

An Efficient Formal Testing Approach for Web Service with TTCN-3

Pulei Xiong, Robert L. Probert, Bernard Stepien

University of Ottawa

(xiong | bob | bernard)@site.uottawa.ca

Abstract: Web service is considered as a prevailing integration technology on Internet/Intranet due to its language and platform-independence. The language and platform-independence characteristics bring difficulties on testing. In this paper, we propose a distributed testing process based on TTCN-3 which intends to solve the difficulties. The approach distributes test activities on both server and client sides. It ensures a systematic testing process. In addition, by specifying test suites at an abstract level in TTCN-3 the test suites are language and platform-independent, and can be reused by diverse types of clients. We demonstrate it via a case study.

1. INTRODUCTION

A web service (WS) is a kind of Internet application framework based on SOAP and XML technology. Two significant attributes of web services are standards-based architecture and XML-based messaging. Web service relies on a family of protocols to describe, deliver, and interact with each other, such as the Web Service Description Language (WSDL), the Universal Description, Discovery and Integration (UDDI) protocol or the Web Service Inspection Language (WSIL), and the Simple Object Access Protocol (SOAP). WSDL, UDDI, WSIL, and SOAP are all based on XML.

Due to the natures of standards-based architecture and XML-based messaging, web service is vendor-neutral, language-agnostic, and platform-independent. Therefore, developers of a web service are not able to assume which type of clients will consume the web service, and vice versa, developers at the client sides are not aware of which language and platform are used at the server side. A web service and its clients can be developed in totally different programming languages including Java, C#, VB, Perl or Python on different platforms.

2. CURRENT ISSUES AND PRACTICES IN WEB SERVICE TESTING

Testing web services is different from testing traditional software. From the client sides perspective, a web service client is not an actual final user; instead, it is a component of

another software system. To ensure the quality of the system, the developers at the client side need to perform testing on the web service client, just as what they will do on other components in the system. However, the information available about the web service to the developers is quite limited since the web service is usually out of the boundary of the system, even if both the web service and the client are developed in the same organization. Most of the time the only information available to the developers is a service interface in the form of a WSDL file. The service interface is insufficient for functional testing. For example, it does not tell which is the expected response against a set of input data, and it does not tell if there are any boundary value limitations imposed by business rules. It even does not provide any useful clues for non-functional testing such as performance and stress testing. Therefore, it is quite difficult, if it is not impossible, to conduct systematic testing on web service at client sides, e.g. with respect to coverage analysis. On the other hand, from the aspect of the server side, systematic test case analysis and design can be achieved since the developers at server side have comprehensive knowledge of the web service. However, at the server side it is quite difficult to conduct client-oriented tests due to diverse possibilities of languages, platforms and applications at client sides.

Obviously, either server side or client sides cannot take full responsibility of testing a web service. In a complete test process, client sides are weak at systematic test case design, while the server side is weak at test implementation and execution. To ensure systematic test case design and to ease test implementation and execution, it is appropriate to distribute the responsibility to both server and client sides: test case design and specification are conducted at server side and then the test specification is published to all client sides, and test implementation and execution are performed at client sides based on the test specification. Furthermore, at server side tests must be specified at an abstract level --- Abstract Test Suite (ATS), which means they should be language and platform-independent, since only an ATS can accommodate all varieties of client types.

There are both research and tools from industry to facilitate web service testing. In [1], the authors introduced a Java and XML based test harness for web services. In [2] and [3], the authors presented testing approaches based on tools/framework on .NET platform. In [4], the author presented how to use WebSphere WSDK v5.1 to generate a JSP page

for client sides from a WSDL file to test the web service. WSDL2Java, a java tool in AXIS [5], can generate web service proxy from a WSDL file that makes it easier to develop client side applications. However, these tools and approaches do not guarantee any systematic test. In addition, they are language-specific, either for Java or C#, which makes no sense to those client sides where applications are programmed in other languages. Unfortunately, web service intends to be consumed by diverse languages.

In short, due to the natures of language and platform-independency, web service testing is different from traditional software testing. Current testing process and tools do not work well on web service. In the next section, we will discuss a formal testing approach based on TTCN-3, which is deemed to solve the above issues.

