A Basis Path Testing Framework for WS-BPEL Composite Services

THEERAPONG LERTPHUMPANYA$^{1,2}$ AND TWITIE SENIVONGSE$^1$

$^1$Department of Computer Engineering, Chulalongkorn University
Phyathai Road, Pathumwan, Bangkok 10330
$^2$Department of Software Engineering, Bangkok University
Rama IV Road, Klong Toey, Bangkok 10110
THAILAND

Abstract: - Web services technology has been widely accepted as a technology that can answer to the requirements of the Service-Oriented Architecture in building software applications out of loosely-coupled software components called services. This opens an opportunity to aggregate functionalities of various services in a software composition. Web services technology offers the WS-BPEL language for service composers to create composite services out of other services by having them collaborate in a workflow. These composite services can be used further as building blocks for constructing other composite services. This paper proposes a basis path testing framework for service testers to examine and test basis paths of WS-BPEL services that are being constructed. The framework is equipped with a tool that can automatically generate test cases and stubs for constituent services within the WS-BPEL flow. The framework defines steps for service testers to take in order to generate the test suite and use it for basis path testing. With the framework, service composers can be assured of the compositions before deploying the composite services.

Key-Words: - Basis path testing, WS-BPEL, test cases, control flow graph, cyclomatic complexity

1 Introduction

Web services technology has been widely accepted as a technology that can answer to the requirements of the Service-Oriented Architecture in building software applications out of loosely-coupled software components called services. This opens an opportunity to aggregate functionalities of various services in a software composition. This is due to the fact that service consumers' requirements cannot be fulfilled by a single Web service and hence several Web services are aggregated into a certain composition in order to answer the requirements. Web services technology supports a Business Process Execution Language (WS-BPEL) as a standard for describing a workflow of collaborating Web services [1]. With WS-BPEL, the workflow can be seen as an internal process of a composite Web service which executes by the orchestration of an execution engine. A service composer designs a composite Web service by defining Web service instances that will participate in particular tasks within the workflow. Such a composite service can be used further as a component in the construction of other composite services.

Before its deployment, the service composer will have a service tester test the WS-BPEL service to make sure that the composition is correct. Web services testing tools mostly focus on single Web services, treating them as black boxes and testing their functions or performance. WS-BPEL tools support design but not testing the composition of the designed workflows. In this paper, a framework for basis path testing [2] of WS-BPEL composite services is proposed. The framework describes the steps for service testers to take in order to generate test suites and use them to perform basis path testing on the composite services. A test suite for a particular WS-BPEL workflow comprises a set of test cases and a set of stubs of the constituent services within the flow. Hence the test performed on the WS-BPEL is white box testing, considering the internal process of the composite service while performing black box testing on the constituent Web services through their corresponding stubs. The framework comes with a supporting tool that creates a control flow graph [2] for any WS-BPEL flow and determines its basis paths based on McCabe’s cyclomatic complexity [3]. The tool then generates automatically the test cases and stubs for testing those paths.

Section 2 reviews related research and available Web services testing tools. Section 3 describes basis path testing and introduces a case study. The detail of the framework starts in Section 4 with the test case generation process, followed by the stub
generation process in Section 5. Section 6 explains the basis path testing process with a brief description of the supporting tool in Section 7. The paper concludes with some discussions and future work in Section 8.

2 Related Work
There have been research attempts to define frameworks or create testing tools for Web services. The work in [4] extends WSDL of a Web service with information necessary for testing the service including input-output dependency, sequence of method invocation, hierarchical functional description, and sequence specification. This information helps automate regression testing, data flow testing, and path testing. The work in [5] introduces a tool to measure CPU utilization and response time of a Web service by installing a sensor object at the service provider’s site. The tool can also perform unit testing on the service by using the information in the extended WSDL of [4]. In [6], a test master is used to generate XML-based test cases from the WSDL of a Web service and a test engine is used to run the test cases for the service.

Several testing tools have been commercialized so far such as those in Table 1. They primarily target on testing certain aspects of a Web service and now some are extending to testing at business process level, e.g. SOATest and WebServiceTester support load testing for the entire workflow of Web services instead of load testing individual service endpoints.

