Introduction to Testing Webservices

Author: Vinod R Patil

Abstract

Internet revolutionized the way information/data is made available to general public or business partners. Web services complement this by allowing data exchange between applications in a platform independent manner.
A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.

The major challenges faced by the testers of webservices are the absence of a user interface, scalability and security considerations and the distributed nature of webservices. For tackling these challenges the webservice testing strategy must involve proof of concept testing, unit testing, basic webservice testing, testing the SOA, interoperability testing and load testing.

This white paper starts by describing the major challenges faced by the Web services testing community. It then discusses the strategy recommended to test the webservice, followed by a short discussion on Interoperability testing and Load testing. It also includes the use of JMeter (an automated load testing tool) to test webservices. And towards the end the paper highlights some of the major web services testing tools currently available.

Introduction to Webservices

Webservices is a technology that allows applications to communicate with each other in a platform independent manner. Primarily webservices target issues of data and application integration. They help us in exposing business processes as methods or functions, which in turn allow businesses to communicate at an application or process level with business partners.
XML based protocols are used to describe a webservice and standardized XML messages are used by the webservice for communication with other services or with the client.

- Webservices are described using Web Services Description Language (WSDL). The WSDL consists of the URL for the webservice, the methods that are accessible and the input parameter types and the return types of the webservice.
- **S**tandard **O**bject **A**ccess **P**rotocol (SOAP) is used as the messaging standard for communicating with the webservice. The message is wrapped in a SOAP Envelope, which can be delivered across network over most known transport protocols like HTTP, IIOP, and SMTP and so on.
- Webservices are published and located with the help of **U**niversal **D**escription, **D**iscovery and **I**ntegration (UDDI).

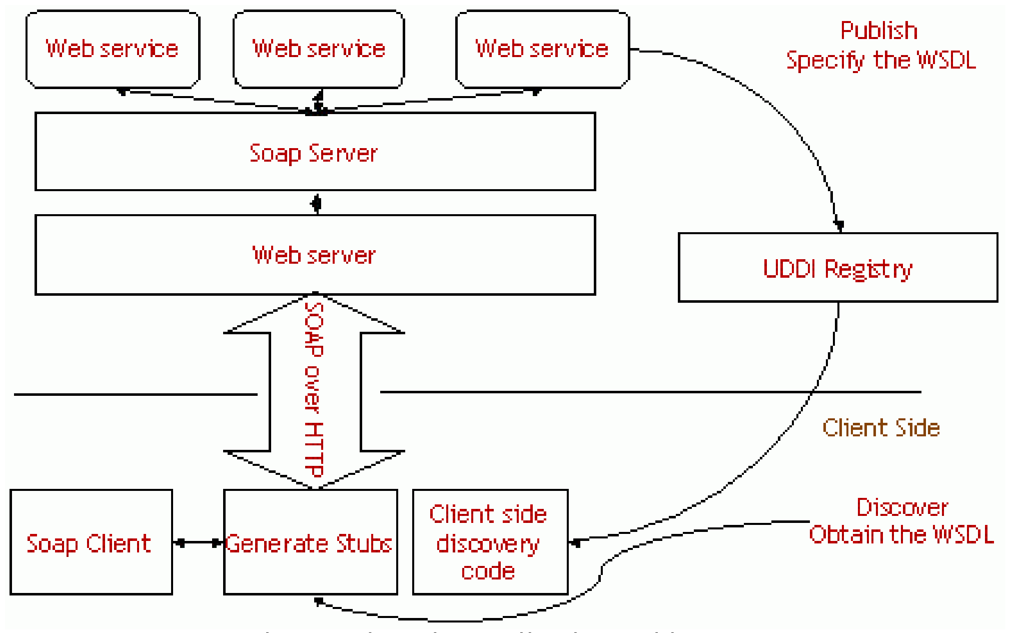


Fig1.1 Webservice application architecture

Challenges in testing webservices

The loosely coupled nature of webservices and non-existence of a User interface present a challenge to the developers and testers alike. Following are some of the challenges that webservice testers have to face.

Scalability and Security

The development and deployment environments of webservices are vastly different. If the webservice is for intranet usage then we have a theoretical maximum number of users that will connect to the service and also we have control over who can access the webservice so we have some security in place. But the scenario for Internet webservice is different. There we cannot make any assumptions about the number of users connected to the service, security or the way in which the users will access the webservice. Also we must know in advance the performance impact in case of large number of users connecting to the webservice.

Absence of User Interface

Unlike traditional web applications web services do not have a user interface. Hence they cannot be tested manually but require writing of test cases. For this the tester needs to have programming skills and an overview of the webservices fundamentals.

