results in database queries whose results are reported by the service in its response messages. The task of quality engineers in this case would be to validate the data returned by requests into the Web service against data returned by analogous SQL queries directly against the underlying database. More complex situations arise when a Web service itself has a complex underlying architecture. For example, a service may aggregate data from multiple backend sources including databases, live feeds, and other Web Services (i.e. SOA components). Depending on the profile of the target Web service, testing the aggregate Web service against all of its data sources

[1] "The Network is the Computer", J. Gage (http://en.wikipedia.org/wiki/John_Gage).

may be a high priority task on the pathway to SOA quality.

Internationalization

As the global enterprise packs more data and services in the SOA package, one challenge that will arise with increasing frequency is that of internationalized enterprise Web Services. Enterprise services will “speak” multiple languages, with request and response messages encapsulating data in multibyte character sets. Automated testing strategies will have to support internationalized service architectures by working with data sets for each character set and locale supported by the architecture. The de facto standard for the Web is the UTF-8 encoding of Unicode, and standard SOAP/HTTP systems all support it. (XML files can be edited directly in Unicode characters where a Unicode-enabled editor is used, or Unicode can be entered in the form of numeric entities using an ASCII editor.) The major implication for the testing process is that language-knowledgeable resources will have to contribute data across the supported character sets, and that quality engineers will have to maintain multiple data sets (both request data and baseline response data) for each transaction. Automation code will have to be built to iterate through each transaction multiple times (once for each character set). Results reporting will have to accommodate output in each character set, so that language-knowledgeable resources can participate in issue analysis.

Scope: Functional/Regression *and* Performance Testing

Throughout the discussion above, we have generically referred to ‘quality engineering’, ‘test automation’ or ‘SOA testing,’ without specifying much about the contents of these activities. In fact, the two main testing concerns for Web Services, data validation and performance, are in scope at all levels. Both functional/regression and performance testing efforts need to address each service in a decoupled fashion (i.e. without interactions with other services). Both must address any coupling of services due to persistence of state. Finally, both performance and regression tests will have to examine each service “under the hood” to verify the data and performance behaviors of the underlying components (databases, live feeds, middleware components, etc.) Another way to put this is: the entire SOA architecture, at all levels, feeds into the quality of the whole, and a badly malfunctioning service, which may only be reflecting the behavior of one of its underlying components, can dramatically affect the whole architecture. The ability to localize any issue rapidly, whether it is data-related or performance-related, is critical to establishing and maintaining a quality SOA implementation.

Data

Clearly, with the range of functionalities that are supported by many SOA architectures, data for testing purposes is a critical part of the software quality process. There are multiple tools in the market space that interface with the data aspect of testing and quality; these generally come in two flavors in regard to data: capture/playback tools and data entry tools. Vendors of capture/playback tools will argue that your quality team will be able to satisfy all testing-related data needs by simply capturing a volume of the live messaging on your architecture and storing it for playback through a tool. These vendors will argue that the task of “manually” assembling a “complete” data set to fully test all your architectural components is insurmountable, due to the massive complexity of the data and the architecture. This latter point is not a bad one; frequently, systems *are* so complex that assembling a full set of “synthetic” data would require an unimaginable resource request. However, relying on a sample of current traffic is equally fraught with difficulty: there is absolutely no guarantee

that captured data will contain every critical transaction that you must test. In fact, while data capture is not a bad way to start assembling a full data set for SOA testing, be sure that any data-capture-based tool you invest in offers you the additional abilities to massage captured data, and to input "cooked" data to the library of recorded transactions. With both capabilities, you will be able, over time, to address all test data requirements.

A Modest Proposal

In the spirit of furthering what SOA should mean for quality-engineered data and services in the enterprise, RTTS proposes that part of the evolving definition of the Service Oriented Architecture should include Self-diagnosis or Self-monitoring. That is, every enterprise SOA architecture should include a service that provides real-time information on the health of the rest of the architecture. Technically, this is only a matter of wrapping one or more of any number of post-deployment monitoring tools in the market in an SOA wrapper. Vendors of post-deployment monitoring software should play a role in this – by releasing their tools with Web Service wrappers. But, prior to vendor action, every architecture team should make self-diagnosis a feature of its SOA architecture. There are multiple server-monitoring tools in the market space already (your organization may already have licenses to one or more), and any state-of-the-art Web Services SDK has tools for quickly wrapping existing functionality in a SOAP wrapper. The addition of a self-monitoring service is both technically feasible and business critical – every team responsible for a deployed architecture needs to know its' real-time status.

The addition of Self-diagnosis will help mature the SOA paradigm, so that SOA includes the notion of discovery not only for the locations of constituent services, but also for the health of the entire SOA enterprise architecture. Including a service to monitor services, SOA will widen its existing definition of service discovery to service discovery and service health and availability.

Summary

SOA is the hottest paradigm in enterprise architecture right now. Critical to the deployment of quality SOA components are quality and testing strategies that reflect the SOA concept:

- Respect the loose coupling of services, and test at the SOA component level
- Respect the stateless nature of SOA components
- Break through the abstracted layer on an as-needed basis
- Prepare to handle internationalized SOA Services
- Include both Functional/Regression testing *and* Performance testing in your quality effort
- Build an SOA stack that Self-monitors

About the Author
Jeff Bocarsly

Jeff has successfully implemented automated software testing projects at many Fortune 500 firms over the last six years. His experience includes projects in various sectors including financial services, software vendors, media, pharmaceutical, and insurance / reinsurance companies. He specializes in implementing test automation and methodology solutions. Jeff is an expert architect with specific expertise in application development and programming and is co-author (with Dan Chirillo) of the upcoming book "Automating Testing of HTML and Java Applications With IBM Rational Functional Tester".

Jeff's prior experience comes from the academic field, where he was a professor of chemistry at the University of Connecticut. Jeff also provided UNIX system and LAN administration. Jeff also was a postdoctoral associate at Yale University.

Jeff holds a BS from UCLA, a Masters Degree and Ph.D. from Columbia University.

About RTTS
RTTS is the premier professional services organization specializing in the testing of IT applications and architecture. They have been serving Fortune 500 and mid-sized companies in many vertical industries including, financial, insurance, telecommunication, pharmaceutical, and technology firms in the U.S and abroad since 1996. RTTS has offices in New York, Orlando, Phoenix, and Philadelphia. RTTS draws on its best-of breed tools, expert test engineers and proven methodology to provide the foremost end-to-end solution to ensure application functionality, reliability, scalability and network performance. To learn more about RTTS visit www.rtts.com.