<table>
<thead>
<tr>
<th>Tool</th>
<th>By</th>
<th>Testing Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>- Load Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Client Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Regression Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Performance Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Workflow testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Regression Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Load/Pressure Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Performance Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Business Process Orchestration Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Secure Service Testing</td>
</tr>
<tr>
<td>Stylus Studio [9]</td>
<td>Progress Software Corporation</td>
<td>- Functional Testing</td>
</tr>
</tbody>
</table>

Table 1. Web services testing tools (continued)

<table>
<thead>
<tr>
<th>Tool</th>
<th>By</th>
<th>Testing Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>- Performance testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Regression Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Load/Stress Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Performance Testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Client Testing</td>
</tr>
</tbody>
</table>

In summary, research work and commercial tools mostly consider Web services as black boxes to test their functions or performance; none consider internal processes of the composite services for basis path testing.

3 Basis Path Testing for WS-BPEL
Basis path testing [2] is a white box testing that aims for deriving a logical complexity measure of a procedural design of a program and using the complexity measure as a guide for defining a basis set of execution paths. For WS-BPEL, a service tester will perform basis path testing to test whether all the loops and conditions within the workflow give proper outputs and whether existing data flows and control flows are correct. Test cases that exercise the basis set will execute every statement in the workflow at least once during testing.

Basis path testing starts with creating a control flow graph of the WS-BPEL and computing McCabe’s cyclomatic complexity (V(G)) [3] which determines the number of independent paths (i.e. basis paths) in the flow. This number corresponds to the number of test cases needed. In this paper, a WS-BPEL-based loan approval service, adapted from [12], is used as a case study. It is composed of Customer, Assessor, ApprovalA, ApprovalB, and ApprovalC Web services collaborating in approval of a customer’s loan request based on the loan amount and the assessed risk level. Fig. 1 shows a control flow graph of this composite service. Each numbered node in the graph represents a WS-BPEL construct and V(G) of this graph is

\[
V(G) = \text{no. of edges} - \text{no. of nodes} + 2 = 25 - 23 + 2 = 4
\]

According to V(G), there are 4 basis paths for this loan approval service, and therefore 4 test cases are needed to test these paths:

Based on the control flow graph and V(G), our testing framework defines (1) test case generation process (2) stub generation process and (3) basis path testing process. Service testers can follow these processes with assistance from our testing tool. The whole framework is depicted in Fig. 2 and details are as follows.

4 Test Case Generation Process

In Fig. 2, the test case generation process takes as inputs the WS-BPEL flow and WSDL of a composite service as well as WSDLs of constituent Web services within the workflow:

1.) Node IDs are first associated with WS-BPEL constructs; this is for the purpose of representing the executing nodes during testing by our tool. In this research, we use Oracle BPEL designer [13] as a tool to create a composite Web service, therefore its Java embedding feature is used to display node IDs (such as node IDs 1 and 2 are associated with <receive> and <assign> nodes of the WS-BPEL in Fig. 3).

2.) The control flow graph is created, V(G) computed, and basis paths determined. Then a test case for each basis path is generated by analyzing variables related to the path and parameters of service invocation in WSDLs. Test data for each test case are generated based on (1) types of data (2) tester-defined value range (for numerical data) and value length (for string data) (3) tester-defined constants for expected outputs and (4) condition values that cause the control to flow onto that path. Fig. 4 shows the test cases generated for the loan approval service in Table 2.
From Table 2, for example, test case 1 is for testing path 1 listed in Section 3. This path involves Customer and Assessor Web services, and executes when the loan amount is not more than 10000 and the assessed risk level is low. The test case lists data that are needed to execute the path. That is, given the initial input to the flow which is the client’s name and the expected output “Yes” from the tester, the flow will invoke the Customer service which should return the client’s data and loan amount. Our testing tool will generate a random string (“aKdStoGeFQ”) for the client’s data and a random float (8594.21) for the loan amount (types of the client’s data and amount are defined in the WSDL of the Customer service). Note that this random loan amount will cause the control to flow to the Assessor service. The tool will also generate a string “Low” as the output from the Assessor service; this corresponds to the branching condition and triggers the flow to complete path 1 with the expected output “Yes”. Test cases for other paths are generated in a similar manner. All generated test cases for a WS-BPEL are presented in an XML document (they are presented above in a simple table only for brevity).