Distributed across network

Applications are generally built by integrating many webservices to leverage existing webservice functionality. These webservices may be developed by the same developers or may be provided by a third party. So thorough black box testing must be performed. Also these services are distributed over the network and may be hosted on different operating systems and deployed in different environments. Hence while testing we have to take into consideration the issues of availability, performance, reliability and security.

Testing the service

Types of testing

As with traditional applications, there are different sorts of testing that are needed to be carried out in case of webservices.

Proof of concept Testing

Webservice is a new concept and because of this we need to make sure that the architecture that we have chosen for our application is a correct one. There may be many options to choose from – for example which programming language we are using for developing the webservice, the database vendor that we will choose for the application etc. This type of testing mainly is conducted to gauge the correctness of the architecture. It basically helps us in knowing if our system is designed correctly.

Functional testing

Webservice is designed to solve a business problem. It has a predefined function to perform. This type of testing validates whether the service performs the intended function correctly, does it handle the exception conditions gracefully and does it handle the boundary value conditions.

Regression testing

Requirements may change as we start evaluating the software. Clients desire a change in existing functionality or addition of new functionality after seeing the system in action. Hence we need to change the existing system. In this change some functionality may be lost or altered. Regression testing aims to ensure that the webservice is still working across builds or releases. This sort of testing needs to be carried out during each release; hence it is an ideal candidate for automation. Test cases written in the unit-testing phase using JUnit can be used for regression testing. Once written the test cases can serve as a benchmark which any subsequent release must pass.

Load testing

Load or stress testing is a test of the performance of the webservice when many simultaneous users are accessing the system. The response of the webservice must be consistent and also its performance must not degrade with the increase in the number of users. Load testing gives us a feedback on these parameters and due to its very nature automated testing tools must be used for Load testing.

Webservice testing strategies

Unit testing

Webservice is similar to any other traditional application, so unit testing is a must. Unit test cases must be written before the application is developed. As and when the application is built, test cases are applied on the code. Hence the functionality is verified as and when we develop the webservice.

Basic testing

The main aim of this testing is to test whether the webservice is accessible and can be invoked properly. Main focus in this phase should be to carry out the following procedures.

- Get the WSDL file and test whether it is well-formed and in compliance with the WSDL specifications published by W3C
- Using this WSDL file generate the client side stubs that handle the interaction with the webservice.
- Test the webservice functionality that is whether the webservice responds to the requests submitted to it correctly. This involves coding a sample invoker to the client stubs.
- Invoke the sample invoker by passing it the parameters required by the webservice. Check the response of the webservice from a functionality point of view.
- The sample invoker calls the client stubs which further call the webservice. The stub constructs the SOAP message from the parameters passed to it and passes this message to the service. This message can be monitored by a Sniffer program like TCP Monitor.

- If there are any security checks, like username and password we need to test their effectiveness. The intent of this step should be to break in the system and gain unauthorized access.

Testing SOA

As organizations create a web service interface to their systems and overcome security issues, they will be able to exchange data with business entities such as customers, suppliers and partners in a more uninhibited and loosely coupled manner. Enterprises and established groups of business partners will find that UDDI-based service registries will become a critical enabler of the dynamic discovery of Web services within controlled environments.
For testing such collaborating webservices we need to focus on the following

- In a system where webservices interact with each other, we need to test the 'publish', 'find' and 'bind' capabilities of the constituent webservices.
- A particular SOAP message may typically have a designated recipient, but may also have one or more intermediaries along the message route that take actions based upon the instructions provided to them in the header of the SOAP message. Web services testing must verify the proper functionality of these intermediaries also.

Interoperability testing

In the loosely coupled environment of a service-oriented architecture, separate resources don't need to know the details of each others working, but they need to have enough common ground for reliably exchanging messages without error or misunderstanding. Standardized specifications help in creating such a common ground, but differences in implementation may still cause problems in the communication. Interoperability is when services can interact with each other without encountering such problems.

Interoperability issues

- **Absence of datatypes in request or response**
  SOAP is the standard for communication between webservices. SOAP requests are XML documents and XML provides flexibility regarding type casting the data being passed. The flexibility can be a problem for SOAP interoperability. For e.g. in the below message we do not have type information for the <result> element.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
     <ns1:echoStringResponse
xmlns:ns1="http://TestInterop.org/">
          <result>Hello, This is a string</result>
     </ns1:echoStringResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

  The interpretation of the data contained in the <result> element then becomes dependent on the underlying SOAP deserializer.

