Testing Methodology

Document Type:
Test Methodology Description

Editors:
Ram Jeyaraman, Microsoft
Jacques Durand, Fujitsu

Last Edit Date:
12/2/2008 11:05:00 AM

Document Status:
Version 1.0

This document is a Working Group Draft; it has been accepted by the Working Group as reflecting the current state of discussions. It is a work in progress, and should not be considered authoritative or final; other documents may supersede this document.

Table of Contents:

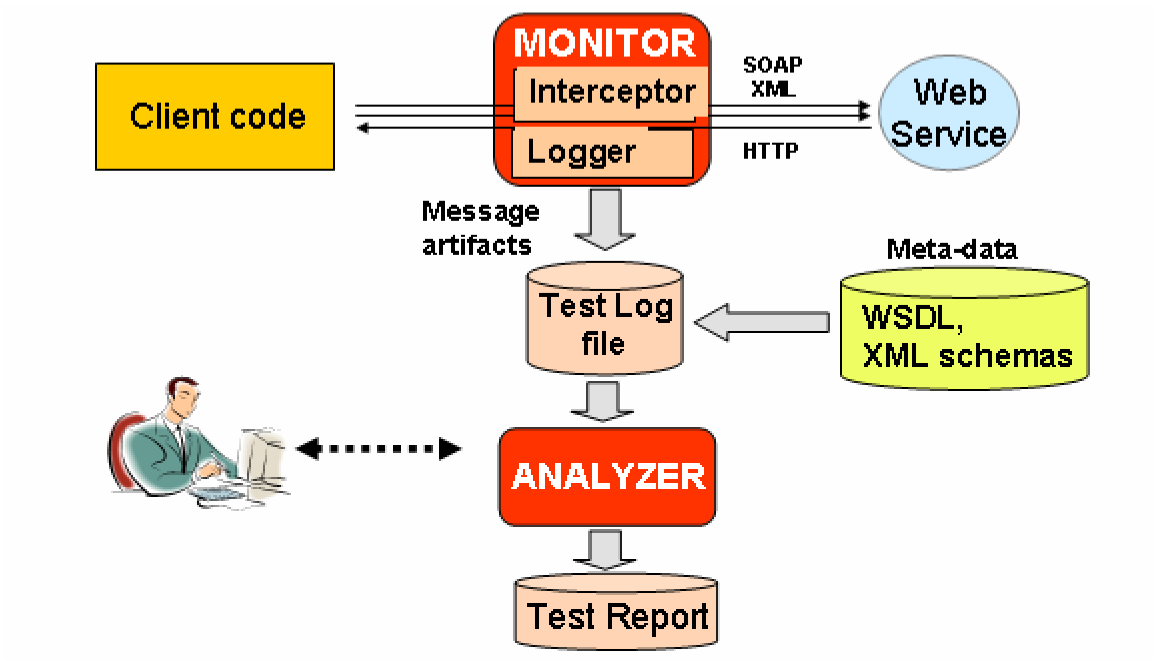
1 Overview

The testing approach used for the latest WS-I profiles (Basic Profile 1.2 and 2.0, Reliable and Secure Profile 1.0) differs from the one used in the past, although overall the general testing process remains similar: a “logging” phase including live message capture, followed by a distinct Analysis phase that involves a separate tool.

The overall test process involves two phases:

- Test log creation, including message capture and metadata consolidation (WSDL definitions, etc.)
- Test analysis, including evaluation of WS artifacts in the test log, against the requirements of a particular profile

The following figure illustrates the test process. A difference with the previous approach is that the Test Log file now consolidates all artifacts under test (messages + metadata).



The above figure represents the run-time test tools: Monitor and Analyzer. The Monitor itself includes a "logger" that generates the proper XML format for the message capture. In order to be able to reuse the previous Monitor from Test Tools 1.0, a conversion program (XSLT) has been created to convert the output of the Monitor 1.0 into the new test log format defined for BP 1.2 and 2.0.

Another tool is used to generate the Analyzer code that is specific to a particular profile (e.g. BP2.0 or RSP 1.0). It is not illustrated in the above figure, but is shown in the next figure. This generation tool takes as input the test assertions (scripted in XPath2.0) that are part of the profile XML document.