3. A FORMAL TESTING APPROACH WITH TTCN-3

3.1 TTCN-3 and Internet/Web Application Testing

TTCN-3 is an international standard test specification and implementation language. It has been developed by ITU and ETSI (European Tele-communication Standards Institute). TTCN-3 intends to support black box testing for reactive and distributed systems. Typical areas of application for TTCN-3 are protocols, services, APIs, and software modules [6]. TTCN-3 specifies test suites at an abstract level [7]. In [8][9][10], we discussed how to apply TTCN-3 to web application and web content testing. In [11], the authors presented how to generate TTCN-3 scripts from XML DTD or schema to automate web service testing. In this paper, we illustrate how to build a systematic testing process for web service that distributes test activities to both server and client sides, how to facilitate the process by specifying ATS in TTCN-3, and how to implement ATS at client sides by utilizing third-party TTCN-3 and web service tools.

3.2 Distributed Testing Process

In this distributed testing process, both server side and client sides should be involved in testing activities (see figure 1). Test case analysis and design that is based on models and/or source code of WSSs, and ATS specification that is based on WSDL files and test cases, are conducted at server side. The ATS will be published via Internet/Intranet, and then it will be retrieved at client sides. The ATS compiling and implementation by developing Test Adapter (TA) and Encoder/Decoder (CoDec) in a native language are performed at client sides. Finally, the test is executed at client sides.

The testing process brings the following outstanding advantages. First, test case design can be conducted systematically by applying proper methods at server side. The

quality of the test cases should be much better than those developed ad hoc at client sides. Second, comparing to developing executable tests, specifying ATS is simpler since all language and platform-specific details are exempt from consideration. Testers can focus on specifying test logics such as test steps and test result verification, without considering how to implement them. Third, test case design, ATS specification and maintenance are all done at server side and shared at client sides. Obviously, this is much more efficient than developing tests at every client side individually. Fourth, accordingly, testers at client sides only need to compile ATS to a native language and develop TA and CoDec for the ATS since all server side-specific test information is included in the ATS.

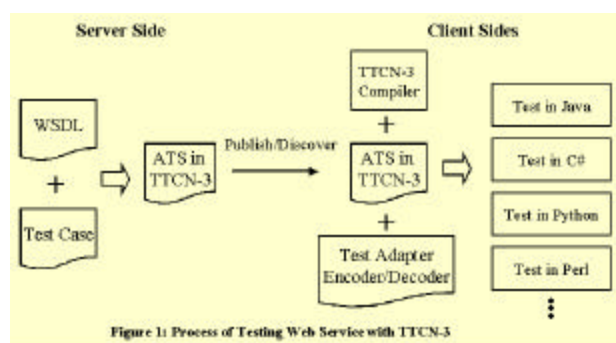


Figure 1: Process of Testing Web Service with TTCN-3

4. CASE STUDY

4.1 Introduction

We conducted two functional tests and two stress tests on the web service “getQuote”. “getQuote” is a sample web service deployed on AXIS, which is a web service engine built in Java. The test process includes four steps: (1) Test Case design, (2) ATS specification, (3) Test implementation, and (4) Test execution. The first two steps are assumed to be accomplished at server side, while the last two steps assume to be accomplished at client sides.

4.2 Test Case Design

We can conduct test case analysis and design based on models and source code that are available at the server side, which is just the same as we do on a regular component. For example, we can derive test scenarios from use cases, and then create concrete test cases from the test scenarios. We can do test coverage analysis against the test cases. We can assign priorities to the test cases based on risk and yield analysis. The web service “getQuote” accepts parameters including user ID, password and stock symbol. If the user and password are valid and the stock symbol is correct, it returns

the quote of the stock (normal scenario). If the user and password are valid but the stock symbol does not exist, it returns -1.0 (alternative scenario). We did two functional tests on the web service: one is for testing the normal scenario (#1 in table 1); another one is for the alternative scenario. In addition, we did two stress tests: we simulate invoking “getQuote” concurrently with 50 and 100 clients (#2 in table 1) to see if the server could accommodate the request flux.