- **Interpretation of data types**

Webservices use SOAP serializer and deserializer to translate the SOAP message to the native language in which the webservice is implemented and here we encounter dependency on the native implementation for e.g. the way in which Date objects are defined in Java is different from .NET or C++ date objects. This leads to interoperability problems.

- **Handling of precision**
  BigDecimal data types are used to represent large numbers. There are differences in the maximum precision supported by the underlying programming platform. So it may so happen that the request or response of the webservice contains numbers having higher precision than what can be handled by the native language of the service or the client. The interpretation of these numbers then depends on how such cases are handled by the native language. This may have an impact in applications where precision is of importance like banking or financial domain applications.

Carrying out interoperability testing

To test webservices for interoperability we need to send “echo” invocations to the service in which a client sends a parameter of a certain type (such as integer, string, etc.) and the server simply returns a parameter of the same type and value. The client then examines the returned value to ensure that it matches the value it sent. The below figure illustrates the round trip for one such test.

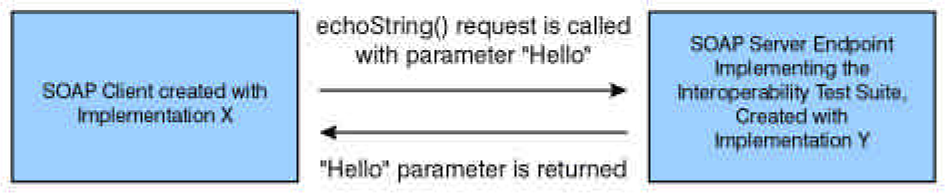

SOAP Client created with Implementation X
echoString() request is called with parameter "Hello"
"Hello" parameter is returned
SOAP Server Endpoint Implementing the Interoperability Test Suite, Created with Implementation Y

Fig 1.2echoString test case

By executing this round trip, the test exercises
1) The ability for the server to parse the client's SOAP envelope.
2) The ability for the server to deserialize the encoded parameter contained within the envelope.
3) The ability for the client to parse the SOAP envelope sent by the server in response.
4) The ability for the client to deserialize the encoded parameter sent back from the server.

Similarly we have test cases for other data types. Toolkits are available for interoperability testing. For e.g. the toolkit provided by SOAPBuilders (an online group created to address interoperability issues) can be found at http://www.xmethods.net/ilab.
The results of SOAPBuilders ILAB interoperability testing are published at http://www.xmethods.net/ilab/ilab.html#client.

Load testing

Load testing must be carried out in order to understand performance statistics of the webservice. Load testing gives us an idea of what the users will experience during their live interaction with the webservice when it is rolled out into production. Load testing entails hitting the webservice with many requests simultaneously and getting measures of times for various parameters like time to connect to webservice and time for the receipt of the response from the webservice. Also the correctness of the response in case of simultaneous requests needs to be tested. Automated tools can be employed for this phase. Let us see the testing of webservice with JMeter (one of the automated tool that's freely available).

Load testing webservice with JMeter

JMeter is a java-based tool to perform load, functional and behavior testing and measure performance. It is used to perform testing of both static and dynamic resources (files, Servlets, Java Objects, Webservices, Data Bases and Queries). It can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types. You can use it to make a graphical analysis of performance.

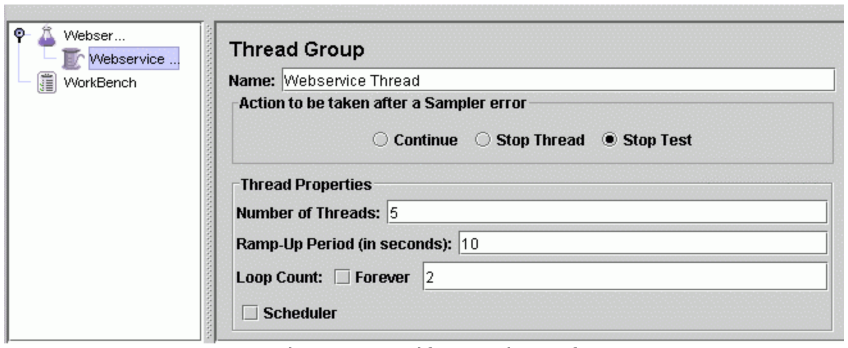
Getting started

To test a Webservice we need to create a Test Plan similar to traditional applications. The following steps need to be carried out.

- Specify the number of users
- Specify the webservice location and a SOAP request to it.
- Add a listener to listen to the results
- Run the test plan

Specify the number of users

Add a Thread group to the Test plan. On the Thread group we can specify the number of users we want to simulate and also the number of requests that each user must submit. The configuration is shown in fig below.