What users have to do: End users will only need to access the Monitor and the Analyzer for their adopted profile, not the generator tool. The Analyzer tool could be made available as Software-as-a-Service, since it could reside on a Web server where users could upload their test log file, and browse the HTML rendering of the Test Report. Individual vendor implementations are required to capture XML wire messages and convert them into a message log format as defined by the schema of the log message file format.

The Test Log file is described in Sections 4 and 6.2

The Test Report is described in Section 5.2.

3

2 Test Assertions

A test assertion is a translation of a WS-I profile requirement into a statement verifiable by the analyzer. The following differences must be noted compared with the previous WS-I testing approach:

Previous approach: Test Assertions associated with a profile are defined in a separate XML document called the Test Assertion Definition doc (TAD).
New approach: Test Assertions associated with a profile are defined inside the Profile document itself, as inline XML elements that are closely associated with the Profile requirement they address.

Previous approach: Test Assertions are defined in plain English in their host document (a XML document separate from the profile document), while their executable version is coded (C# and Java) inside the Analyzer tool.
New approach: Test Assertions are scripted in XPath2 and also commented in plain English. Test Assertions are now part of the original profile XML document (although this is not required by the analyzer tool). The Analyzer tool is able to directly process them.

The structure of Test Assertions is still overall similar as in the previous test approach. Some of its elements have been renamed, and some added. A more detailed description is provided in Appendix A.

What users have to do: Users do not need to understand the scripting of test assertions (although this may help them when debugging). They need to understand the profile requirement associated with a test assertion ID, so that they can understand why a particular item in the test log generated a pass/fail test report.

3 Test Scenarios

The RSP WG has developed a set of interop scenarios that are intended to exercise the various aspects of the WS-ReliableMessaging 1.2, WS-MakeConnection 1.1, WS-SecureConversation 1.4 protocols as profiled by RSP 1.0. The scenarios describe a set of message exchanges that exercise specific protocol elements and test specific profile requirements. Please see RSP 1.0 interop scenarios for a description of the interop scenarios.

In the real world, however, the Web services application to be tested for conformance is expected to generate the message traffic and exercise the various protocol elements defined in the relevant specifications.

What users have to do: Users should be able to run test scenarios from their Web service client (or service), and capture the message exchanges using a Monitor tool (see above).

4 Test Log Files

Test Log files are provided as input to the test analysis phase (Analyzer tool), and are produced by the Monitoring phase involving run-time execution of test scenarios and the resulting message capture.

An enhanced version (but not drastically different) of the former “Monitor log file” (the test input file for the Analyzer) has been defined and used by the new test approach. The test input file, also called the “Test Log file” is consolidating test artifacts as diverse as:

- `service definition files (WSDL, Schemas)`
- `HTTP and MIME message headers`
- `SOAP envelopes and content`

Sample test log file material is given in Section 6.2.

What users have to do: After the Monitor tool (see above) has been used, users must convert – in case the Monitor tool does not - the message capture so that it complies with the test log file XML format. All metadata needed by the analysis (e.g. WSDL, XML schemas) must be consolidated in this test log file.

