WebSob: A Tool for Robustness Testing of Web Services

Evan Martin Suranjana Basu Tao Xie
North Carolina State University, Raleigh, NC, USA
{eemartin, sbasu2}@ncsu.edu, xie@csc.ncsu.edu

Abstract

Web services are a popular way of implementing a Service-Oriented Architecture. Testing can be used to help assure both the correctness and robustness of a web service. Because manual testing is tedious, tools are needed to automate test generation and execution for web services. This paper presents WebSob, a tool for automatically generating and executing web-service requests given a service provider's Web Service Description Language (WSDL) specification. We have applied WebSob to freely available web services and our experiences show that WebSob can be used to quickly generate and execute web-service requests that may reveal robustness problems with no knowledge of the underlying web service implementation.

1. The WebSob Tool

We have developed the WebSob tool for robustness testing of web services, as illustrated in Figure 1 [4]. Given a WSDL from a service provider, WebSob first generates code to facilitate both test generation and test execution. WebSob then generates a test suite. WebSob runs the generated test suite on the generated client code, which eventually invokes the web service. WebSob then collects the results returned from the web service. In particular, our WebSob tool consists of Code Generation, Test Generation, Test Execution, and Response Analysis.

1.1. Code Generation

WebSob generates the necessary code required to implement a service consumer. In addition, WebSob generates a test class that can execute each service independently. Axis [1] provides a Java implementation of the SOAP protocol. We use Axis to generate client-side code from a service provider's WSDL. WSDL is an XML-based language that describes the public interface of a service. It defines the protocol bindings, message formats, and supported operations that are required to interact with the web services

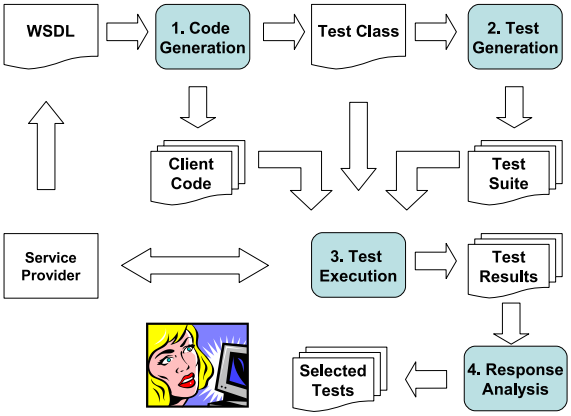

Figure 1. Overview of the Framework

listed in its directory. The Axis utility class, `WSDL2Java`, parses the WSDL and generates the necessary WSDL files that facilitate the implementation of service consumers. A Java class is generated to encapsulate each supported message format for both the input and output parameters to and from the service. A Java interface is generated to represent each port type. A stub class is generated for each binding. A service interface and corresponding implementation is generated for each service. In addition, we generate a wrapper class that leverages the code generated by `WSDL2Java` to invoke the service provider. This wrapper class is designed to allow unit-test generation tools to automatically generate unit tests that exercise the various services offered by the service provider.

1.2. Test Generation

Given the generated wrapper class, WebSob uses a unit-test generation tool to generate a test suite that exercises the services defined in the WSDL. WebSob operates relatively independently of the test generation tool and thus other unit test generation tools (such as Agitar Agitator[1] and Parasoft Jtest[2]) may also be used. Our results are obtained

[1] `www.agitator.com`
[2] `www.parasoft.com`

via JCrasher [2], a third-party test generation tool that automatically generates JUnit [3] tests for a given Java class. For example, JCrasher generates -1, 0, and 1 for arguments with the integer type and it can generate method sequences that create values for those arguments with non-primitive types. We have modified JCrasher to generate additional values for numeric arguments such as the maximum and minimum values supported by that type. JCrasher is designed as a robustness testing tool by causing the program under test to throw an undeclared runtime exception. More specifically, JCrasher examines the type information of a set of Java classes and constructs code fragments that create instances of different types to test the behavior of public methods. These code fragments are used in the generated unit tests to supply inputs to the public methods under test. In our case, the public methods under test are in the wrapper class. Each method there corresponds to a service defined in the WSDL and each method argument corresponds to an input parameter for that service. JCrasher generates unit tests that instantiate the necessary input parameters to invoke the web service.

1.3. Test Execution

Given the generated wrapper class, unit test suite, and client-side implementation, we use JUnit [3] to execute the unit tests against the wrapper class, which invokes the remote web service. JUnit [3] is a regression testing framework that is used to execute a unit-test suite against the class under test. The test class throws an exception if a SOAP failure is encountered.

1.4. Response Analysis

The responses from the web service are classified and analyzed. WebSob selects tests that cause the web service to return robustness-problem-exposing responses. Manual inspection and heuristics may be used to determine whether an exception should be considered to be caused by a bug in the web service implementation or the supplied inputs' violation of the service provider's preconditions. WebSob selects tests whose responses may indicate robustness problems and present the selected tests for manual inspection. We use a packet sniffer or monitor to help facilitate request-response analysis. This monitor acts as a man-in-the-middle between the service consumer and the service provider. The service consumer directs the service request to the monitor who records the request and forwards the request to the service provider. The monitor also records the service response or error condition returned by the service provider. Based on our experience of applying WebSob on various web services, we classify four types of exceptions encountered that may indicate robustness problems:

1. *404 File Not Found.* The 404 or Not Found error message is an HTTP standard response code indicating that the client was able to communicate with the server, but the server either could not find what was requested, or it was configured not to fulfill the request and not reveal the reason why.
2. *405 Method Not Allowed.* The HTTP protocol defines methods to indicate the action to be performed on the Web server for the particular URL resource identified by the client. 405 errors can be traced to configuration of the web server and security governing access to the content of the site.
3. *500 Internal Server Exception.* In certain cases the server fails to handle a specific type of random input generated by JCrasher and produces an Internal Server Exception with an error code of 500. This is the most common exception and contains little insight into what the problem may be.
4. *Hang.* The web service hangs indefinitely or the server takes more than 30 seconds to return a response.

We used WebSob to generate and execute thousands of requests on 35 freely available web services. In many cases it is difficult to determine whether the cause of the exception is a service user error (i.e., the inputs violated some precondition) or a potential problem in the service implementation without access to the service source code. However, in some cases we may infer what the problem may be and what category it may fall under. In addition to manual inspection of selected tests, we may apply more sophisticated analysis such as invariant detection and data mining techniques. This, in conjunction with data visualization, can aide in human understanding of large sets of request-response pairs and produce higher-level rules that may indicate specific problems. For example, perhaps a certain value for a particular argument always leads to a 500 Internal Server Exception. Someone familiar with the problem domain or service implementation may be able to identify bugs directly from such a high-level rule without manual inspection of large sets of test results.

References

[1] Apache. Axis. `http://ws.apache.org/axis/`.

[2] C. Csallner and Y. Smaragdakis. JCrasher: an automatic robustness tester for Java. *Software: Practice and Experience*, 34:1025–1050, 2004.

[3] E. Gamma and K. Beck. JUnit, 2003. `http://www.junit.org`.

[4] E. Martin, S. Basu, and T. Xie. Automated robustness testing of web services. In *Proceedings of the 4th International Workshop on SOA And Web Services Best Practices (SOAWS 2006)*, October 2006.