

# AUTOMATING

*WHAT YOU CAN'T SEE*

## **Testing Middleware for the Enterprise**

By: Jon Howarth and Robert Ryan

*June 29, 2006*

## Table of Contents:

<b>Abstract</b>	3
<b>Problem: Application is Complex</b>	3
Data Oriented Messages	4
API	4
Rapidly Changing	4
Multiple Client Interfaces	5
Can not do pure black box testing	5
<b>The Problem: Environment is Complex</b>	5
<b>Problem to Requirement Mapping</b>	6
Problem: Limited Tools for Business Process Validation and SOA Testing	7
<b>Solution: Suite of Four</b>	8
Manual functional testing	8
Organization and workflow	8
Automated execution	9
Data Management	9
<b>Solution: Architecture</b>	9
<b>Summary: A Solution is possible</b>	10

## Abstract

Testing enterprise middleware that has a service oriented architecture (SOA) front-end and dozens of back-end connections is a daunting responsibility. In this paper, we relay our experience of creating a suite of tools to test what is possibly one of the world's largest middleware implementations. We start out with a comprehensive review of the problems of testing SOA middleware applications. Second, we review our requirements for the tools that we eventually acquired, the tools that the market offers, as well as what is needed but can not be found in the marketplace. Finally, we conclude with one solution that has been in use for over a year, has executed hundreds of thousands of tests, and certifies the functionality of systems that execute over a billion transactions per month.

### Problem: Application is Complex

SOA based middleware applications are inherently more complex than the typical GUI-driven enterprise applications. SOA based middleware presents the world with an inflexible machine interface that is not intuitive for humans to interact with. Messages require and return far more information than an ERP screen and the formats of those messages are designed to conform to enterprise data architectures – not the essentials of machine-human interactions.

Top most in the list of complicating factors is that middleware is not build with the viewer in mind, and why should it? In 'production' nobody ever sees the inputs or the outputs of the application. Testers can not be thrown at a web service and be expected to find the obvious problems. They can-not simply fill in the create customer screens with obvious data and see if bad data throws an exception. They need to know how to generate a contract-compliant request, and that requires training and knowledge in the fundamentals of SOA. Before a message is passed on to the business-logic processing portion of a web service, it is generally inspected for adherence to an intricate set of rules. If the tester does not understand these rules intimately, there is no hope of even finding the 'obvious' problems.

Testers will never finish testing an application if they are forced to work with text editors to create requests, low-level network HTTP protocols to send and receive messages, and sharp eyes to determine if a response is acceptable. XML responses can be returned in any number of permutations and still be correct. The eye is a poor identifier of ill-formed XML and incorrect values.

Testers need something user-friendly to work with. They need a human-readable interface to the WSDL, contract, request and response SOAP. They need tools to generate template requests that can be manually tailored for exact test cases. They need a tool to programmatically compare an expected XML document with an actual document to determine if they are materially different.

## **Data Oriented Messages**

Web services messages are designed to contain all the fields that are needed to process a particular business function. These functions are executed without any dependence on a predefined state, which is vastly different than a set of GUI screens. A web service does not have the context of who is logged in and what customer they are interested in. All of these attributes must be passed in the request message. Since each message must contain several fields, it creates a vastly larger set of permutations than what can exist in typical enterprise applications with a graphic user interface.

Testing this vast number of permutations is generally neither possible nor preferable in terms of costs and benefits. The most important scenarios must be identified and stressed. Ancillary permutations must be abandoned. This demands correctly identifying the most relevant test cases. For the tester who does not have visibility into the underlying constructs to message processing, there is only the specified functionality that can identify those most important paths. Requirements-based testing tools can map out the core flows to be tested. However, testing efficiencies such as automation and data and test case management are essential to releasing web services that meet expectations.

## **API**

Web services are inherently technical and low-level interfaces into the business functions of the enterprise. This is because other applications demand rigidity and predictability to interact with each other. For some test cases a tester can-not allow a response XML to format dollars with a dollar sign. This would be a showstopper since dependant applications are expecting a floating decimal and can not process the non-numeric character. This level of strictness is unusual for humans but is the most efficient way for computers to interact with each other. This strictness demands a tool that can inspect the contract (WSDL) and compare it with requests and responses sent to and from the application. XML elements can be returned in different orders, and still comply with the contract. For a person comparing two XML responses with different orders it is a daunting task to spot the defect. A small variation in a data type such as the number of decimal places or capitalization is a defect that could cause a total failure. These are practically impossible test cases to verify by brainpower alone.