5 Test Analysis

5.1 The Test Analyzer

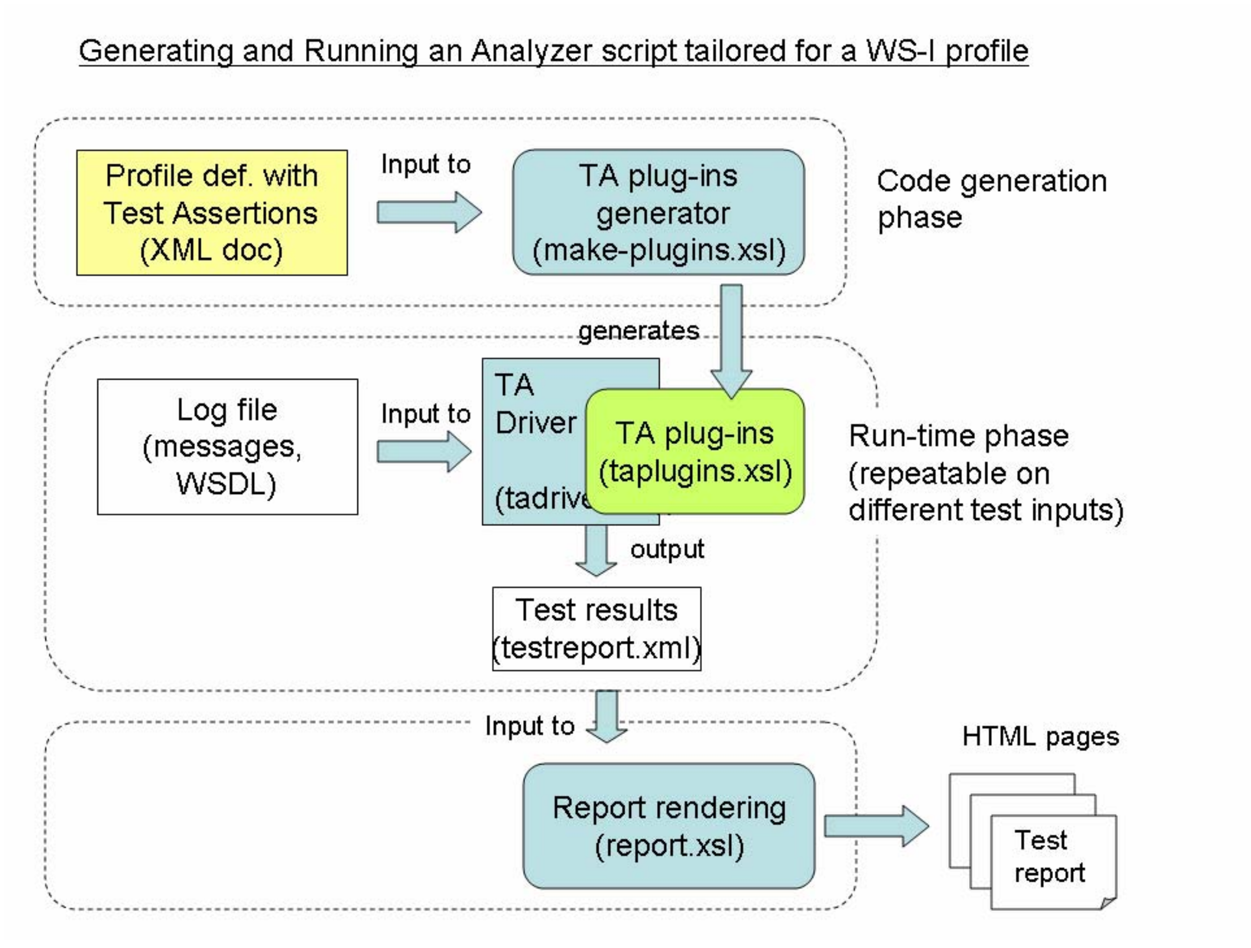
Unlike the previous test approach, where the Analyzer tool was developed in conventional programming languages (C#, Java) the new Analyzer is developed using XML processing technologies that are portable on any major platform, and based on W3C standards.

The Analyzer core module has been developed using XSLT 2.0. It can be run using any XSLT2.0 processor, although some advanced features used in some TAs (like schema validation) may not be available in every offering on the market for which a sufficient offering exists today for both .NET and Java platforms - including freeware such as products from Saxonica and Altova.
The test analysis phase works in two steps. The first step only needs be executed once for each new profile:

Step 1: Generation of the Analyzer code for a particular profile.
Input: the Profile definition document (XML) with embedded TA definitions.
Output: an XSLT script that represents the actual Analyzer tool for this profile.
Processor: a code generator written in XSLT2.0.

Step 2: Run-time analysis of Web services artifacts.
Input: a test log file.
Output: a test Report file in the same format as the previous generation of test tools V1.0, so that the same HTML rendering XSLT script can be reused.
Processor: the Analyzer XSLT tool produced by Step 1 for this profile.