3.) The test case generation process also creates a template for each constituent Web service in the workflow. The template will be used for generating a stub for each constituent service which will be invoked when testing the flow (see Section 5). For our case study, each of the Customer, Assessor, ApprovalA, ApprovalB, and ApprovalC services will have a template generated. Fig. 4 shows the template for the stub of the Customer service. The template is a WS-BPEL flow that merely receives the same input as the Customer service and assigns certain data values from a particular test case as outputs. The placeholders that will be replaced by the client’s data and loan amount from a particular test case are shown underlined.

Fig. 5. Stub of Customer service for test case 1

### 5 Stub Generation Process

In Fig. 2, the stub generation process takes as inputs the XML-based test cases document and the stub templates from Section 3. Test data are obtained from each test case to replace the data placeholders in relevant templates to create service stubs for that test case. For example, stubs for Customer and Assessor services will be created for test case 1 of Table 2. Fig. 5 shows the Customer stub for test case 1 (test data are shown underlined).
6 Basis Path Testing Process
In Fig. 2, the basis path testing process takes as inputs the WS-BPEL of the composite service with embedded node IDs (c.f. Section 3) and the stubs created for each test case (c.f. Section 4). The service tester will deploy the WS-BPEL and the stubs for each test case by using a WS-BPEL designer tool (i.e. Oracle BPEL designer in this case) and run by a WS-BPEL engine (i.e. Oracle BPEL Process Manager [13] in this case). The execution result will list the WS-BPEL nodes that are visited during the run by using that particular test case; this result will be checked against the basis path generated in Section 4 to see whether the flow is correct and gives proper outputs.

The tool provides a menu for a service tester to (1) open a WS-BPEL file of the composite service to be tested (2) create a control flow graph and find basis paths for the WS-BPEL (Fig. 7-8) (3) specify constants for test data together with range and length for values to be generated at random (Fig. 9-10) and (4) generate test data, stub templates, and stub services for test cases of all basis paths (Fig. 11).

7 Basis Path Testing Tool
The basis path testing tool assists all processes for basis path testing. The service tester first builds a composite service with a WS-BPEL designer. In our case, Oracle BPEL Designer produces a WS-BPEL file and a WSDL file for the composite service, together with an XML file that lists PartnerLinks (i.e. links to WSDLs of constituent Web services). These files are then uploaded to the tool whose tasks are in Fig. 6.

![Fig. 7. Create control flow graph of loan approval](image1.png)

![Fig. 8. Basis path 1 of loan approval](image2.png)

![Fig. 9. Specify range and length for data values](image3.png)

![Fig. 10. Specify constants (input and expected output)](image4.png)

![Fig. 6. Tasks of basis path testing tool](image5.png)
The tool is implemented with Java and its usage is straightforward as it follows steps for running a basis path test on programs in general. Since cross-vendor deployment of WS-BPEL processes and WSDLs is not well-supported (i.e. they are specific to the tool environment that create them), our testing tool may be considered an add-on to Oracle’s BPEL development environment. It would be more useful if we can enhance it to support WS-BPELs and WSDLs from different vendors.

8 Conclusion
This paper proposes a basis path testing framework for WS-BPEL composite services. The framework covers the test case and stub generation processes that produce XML-based test cases for basis paths and WS-BPEL service stubs that are relevant to the test cases. This test suite is aimed for a service tester to deploy and run, e.g. using Oracle BPEL designer and BPEL Process Manager, and the basis paths and expected outputs can then be validated.

Our basis path testing tool is currently at a prototype stage. It supports only sequence, condition, and repetition patterns of control. Since there is correspondence between parallel and sequence patterns [14], it may be possible to transform a WS-BPEL with parallel control into one with sequence. The tool does not yet support all XML schema data types in the generation of test data; only integer, float, boolean, and string are supported but these should represent common data types used in any composite services in general. It does not consider infeasible paths that cannot be accessed either. Such shortcomings will be dealt with in the next version of the tool.

Acknowledgement:
This research is part of the Engineering New Paradigm Software for Enterprises with Service-Oriented Architecture Project (Phase 2), supported by Thailand’s Software Industry Promotion Agency (Public Organization).

References: