

Coyote: An XML-Based Framework for Web Services Testing

W. T. Tsai*, Ray Paul†, Weiwei Song*, Zhibin Cao*

*Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287

†Department of Defense, Washington DC

wtsai@asu.edu

Abstract

Web services received significant attention recently and several important web service platforms such as .NET are now available. The testing and evaluation of web services are important for both service providers and subscribers. This paper proposes an XML-based object-oriented testing framework Coyote to test web services rapidly. Coyote consists of two parts: test master and test engine. The test master allows testers to specify test scenarios and cases as well as performing various analyses such as dependency analysis, completeness and consistency, and converts WSDL specifications into test scenarios. The test engine interacts with the web services under test, and provides tracing information. The test framework incorporates key concepts from object-oriented application frameworks so that it is relatively easy to change test scenarios/cases. Due to the distributed nature of web services, Coyote focuses on integration testing instead of module testing.

Keywords: web services, object-oriented application frameworks, scenario analysis, WSDL, integration testing.

1. Introduction

Web services received significant attention recently and several important web service platforms such as .NET are now available [Troelsen 2001]. The testing and evaluation of web services are important for both service providers and subscribers because testing web services is difficult and time-consuming. Testing web services is difficult because web services are distributed applications with numerous runtime behaviors and involves multiple standard protocols such as UDDI and SOAP. Furthermore, for a service subscriber, often only the WSDL specifications are available, thus only black-box testing is feasible because the design and implementation details of participating web services are not available. Finally, in a large application, multiple web services will be involved, selected, and executed at runtime, and these features makes web service testing challenging.

This paper proposes an XML-based object-oriented (OO) testing framework Coyote to test web services rapidly. Coyote consists of two parts: test master and test engine. The test master allows testers to specify test scenarios [Tsai 2002a] and cases as well as various analyses such as dependency analysis, completeness and consistency [Tsai 2001a], and converts WSDL specifications into test scenarios. The test engine interacts with the web services under test, and provides tracing information. The test framework incorporates key concepts such as design-for-change and design patterns from OO application frameworks [Fayad 1999] so that it is relatively easy to change test scenarios/cases. Recent test framework such as Cactus [Cactus] often includes support for test execution in addition to test scenario/case organization. Coyote is test framework that support both test execution and test scenario management. Due to the distributed nature of web services, Coyote focuses on integration testing instead of on module testing.

2. An OO Framework to Test Web Services

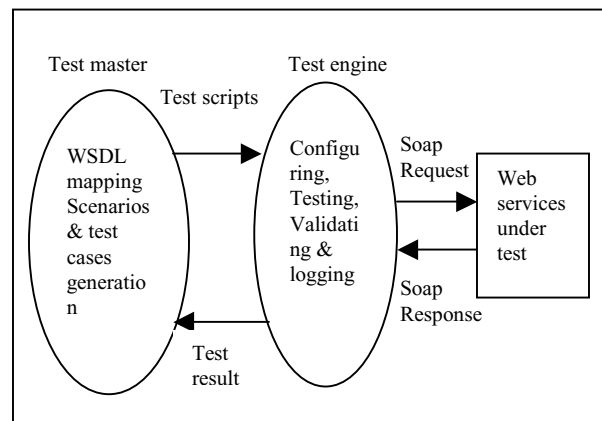


Figure 1 The Structure of Coyote

Figure 1 shows the overall structure of Coyote. The test master maps WSDL specifications into test scenarios, performs test scenarios and cases generation, performs dependency analysis, and completeness and consistency checking. A WSDL file contains the signatures

specification of all the web services methods including method names, and input/output parameters [W3C], and the WSDL can be extended so that a variety of test techniques can be used to generate test cases [Tsai 2002c]. The test master extracts the interface information from the WSDL file and maps the signatures of web services into test scenarios. The test cases are generated from the test scenarios in the XML format which is interpreted by test engine in the second stage. For example, if a bank web services has two methods *checkBalance* and *deposit* and their signatures are as :

Double checkBalance (string account);

Boolean deposit(string account, double amount);

And the user scenarios is a customer deposit a certain amount of money and then check the balance. The test scripts will be:

```
<scenario name="Check Balance">
  <scenario_configuration>
    <URN>BankServices</URN><SOAProuterURL>
      http://localhost:8080/soap/servlet/rpcrouter
    </SOAProuterURL> </scenario_configuration>
    <scenario_path>
      <method sequence="1">
        <name>deposit</name>
        <data direction="input">
          <dataname>account</dataname>
          <type>string</type></data> ...
        <data direction="output"><dataname/>
          <type>Boolean</type></data></method>
      <method sequence="2">
        <name>checkBalance</name>
      ... </method>
    </scenario_path> <scenario_testcase> <testcase>
      <testdata sequence="1">foo</testdata>
      <testdata sequence="1">100.00</testdata>
      <testdata sequence="1">>true</testdata> ...
    </testcase> </scenario_testcase> </scenario>
```

Testing engine reads the test scripts produced by test master and executes the test at the target web services, it also logs the execution trace and sends the test results back to the test master. In other words, the test engine acts like design patterns Mediator and Proxy.

The actual test execution involves three phases:

- **Configuration:** This sets up the connection to the target web service, configure the testing data with the information specified in tag `<scenario_configuration>`.
- **Test:** this generates the SOAP request messages, invokes the particular service method with input parameters specified in tag `<testcases>` and `<scenario_path>`.

- **Validating & logging:** This checks and assesses the testing result in the SOAP response message against the expected output specified in the test scripts, and logs assessing results.

The testing framework can be configured in two ways: a single test engine versus multi web services or multi decentralized test engines versus multi web services supporting the distributed testing, where different testing engines can collaborate to perform a single test task and share information during the testing process. Several synchronization schemes are available to coordinate the test engines in a distributed manner.

Coyote also provides support for the regression testing. Once the test scripts have been generated, the regression test can be easily performed at anytime. The retest-all and selective-test strategies are incorporated in the framework.

Reference:

[Cactus] Cactus project,

<http://jakarta.apache.org/cactus/index.html>.

[W3C] W3C, Web Services Description Language

<http://www.w3.org/TR/wsdl>

[Fayad 1999] M. Fayad, D. C. Schmidt, and R. E. Johnson, *Building Application Frameworks*, Wiley, New York, NY, 1999.

[Troelsen 2001] A. Troelsen, *C# and the .NET Platform*, Addison Wesley, Reading, MA, 2001.

[Tsai 2001] W.T. Tsai, X. Bai, R. Paul, W. Shao, and V. Agarwal, "End-to-End Integration Testing Design", Proc. of IEEE COMPSAC, 2001, pp. 166-171.

[Tsai 2002a] W. T. Tsai, X. Bai, R. Paul, K. Feng, and L. Yu, "Scenario-Based Modeling and Its Applications to Object-Oriented Analysis, Design, and Testing", Proc. of IEEE WORDS 2002.

[Tsai 2002b] W. T. Tsai, L. Yu, R. Paul, T. Liu, and A. Saimi, "Developing Adaptive Test Frameworks for Testing State-Based Embedded Systems", in Proc. of IDPT, 2002.

[Tsai 2002c] W. T. Tsai, R. Paul, Y. Wang, C. Fan, and D. Wang, "Extending WSDL to Facilitate Web Services Testing", Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287, 2002.