Thread Group
Name: Webservice Thread
Action to be taken after a Sampler error
Continue Stop Thread [✓ Stop Test]
Thread Properties
Number of Threads: 5
Ramp-Up Period (in seconds): 10
Loop Count: Forever 2
Scheduler

Fig 1.3 Specify number of users

For each user we must specify a thread of execution. We want 5 users so we must specify Number of threads as 5 here.

The requests from each user are repeated for the value specified in the Loop Count field. Here we have specified the value as 2 so each thread will repeat its request for 2 times. If we want the request to loop continuously then we have to check the "Forever" check box.

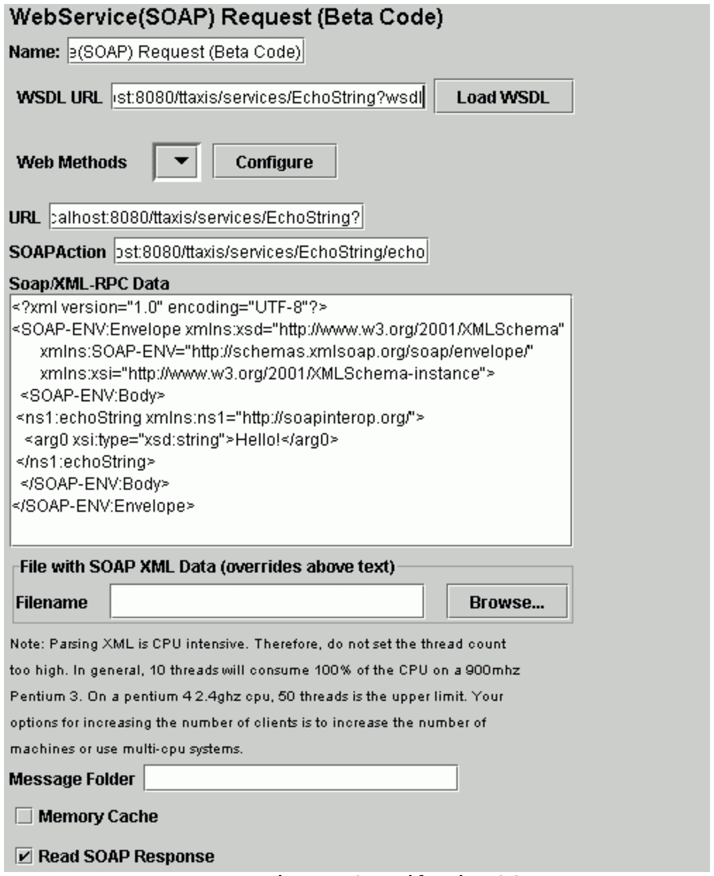
The Ramp up period tells JMeter how long to delay between starting each user. For example, if you enter a Ramp-Up Period of 10 seconds, JMeter will finish starting all of your users by the end of the 10 seconds. So the delay between the 5 users will be of 2 seconds. If you set the value to 0, then JMeter will immediately start all of your users.

Specify the webservice location and a SOAP request
The SOAP request that the users will submit must be specified with the help of "Webservice Request Sampler". This is available on the Thread Group when you right click it. Select the Add-->sampler-->Webservice (SOAP) Request.

Next we need to specify the WSDL file location in the "WSDL URL" text box and load the WSDL. If the WSDL is loaded correctly then the "Web methods" drop down will be populated with the list of methods in this webservice. Select a web method from the drop down and click the "Configure" button. This wi ll populate the URL and SOAP Action fields.

Lastly we need to specify the SOAP message that we want to send to the webservice. We can specify the message directly in the text area or specify a file to pick the SOAP request from.

The fully configured Webservice Sampler will look as shown in the figure below.



Name: e(SOAP) Request (Beta Code)

WSDL URL: st:8080/ttaxis/services/EchoString?wsdl

URL: :alhost:8080/ttaxis/services/EchoString?

SOAPAction: ost:8080/ttaxis/services/EchoString/echo

Soap/XML-RPC Data

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAP-ENV:Body>
<ns1:echoString xmlns:ns1="http://soapinterop.org/">
  <arg0 xsi:type="xsd:string">Hello!</arg0>
</ns1:echoString>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

File with SOAP XML Data (overrides above text)

Filename

Note: Parsing XML is CPU intensive. Therefore, do not set the thread count too high. In general, 10 threads will consume 100% of the CPU on a 900mhz Pentium 3. On a pentium 4 2.4ghz cpu, 50 threads is the upper limit. Your options for increasing the number of clients is to increase the number of machines or use multi-cpu systems.