Table 1 Test Case Specification

Test Case	Input	Expected Output
#1	User ID = user1, Password = pass1, Symbol = XXX	55.25
#2	User ID = user1, Password = pass1, Symbol = XXX Number of concurrent clients = 100	More than 95% requests success

4.3 ATS Specification

This step is also conducted at server side. At this step, we do not need to consider how these test suites are implemented at client sides. No client side environment information is required. ATS is built only based on test cases and the WSDL file for a web service.

One property of ATS is language and platform-independency. Therefore, when we specify ATS in TTCN-3, we should be very careful not having any language or platform-specific information in it. On the other hand, however, we have to include all test logic information that is sufficient to implement the ATS at client sides, including data types and concrete data related to test input and expected result, test steps, test structure, etc.

We build an ATS for “getQuote” based on the test case specification in Table 1 and the WSDL file for the web service. A WSDL file is a XML document that describes the interface to a web service in a language and protocol independent way. Two elements in the WSDL are used to invoke the web service: the URL of the web service and the Qualified Name (QName) of the web service.

4.3.1 ATS for Functional Tests

The steps to specify ATS in TTCN-3 are as follows. A typical program style can be found in [12].

First, a record type is defined to specify URL, QName and input data:

```
type record wsRequestType {
    charstring protocol,
    charstring host,
    charstring portNum,
    charstring path,
    charstring nameSpace,
    charstring opName,
```

```
charstring user,
charstring passwd,
charstring symbol } }
```

Second, using the previously defined data type, we can define test data templates that specify test input (request) and expected result (response):

```
template wsRequestType wsRequest - Valid := {
    protocol := "http://",
    host := "localhost:",
    portNum := "8080",
    path := "/axis/services",
    nameSpace:= "urn:xmltoday-delayed-quotes",
    opName := "getQuote",
    user := "user1",
    passwd := "pass1",
    symbol := "XXX" }
```

```
template charstring wsResponseValid := "55.25";
```

Third, a test configuration is defined that specifies communication ports and components to construct a test structure:

```
type port wsPortType message {
    out wsRequestType;
    in charstring; }
type component ptcType {
    port wsPortType wsPort;
    timer localTimer := 3.0; }
type component systemType {
    port wsPortType wsPort; }
type component mtcType { }
```

Fourth, a function is created to specify the behavior of the ptcType test component as a sequence of sending a WS request followed by an alternative of receiving a normal response and various abnormal responses :

```
function ptcBehaviour(in wsRequestType wsRequest, charstring
wsResponse) runs on ptcType {
    wsPort.send(wsRequest);
    localTimer.start;
    alt
    {
        [] wsPort.receive(wsResponse) {
            localTimer.stop;
            setverdict(pass); }
        [] wsPort.receive {
            localTimer.stop;
            setverdict(fail); }
        [] localTimer.timeout {
            setverdict(fail); } }
    }
```

Fifth, a test case is created to test the normal scenario for a single test using the previously defined function:

```

testcase GetQuoteTest_Valid() runs on mtcType system
systemType
{
    var ptcType ptc;
    ptc := ptcType.create;
    map (ptc:wsPort, system:wsPort);
    ptc.start(ptcBehaviour(wsRequestValid,
                        wsResponseValid));
    ptc.done; }

```

Finally, execute the test case at the control section:
`execute(GetQuoteTest_Valid());`

4.3.2 ATS for Stress Tests

Only minor modifications on the ATS for the functional tests are necessary to make it usable for stress testing. All that is needed is to make an array of parallel test components.

```

type component systemType {
    port wsPortType wsPort[NUMBER_OF_PTCS]; }

```

Each parallel test component, referred by its index, is created and executed independently by the TTCN-3 test platform:

```

for (i:=0; i<NUMBER_OF_PTCS; i:=i+1) {
    ptc[i] := ptcType.create; }
for (i:=0; i<NUMBER_OF_PTCS; i:=i+1) {
    map (ptc[i]:wsPort, system:wsPort[i]); }
for (i:=0; i<NUMBER_OF_PTCS; i:=i+1) {
    ptc[i].start(ptcBehaviour(wsRequestValid,
                            wsResponseValid)); }
for (i:=0; i<NUMBER_OF_PTCS; i:=i+1) {
    ptc[i].done; }

```

4.4 Test Implementation

Test implementation is performed at client sides based on ATS. Since ATS provides all necessary test logic information at server side, testers at client sides can focus on ATS implementation. They will not feel frustrated about test case design due to lack of knowledge on web services.