## **Rapidly Changing**

The landscape of a competitive enterprise application is constantly changing to adapt to market forces. SOA based middleware is exposed to these changes since new application functionality is constantly evolving and these evolutions constantly demand more and more interaction with other systems (products) in real time. For example, an ATM system today will offer free upgrades to checking accounts tomorrow. These changes are needed for the modern enterprise to stay competitive. The cost of these changes is a constantly evolving ecosystem of interactions between systems, all built on the backbone of service oriented architecture.

## **Multiple Client Interfaces**

Most systems interact with each other in a synchronous, request-response manner, but some systems run in a batch mode, fire off millions of requests, and do not care to wait around for responses. Thus middleware must offer asynchronous queuing-based interfaces. These interfaces do not have the same request-response based universal standards that SOAP-based web services offer. There are several protocols for entry points to these queues such as TIBCO, webMethods, and MQ. Each of these protocols add complexity for the tester as well as increases the number of permutations of a test case.

## **Can-not do pure black box testing**

Enterprise middleware pulls and modifies data from disparate systems. Many business processes provide only a confirmation message that an update has occurred. If this update affects multiple systems there is frequently a need to inspect these systems, directly for a successful post-condition. Since middleware provides only those functions necessary to support the business-level needs it rarely includes functionality only a tester would need to verify that an update has reached all the necessary destinations. This can mandate logging into an SQL database and verifying a row has been created or inspecting a log file to ensure a message has been processed from a queue. If testing is to be automated, then the tool needs scripting capability the tester can author, including the capability to setup a data condition in preparation for a test. Also, part of the role of the tester is to provide enough information about a defect for the developer to understand or recreate the condition. This requires pulling tracings and data about a test result from sources across the enterprise.

## **The Problem: Environment is Complex**

One of the reasons companies are increasingly turning towards SOA is to reduce the costs of application development through simplification of the integration costs. While this may reduce the complexity of interfacing with the environment for client applications, it often leads to an increase in complexity in the SOA application itself. The SOA application needs to integrate with technologies varying from web service and other middleware applications to CICS legacy applications. As the clients move more toward leveraging the SOA infrastructure, the number of integration points often increases. This increase in environmental touch points leads to a very complex testing and data management problem.

The companies turning to SOA are often large institutions like health care and banking, where much of the environment you are forced to test is out of the control of your testing organization. This environment is usually not dedicated to the testing of your application, but is often a shared environment used by many applications. The subsystems that house the data your tests depend on are also out of your control. You cannot stop the business processing in the dependant subsystems, for example, in a banking environment; accounts with a zero balance will cycle and close each month. This data volatility makes automated testing difficult because you must have a deep understanding of the applications and people in the subsystems your application depends on. The initial setup of the data in these dependant subsystems is difficult since it is necessary to either pay

another group to setup the data or train team members within your organization on each of the dependant subsystems.

Not being the owner of the subsystems and the sheer number of the dependant subsystems makes traditional data management strategies impossible. Some of the reasons these traditional strategies are incomplete include:

- Systems to generate data for your automated tests at runtime may not be present
- The data creation may require coordination between multiple groups
- The data creation may require mainframe cycles, which the entire enterprise is dependant upon and cannot be run at your group's leisure
- 3rd party involvement in the environment
- The large number of subsystems which house the data and their interdependencies

## Problem to Requirement Mapping

To be a successful testing organization in an environment laden with a multitude of application and test environment challenges, you will need a suite of tightly coupled test tools. The identification of these tools begins with a detailed exploration of your application and environment. The current testing challenges must be identified and the future testing issues must be planned. The heart of the search for a good tool set begins with the definition of the testing tool(s) requirements. The application, environmental, and tester needs will dictate these requirements.

A diverse range of skill sets is needed to appropriately define the testing tool requirements. The key skill sets to include in the requirements definition phase include team members with automated testing experience, environment experts, application subject matter experts, and management.

The tool requirements depend completely upon the scope of the undertaking. Some of the key requirements required for automating SOA applications in a gray box environment include:

<b>Problem</b>	<b>Requirement</b>
Application has no GUI	Testers want a user friendly testing interface. Ability to generate a request from a WSDL
Data intensive messages	Support for the automation of many data permutations
Application has strict (API) interface	Support to programmatically verify the response is contract compliant.
Application is rapidly changing	Support for flexible test suites and the rapid execution of these test suites.