Message Folder

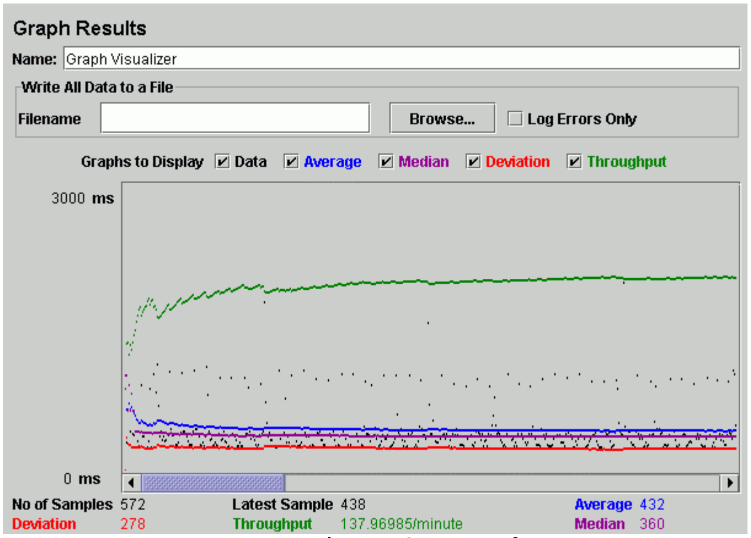
☐ Memory Cache

☑ Read SOAP Response

Fig 1.4 Specify the SOAP Message

Add a listener to listen to the test results

The final element you need to add to your test plan is a Listener. This element is responsible for storing all of the results of your requests in a file and presenting a visual model of the data.
Select the Thread Group element and add a Graph Results listener (Add --> Listener --> Graph Results). Next, you need to specify a directory and filename of the output file.



Name: Graph Visualizer
Filename:
Graphs to Display: ✓ Data ✓ Average ✓ Median ✓ Deviation ✓ Throughput

No of Samples 572 Latest Sample 438 Average 432
Deviation 278 Throughput 137.96985/minute Median 360

Fig 1.5 Output of JMeter

Running the test plan

Save the test plan and from the run menu, select Run. JMeter lights up a green square in the upper-right-hand corner to indicate if a test is currently running. The square is turned gray when all tests stop. The results will be shown in the graphical format as shown above or if you specify a file to save the results then the results will be saved to the file and can be referred later.

Webservices Testing tools

Many industry experts expect testing tools to play a big part in the success of Web services implementations. Following is the list of some of the Web Services testing vendors and their tools.

Vendor	Product	Description
Red Gate	Advanced .NET Testing System (ANTS)	This tool predicts the Web service behavior and performance under stress of multiple user requests. It simulates multiple clients accessing a Web application at the same time. ANTS is the first product designed to test .NET XML Web services.
Parasoft	SOAPtest	This tool measures the three main areas functionality, load, and regression. SOAPtest can evaluate both the performance of SOAP transactions at the server level and the user experience at the client level. It compares the actual responses from a Web service to the desired responses and also tests the internal construction of the components that provide the Web service.
Segue	SilkPerformer	It is a load testing tool which can test Java or .Net Web services. SilkPerformer allows you to simulate thousands of users so that we can predict the behavior of our deployed webservice.
Altova	Xmlspy 5	XMLSPY 5 includes full SOAP capabilities that include interpretation of WSDL, creation of SOAP requests, submitting them to the Webservice and viewing the SOAP Response
Apache	JMeter	General-purpose load drivers that you can use to develop web service clients for performance tests and display the collected data in a graphical format.

Conclusion

The loosely coupled nature and absence of UI in webservices pose a number of unique challenges during testing. We need to plan for testing right from the design stage beginning with a proof of concept testing. Unit test cases help us layout a plan for functional testing before we code the service. Once the webservice is developed a basic test needs to be performed to test access to it. If the application involves interaction of services with each other, then the test strategy should also include testing the application from a Service Oriented Architecture perspective. Interoperability testing ensures that the service will integrate seamlessly with varied client environments. To know the performance and scalability of the webservice we need to perform full-scale load testing. And finally as automated tools are conducive and best suited for testing webservices, they should be used to perform the testing of webservices.

About the author

Vinod Patil is a Software engineer with Patni computer systems ltd, India. He is a Mumbai university first class graduate and holds a B.E Computer engineering degree from Veermata Jijabai Technological Institute (V.J.T.I). He has been involved in developing applications in the J2EE environment and software development in object oriented environment in Patni for the last two years. Most recently he has been involved in designing and implementing a Webservice interface for data of a project being executed in Patni. His areas of interest include tracking the latest trends in Web service, designing and developing distributed and scalable J2EE applications and exploring the utility of new tools for developing and testing J2EE applications.