To get Test Executable (TE) from ATS, a TTCN-3 compiler is required to translate ATS to program in a native language at client sides. In our case, we used TTthree, a TTCN-3 to Java compiler developed by Testing Technologies IST GmbH, to translate ATS to Java code [12].

Except for generating TE from ATS, we need to implement a TA and CoDec, which are standard entities in the test architecture defined in [13], to make TE run on a specific platform with System Under Test (SUT). A TA deals with any aspects that cannot be concluded from information being present in ATS, such as test system user interface, test execution control, test event logging, communication with SUT, and timer implementation [12]. Encoder translates TTCN-

3 data to real world data, while Decoder translates real world data to TTCN-3 data.

TTthree runtime environment provides default implementation for TA and CoDec. In our case, two important methods need to be overwritten: `triSend` in TA and `encode` in CoDec.

`triSend` constructs a SOAP message and sends it out to request for “getQuote” web service, then listens to the returned SOAP message, and parses the message. We can code `triSend` using standard Java API, or using any third-party web service tools for clients. In our case, we used AXIS client API that makes programming much simpler: we do not need to know any details about WSDL file and SOAP message, and we do not need to construct and parse SOAP message. Sending out requests and receiving responses are all done by calling several AXIS APIs as shown below:

```

//set up connection
Service service = new Service();
Call call = (Call) service.createCall();
call.setTargetEndpointAddress(new URL(url));
call.setOperationName(new QName(nameSpace,opName));
call.addParameter("symbol",
                  XMLType.XSD_STRING,ParameterMode.IN);
call.setReturnType(XMLType.XSD_FLOAT);
call.setUsername(user);
call.setPassword(passwd);
//invoke web service
Object ret = call.invoke(new Object[] {symbol});

```

It is worth to mention that the functional tests and the stress tests use the same TA and CoDec. This shows how ATS increases the reusability of the test implementations.

4.5 Test Execution

Figure 2 is the screenshot of test execution result for test case #1. In this particular case it passed.

Figure 3 is the screenshot for test case #2. From it we can see 17 out of 100 clients failed to access “getQuote” successfully due to connection time-out. The test case is set to fail. This means the server side cannot accommodate 100 concurrent clients. Although from the point of view of client sides, it is unnecessary for us to analyze why the server cannot support more than 100 concurrent clients and how to improve it, the test result does help us to consider how to improve the system at client sides: e.g. limit the number of concurrent requests to avoid connection time-out; and once there are failed connections, set up reconnections in a reasonable time frame. This may bring a friendly interface to users, rather than showing them failed connection information.