<b>Problem</b>	<b>Requirement</b>
Application supports multiple protocols	Support for HTTP/HTTPS and MQ protocols
Testing is not black box and requires validation outside of response message	Support for an automated validation of files, and databases, and the ability to automate complex validation scenarios (customizable)
Environment dependencies are outside of our control	Support for managing data needs in dependant systems
Data maintenance is costly	Support for data re-use across test cases

### **Problem: Limited Tools for Business Process Validation and SOA Testing**

As we have demonstrated, the application and environmental requirements related to SOA testing are extremely complex. The multiplier effect across disparate technologies, applications, protocols, and processes create a significant testing challenge. Our testers wanted the ability to reliably test interfaces across and between distributed, heterogeneous systems, not just independent systems. We quickly realized that no one tool was going to be able to solve all of the issues associated with testing SOA application quality in the enterprise environment. Functional and UI testing tools covered end user interactions, but lacked the coverage and depth to treat the integration middleware as anything other than a “black box.” In addition, existing homegrown testing harnesses were tailored to specific scenarios, but they were costly to maintain, required technical resources to use, and did not integrate well with other testing tools.

This was a concern to us because, as a goal, we wanted to validate composite business applications and processes that accessed a multitude of systems. Additionally, while we felt that automated testing would lower our project risk, we wanted to ensure that the cost of maintaining a complex and comprehensive set of automated tests did not exceed their value. Finally, we wanted to minimize the impact of having to require our testing team to become productive with multiple testing tools, desiring to find an integrated, cooperative solution.

## Solution: Suite of Four

No one tool was able to solve all of the issues associated with testing SOA based middleware applications in the enterprise environment. The solution was to define the requirements for testing in an enterprise environment and identify a suite of tools to meet these requirements. Each tool had a distinct role and responsibility in compiling the complete solution. Even though the tools identified overlapped in multiple areas, we were able to strictly assign responsibilities between the tools. The defined roles allowed for each tool to do what they do best.

*The suite of tools was broken down into four distinct areas to meet the requirements:*

- Manual functional testing
- Organization and workflow
- Automated Execution
- Data Management

### **Manual functional testing**

**SoapScope from Mindreef** was identified as the manual functional testing tool. SoapScope is an industry standard web service testing tool with capabilities for one off functional testing. The key requirements that this tool met include:

- Generation of a SOAP/XML request from a WSDL, Web Service Definition Language
- Ability to inspect the WSDL and SOAP request in a pseudocode view, which is easier to read than a raw (text) view
- Ensure contract compliance of the SOAP requests

### **Organization and workflow**

**TestDirector for Quality Center from Mercury Interactive** was identified as the organization and workflow tool. This tool played a key role in the management of the test assets. The requirements this tool fulfilled included:

- Test case management
- Test execution management
- Workflow management (the life cycle of a test case and test script within the QA organization)
- Rapid response to testing needs such as on-the-fly creation of targeted test suites
- Integration with the automated execution tool

### **Automated execution**

**Integra Enterprise from Solstice Software** was identified as the automated execution tool. It provided a

testing workbench to give developers the ability to build automated regression tests to handle complex testing scenarios across multiple protocols. It also offered fully automated simulation and business process validation capabilities. Further, the tool:

- Supported the needed protocols (such as HTTPS and MQ) for isolating and testing the applications under test (SOA), which was critical to validating the business processes and comparing actual results against a baseline.
- Was straight forward and easy to use, yet comprehensive and flexible enough to support our specific needs
- Provided a File Adapter to allow us to automate the review of log files
- Supported introspection of message formats and externalization, review and comparison of message data created from scripts
- Integrated seamlessly with TestDirector for Quality Center, allowing us to manage all of our automated testing centrally

### ***Data Management***

No tool in the market was available to meet the needs of the data management requirements. The implementation team was forced to develop an in-house **Data Repository** to meet the requirements. The key requirements met by the data repository include:

- A searchable repository for test data, which promotes data re-use
- The storage of not just the data but also the data conditions
- ODBC interface to populate SOAP requests at run time