These two steps are illustrated in the following figure:


Generating and Running an Analyzer script tailored for a WS-I profile
Profile def. with Test Assertions (XML doc)
Input to
TA plug-ins generator (make-plugins.xsl)
Code generation phase
generates
Log file (messages, WSDL)
Input to
TA Driver (tadriver
TA plug-ins (taplugins.xsl)
Run-time phase (repeatable on different test inputs)
output
Test results (testreport.xml)
Input to
Report rendering (report.xsl)
HTML pages
Test report

- The “blue boxes” in the above figure, represent code that is permanent and does not change from one profile to the other.
- The “yellow/green boxes” represent code or data that varies from one profile to the other.
- The “white boxes” represent data that changes from one test run to the other.

<u>What users have to do</u>: Users have to run the Analyzer run-time (Step 2 above), to obtain a test report.

5.2 The Test Report

The test report resulting from the processing of a test log file (Step 2) is an HTML document. It provides, for each artifact of the log file that is a target of at least one Test Assertion, a detailed report about this target. In particular, every TA that applies to this target will have its result (passed / failed / warning / NotApplicable / MissingInput) listed for the target, along with error messages when appropriate.

A cross-summary is also provided: for each Test Assertion, how many targets returned each one of the possible results: passed / failed / warning / NotApplicable / MissingInput.

When a test assertion is processed over a target, it will complete with one of the following outcomes:

- **passed**
  The test assertion script completed the verification on the target without detecting any errors. This is an indicator of conformance to the profile requirement.
- **failed**
  The test assertion script detected some error indicating that the target does not conform to the profile requirement.
- **warning**
  The test assertion was not conclusive about the target. The user is advised to more closely examine the target material or focus future testing on this test assertion as there is a potential for conformance failure.
- **notApplicable**
  The target did not qualify for this test assertion, which means that although the target was of the type of artifact relevant to this test assertion, it failed the prerequisite expression or it failed a prerequisite test assertion. In both cases, the test assertion is not relevant to this target.
- **missingInput**
  The test assertion was not processed due to missing collateral input (see “cotarget” in Test Assertion structure in Appendix).

The following figure shows the HTML rendering of the summary report for artifacts of type “message”. The first column lists all the test assertion IDs (here, of the form RSPxxxx) exercised during the test run. Each row in this report reports the summary for a particular test assertion, showing how many messages failed, passed, etc.

Artifact: message

Assertion Result Summary:

Assertion ID	Prescription	Passed	Failed	Warning	Not Applicable	Missing Input
RSP0001	mandatory	2	2	0	0	
RSP0010	mandatory	1	0	0	1	
RSP0120	mandatory	9	0	0	0	
RSP0210	mandatory	2	8	0	0	
RSP0400	mandatory	0	0	1	0	
RSP0600	preferred	1	3	0	0	
RSP2100	mandatory	2	1	0	0	
RSP2101	mandatory	2	1	0	0	
RSP2110	mandatory	1	2	0	0	
RSP2113	mandatory	0	2	0	0	

6 Appendix

6.1 Appendix A: Test Assertion Structure

The structure of a Test Assertion in the new test approach, is defined as follows. This structure maps to an XML mark-up.

- **Description**: this is a plain English statement of the Test Assertion.
- **TA Id**: the identifier of the Test Assertion.
- **Source**: the normative profile requirement that this test assertion is addressing, identified by its requirement ID (Rxxxx).
- **Target:** (formerly “primary Entry Type”) A Test Assertion always targets instances of a specific artifact type, for example, a SOAP Envelope, or a message (itself containing a SOAP envelope) or a WSDL port binding, etc. The Target element identifies this artifact type. The Target contains an XPath expression that defines the set of instances to which the TA applies, when evaluated over an XML file containing message capture and metadata (see Test Log file). The set of instances to be considered can be further narrowed by conditions (XPath predicates) inside the XPath expression. The Target expression was called the “Context” in previous test approach.
- **CoTarget:** (formerly “secondary Entry Type”) A pointer to another artifact that is required to evaluate the TA over the specified Target. It is usually correlated with the Target in some way. For example, the Target may be a request message as captured on the wire, and the CoTarget may be the message parts description in WSDL that relates to this wire message. The CoTarget is scripted as an XPath expression that will select the related material in the Test Log file.
- **Pre-requisite:** A pre-condition that must be satisfied over the Target instance (and possibly its CoTarget) in order for this Target to qualify for this TA. The Pre-requisite may refer to other Test Assertion that must be passed by the Target in order to qualify. If the Pre-requisite evaluates to “false”, then the outcome of the TA for this Target will be “NotApplicable” in the test report.
- **Predicate**: (formerly the “Assertion”) A logical expression that ca be evaluated over the Target (and CoTarget if appropriate). The Predicate is only evaluated if the TA is applicable, i.e. if the Pre-requisite – if any – has already evaluated to “true”. In general, if the Predicate result is “true” then the Target fulfills the profile requirement addressed by this TA. If the result is “false” then the Target violates the profile requirement. However the predicate may be worded in a different way, e.g. to support negative testing. The interpretation of its result in the finbal test report is controlled by the *Reporting* element.