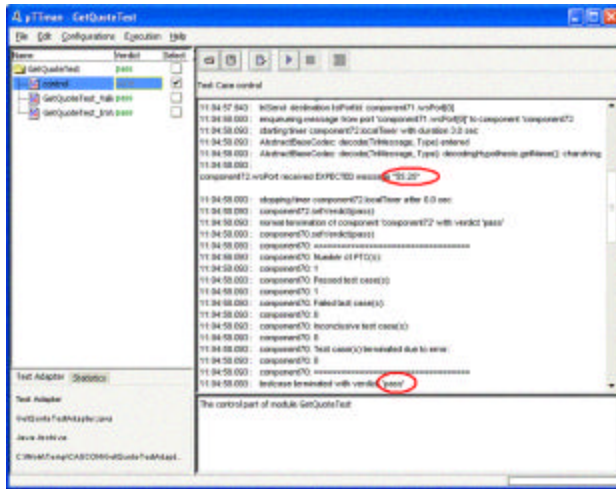


Figure 2: Screenshot for Test Case #1

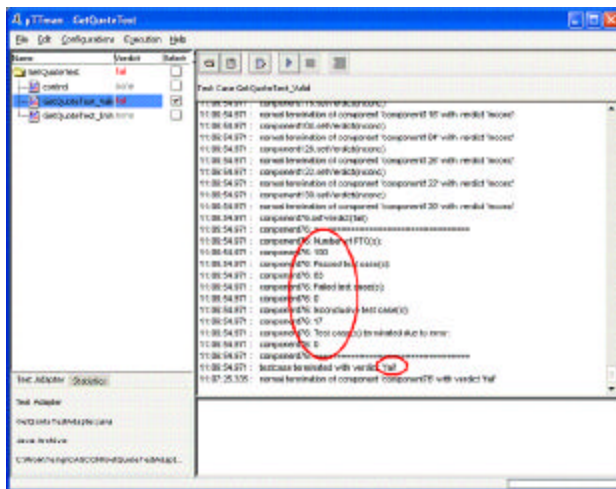


Figure 3: Screenshot for Test Case #2

5. CONCLUSION

In this paper, we proposed a formal testing approach for web services with TTCN-3, which distributes the test activities to both server and client sides. This facilitates the web service testing while the traditional testing approaches failed due to the difficulties on testing brought by the language and platform-independence characteristics of web services. It ensures systematic testing and increases the reusability of the test implementations. In the future, we may consider developing a prototype tool to automatically generate an ATS in TTCN-3 from WSDL files and test case specifications. Besides, it is also worthwhile to discuss how to extend web service publish/discover protocols such as UDDI or WSIL to make possible the delivery of an ATS with a WSDL file.

ACKNOWLEDGEMENTS

The authors would like to thank Testing Technologies IST GmbH for providing us the necessary tool — **TTthree** to carry out this research. As well, we are grateful to NSERC and OCE for supporting this research.

REFERENCES

- [1] Rajesh Sumra, R. Venkatvaradan, “Web Service's Test Harness: A Functional, Load, and Performance Testing Framework for Web Services”, June 2005, <http://www.developer.com/services/article.php/2229161>,
- [2] Roger Jennings, “Test Web Services Without Writing a Client”, http://www.fawcette.com/xmlmag/2002_05/online/webservices_rjennings_05_03_02/default.aspx, June 2005
- [3] James McCaffrey, “Automate Your ASP.NET Web Services Testing”, <http://msdn.microsoft.com/msdnmag/issues/05/03/TestRun/default.aspx>, June 2005
- [4] Ron Ben-Natan, “Building Web Services, Part 1: Build and Test”, IBM WebSphere Tutorials, Jan 2004
- [5] AXIS project, “AXIS User's Manual v1.2”, <http://ws.apache.org/axis/java/userguide.html>, June 2005
- [6] TTCN-3 Home Page, <http://www.etsi.org/ptcc/ptcttcn3.htm>, June 2005
- [7] ETSI ES 201 873-1, “The Testing and Test Control Notation version 3, Part1: TTCN-3 Core Language, V2.2.1”, Feb. 2003
- [8] Pulei Xiong, Robert L. Probert, “A Multi-Method Testing Approach in TTCN-3 for Web Applications”, Proc. of the 14th IEEE Intel. Symposium on Software Reliability Engineering (ISSRE), pp. 261-262, Nov. 2003
- [9] Robert L. Probert, Pulei Xiong, Bernard Stepien, “Life-cycle E-Commerce Testing with OO-TTCN-3”, Proc. of 1st Intel. Workshop on Theory Building and Formal Methods in Electronic/Mobile Commerce (TheFormEMC), pp. 16-29, Oct. 2004
- [10] Robert L. Probert, Bernard Stepien, Pulei Xiong, “Formal Testing of Web Content using TTCN-3”, TTCN-3 User Conference 2005, June 2005
- [11] B.Stepien, I.Schieferdecker, “Automated Testing of XML/SOAP based Web Services”, Proc. of the 13th. Fachkonferenz der Gesellschaft für Informatik (GI) Fachgruppe KiVS, Feb. 2003
- [12] Testing Technologies IST GmbH, “Using TTthree: Users Manual and Programming Guide”, August 2004
- [13] ETSI ES 201 873-5, “The Testing and Test Control Notation version 3, Part5: TTCN-3 Runtime Interface (TRI), V1.1.1”, February 2003