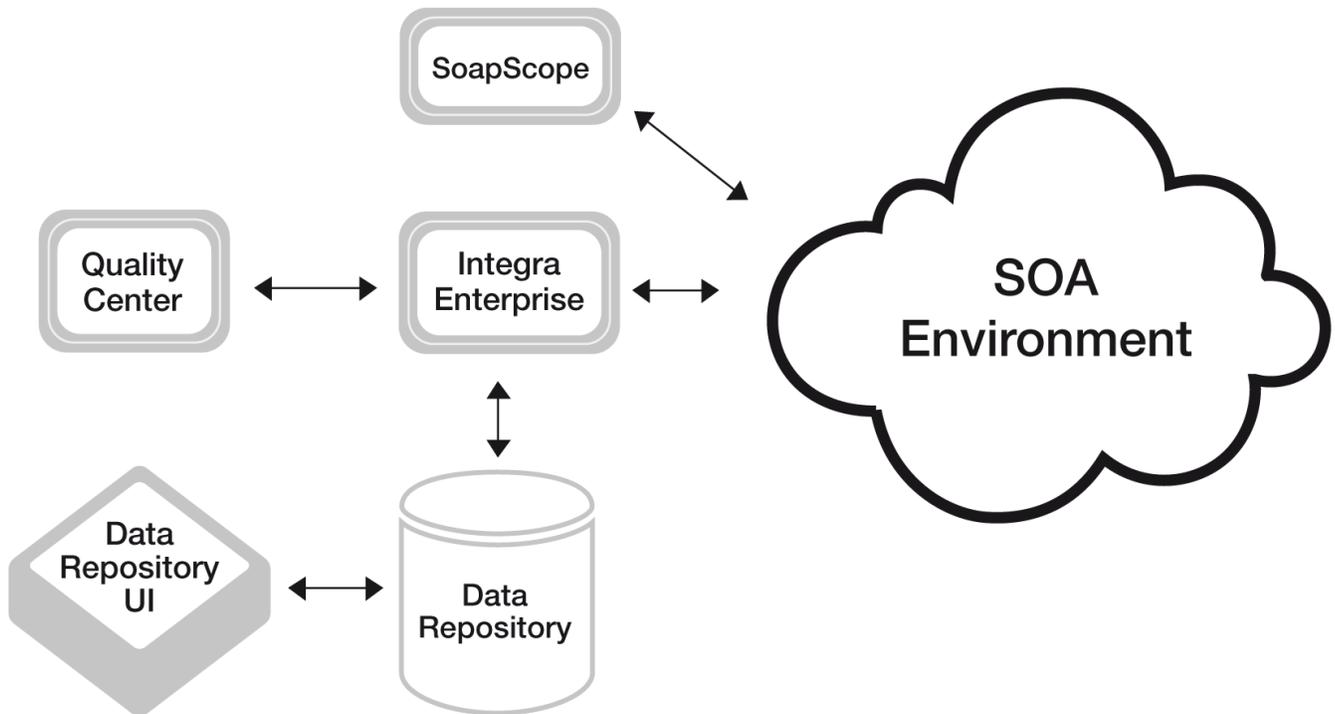
## **Solution: Architecture**

The suite of tools together creates a robust platform for testing in an SOA environment. The responsibilities, and interfaces between the tools are well defined with flexibility built into the integration.

*The lifecycle of a test case in the platform illustrates the good workflow and integration of the tools:*

- The test case is created from a message specification in the test plan module of Test Director for Quality Center
- The test case data needs are described in Test Director for Quality Center
- Test data for the test case is searched for in the Data Repository
  - If the test data is not in the data repository, the test data is created in the dependant subsystem

- The SOAP XML request is generated in SoapScope from the WSDL
- The SOAP XML request is automated in Integra Enterprise
  - The test case in Test Director is 'linked' with the test script in Integra Enterprise
  - The dynamic data is extracted from the SOAP XML request
  - The data and its associated data conditioning is logged in the data repository
  - Comparison rules (For example, ignoring dynamic fields like session id) are authored in Integra Enterprise
- The test case is executed from Test Director for Quality Center



## Summary: A Solution is possible

Much of the complexity of testing middleware is rooted in the dependencies upon other systems, and those systems are usually out of the control of the middleware testing group. The data requirements are the primary dependencies, but the versions of the APIs that middleware use to interact with the dependent systems can also greatly complicate testing. Because so much is out of control, it is fruitless to attempt to control all dependencies – it is better to mitigate. By mitigating we mean maintaining a repository of data conditions that must exist in dependent systems to satisfy testing needs. It also means being able to rapidly assemble suites of tests that verify specific chunks of functionality that are provided by specific back-end systems.

The SOA based middleware applications are constantly evolving. Through the implementation of integrated tools like Test Director for Quality Center and Integra Enterprise, the test teams are able to quickly identify test cases and scripts to provide coverage of the evolving application.

The GUI-less interface to SOA applications, as well as the unique aspects of data-intensive messages that they utilize, requires tools that provide a human-friendly interface like SoapScope and Integra Enterprise. This interface performs much of the ‘heavy lifting’ needed to format requests, as well as automating the execution of tests. The tools also allow for the comparison of expected vs. actual XML results and the real-time verification of post-conditions.

***No one tool can satisfy all of the application, environment, and testing requirements. The only option is to implement a tightly coupled suite of tools to address the requirements.***