- **Prescription level**: Indicates the level of compliance expected from the Target, to the profile requirement. Possible values: mandatory, preferred, permitted. These values match the RFC2119 keywords MUST, SHOULD and MAY, and their equivalent.
- **Reporting**: Indicates how to interpret the Predicate outcome (true/false) from a conformance point of view in the test report. Possible reporting results are: "passed", "failed", "warning", "undetermined". In an ideal situation, the reporting map (which is also the default mapping) is: true="passed", false="failed". Due to testing constraints, the Predicate may not provide as good an indicator as the default one. It can then be mapped as: true="warning", false="failed" or in other ways.

6.2 *Appendix B: Sample Test Log File (may need be shortened)*

An example is given in Figure 3 below. This test log file contains a pair of messages from an HTTP request-response exchange. This log shows how all the message header content – incuding HTTP headers – has been captured and serialized using XML.

```xml
 <?xml version="1.0" encoding="UTF-8" ?>
wsil:testLog xmlns:wsil="http://www.ws-i.org/testing/2008/02/log/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.ws-i.org/testing/2008/02/log/ log.xsd">
wsil:messageLog>
wsil:message type="request" id="1" conversation="1">
wsil:httpHeaders>
wsil:requestLine>POST /WSITest/servlet/rpcrouter HTTP/1.1</wsil:requestLine>
wsil:httpHeader value="volodin:9080" key="Host" />
wsil:contentTypeHeader type="application" subtype="soap+xml">
wsil:parameter value=""utf-8"" key="charset" />
 </wsil:contentTypeHeader>
wsil:httpHeader value="626" key="Content-Length" />
 </wsil:httpHeaders>
wsil:messageContents>
SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
SOAP-ENV:Body>
ns1:setEmail xmlns:ns1="http://tempuri.org/wstest.WSTestService">
email xmlns:ns2="http://wstest/" xsi:type="ns2:EmailType">
ns2:flag xsi:type="xsd:boolean">true</ns2:flag>
ns2:email xsi:type="xsd:string">email</ns2:email>
ns2:host xsi:type="xsd:string">host</ns2:host>
 </email>
 </ns1:setEmail>
 </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
 </wsil:messageContents>
 </wsil:message>
wsil:message type="response" id="2" conversation="1">
wsil:httpHeaders>
wsil:requestLine>HTTP/1.1 200 OK</wsil:requestLine>
wsil:httpHeader value="WebSphere Application Server/5.0" key="Server" />
wsil:httpHeader value="JSESSIONID=0000CCW5FBPKVDJZKV5BVFY1PQI:-1;Path=/"
 key="Set-Cookie" />
wsil:httpHeader value="no-cache="set-cookie,set-cookie2"" key="Cache-Control" />
wsil:httpHeader value="Thu, 01 Dec 1994 16:00:00 GMT" key="Expires" />
wsil:contentTypeHeader type="application" subtype="soap+xml">
wsil:parameter value=""utf-8"" key="charset" />
 </wsil:contentTypeHeader>
wsil:httpHeader value="446" key="Content-Length" />
wsil:httpHeader value="ru-RU" key="Content-Language" />
wsil:httpHeader value="close" key="Connection" />
 </wsil:httpHeaders>
wsil:messageContents>
SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
SOAP-ENV:Body>
ns1:setEmailResponse xmlns:ns1="http://tempuri.org/wstest.WSTestService" />
 </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
 </wsil:messageContents>
 </wsil:message>
 </wsil:messageLog>
wsil:descriptionFiles>
wsil:descriptionFile encoding="utf-8" filename="WSTestService.wsdl">
 <
here comes the WSDL file)

 </wsil:descriptionFile>
wsil:descriptionFile encoding="utf-8" filename="WSTestServiceBinding.wsdl">

ossibly another WSDL file)

wsil:descriptionFile>

 </wsil:descriptionFiles>
 </wsil:testLog>
```

Figure 3 – Example of Message Log file.

6.3 Appendix C: Q & A on What to Expect from the Testing Tools

Question: Can the testing tools guarantee that a Web Service is conforming to the Profile?
Answer: The tools can only verify the conformance of Web Service *artifacts* that are produced during a testing session. Some artifacts belong to the definition of the Web Service (WSDL); some others result from the observable behavior of the Web Service at run-time. It is rather difficult to test all possible behaviors that a Web Service can exhibit, mostly because exercising these behaviors is application-dependent and requires an application-level understanding of the Web Service. The testing tools are then an *indicator* of conformance of a Web Service to the Basic Profile, based on the artifacts produced. In turn, this is an indicator of interoperability with other business partners who also have tested as conforming to the Basic Profile.

Question: Can the testing tools verify all the requirements of the Basic Profile?
Answer: No. A few requirements of the WS-I basic profile cannot be easily tested, and have no corresponding test assertion. Such requirements fall into one of these categories:

- The profile requirement refers to an external specification document that is too complex to test, for an outcome that has been prioritized as low, given current resources. An example is the requirement on cookies which, when used, must conform to RFC2965.
- The requirement is not possible to test using the current test harness. For example, requirements about the HTTP code value when a request has been redirected.
- The requirement is about interpretative behavior of a Web Service consumer or provider, which exceeds the capability of the test harness, and would require more intrusive technology, or more knowledge of the WS application and semantics.

This is another reason why the tools should be defined as an indicator of conformance, rather than as certification tools. However the tools provide a powerful indicator of the ability of a Web Service to interoperate with any external party known to also comply with the Profile.

Question: How can we be sure that all the operations of a Web Service have been covered in the testing?
Answer: This depends on the Test Scenarios used for this service. A complete coverage of all the Web Service operations will rely on the client program involved in the testing of the WS, which is either ad-hoc, or is a real application in deployment over which the test operator does not have much control. Even so, such a driver may not be able to trigger an exhaustive set of behaviors, e.g. those inducing all kinds of faults.

Question: What are some practical situations where the testing tools show value to Web Services users or vendors?
Answer: An industry may define industry-specific Web Services – e.g. purchase order submission, request for product information - and specific usage scenarios. This industry may require that the Web Service, when used according to these expected interaction scenarios, exhibits a profile-conforming behavior, as verified by WS-I testing tools. In order to achieve this, this industry will likely define a specific test driver for its Web Services. By doing so, this industry has effectively defined an industry-specific test harness and certification criterion for interoperability, based on the Profile. If such a Web Service passes the tests, a vendor in this

industry can claim that it is interoperable with any user application, provided that the user also complies with the Profile, and exercises the expected usage scenarios.
Another scenario shows value for interoperability trouble shooting: a client application may fail to interoperate with a Web Service, although both claim to be conforming to the Basic Profile. Because the testing tools can monitor messages from both interacting parties, the tools can be used to diagnose a failure to interoperate, and to identify the cause: either the client application or the Web Service may exhibit non-conforming behavior during this particular interaction. This will help determine responsibilities.

Question: Will the WS-I Test Framework also support functional testing of a particular Web Service?
Answer: This is outside of the scope of conformance testing to the Basic Profile. Such testing would involve knowledge of the application semantics that is specific to each Web Service. The Monitor developed by the Test working group could however be reused – for example by the Sample Application working group – to provide the message capture necessary to such testing.

Acknowledgements