An Approach for Monitoring Partner Link Invocations under WS-BPEL

Nuttaphong Uaphoemkiat and Taratip Suwannasart

***Abstract*—Service Oriented Architecture (SOA) is commonly used in organizations to design architecture based on organizational services. To achieve the business goal, the business designer can use existing web services to compose the business process by using WS-BPEL. To ensure each web service works with others correctly, the tester needs to test all web services within WS-BPEL. The challenge in web service testing is to trace web service messages consistency and test the coverage of partner link invocations. This paper presents an approach for monitoring partner link invocations under WS-BPEL. The proposed approach focuses on tracing web services messages using code instrumentation technique, and notifies the untested web services to testers. The proposed approach also covers the additional test case generation for the untested web services as well.**

***Index Terms*—Monitor, Partner Link, WS-BPEL, Web Services**

I. Introduction

IN SOA design and development, the business process has been integrated with web services to reduce development time and redundancies. To design a business process, it should call several web services in order to achieve the business goals. For depicting the overall business process, designers have to use a tool such as BPEL engine for designing, simulating, and testing the process. BPEL engine uses a WS-BPEL as a language to design the process and represents the invocation of web services by using partner link nodes. WS-BPEL is used for the orchestration of web services with respect to the business process. This makes the test of web services important for ensuring that the web services can co-operate correctly.

In integration testing, test cases should cover all web services used in business process to reduce errors in the production environment. Moreover, input and output messages from web services should be traced to ensure that web services are able to work with others correctly.

This paper proposes an approach for monitoring partner link invocations under WS-BPEL to trace the input and output messages from web services as well as examining the coverage of tested web services. Furthermore, our proposed approach will generate additional test cases for exercising untested web services in other feasible paths based on path conditions.

Manuscript received January 8, 2017; revised January 24, 2017.
N. Uaphoemkiat and T. Suwannasart are with the Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand e-mail: Nuttaphong.U@student.chula.ac.th, Taratip.S@chula.ac.th

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 introduces background knowledge of WS-BPEL, BPEL engine, and XSD. Section 4 describes the proposed approach. Section 5 discusses conclusions and future work.

II. Related Work

To prevent errors in a production environment, integration testing is important for WS-BPEL to ensure that each web service can work together and also work correctly. Daniel Lubke and Leif Singer [1] present test coverage metrics for BPEL process using code instrumentation. However, their research does not aim to notify the tester for the untested web services. The tester might ignore untested web services and can neglect to generate more test cases for the test coverage of all web services. From their research, we found the concept of storing input and output message logs by using code instrumentation to trace the messages between web services. Bixin Li et al., [2] and Chang-ai Sun [3] present methodologies for generating test cases using path conditions, by finding all untested paths. Although these researches generate test cases based on all path conditions but we aim to generate test cases to cover all partner link nodes. Therefore, we apply their methodologies to generate test cases according to untested web services.

III. Background Knowledges

A. WS-BPEL (Web Services Business Process Execution Language)

WS-BPEL is a language developed from OASIS [4] to describe the interactions between web services following business processes in XML format. The business process designer can compose the business process using XML tags. To call web services, WS-BPEL provides ‘invoke’, ‘receive’, or ‘reply’ tags to represent partner link nodes [2]. Partner link invocations use a WSDL file (Web Services Description Language) to determine addresses, functions, and parameters of web services [5]. The business constraints in the process are manipulated in WS-BPEL by using XML tags such as “if”, “while”, “repeatUntil”, etc. In addition, exceptions and faults are handled by WS-BPEL as well [6].

B. BPEL Engine

BPEL Engine is a tool to design, execute, and manage the WS-BPEL files. The number of current BPEL Engines is increasing exponentially. Each BPEL Engine may have its

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

own specific functions, e.g. Oracle BPEL Process Manager [7] is a BPEL Engine developed by Oracle Company, which contains many functions such as designing, deploying, executing, and testing business processes [8]. This tool also shows the tested paths executed by the corresponding test cases, and helps in tracing the input and output messages of each BPEL node. Tracing can be performed by clicking on the intended node, and then the tool will display the trace in XML format (Fig. 1). Therefore, the tester has to consider which BPEL nodes to be selected and also requires knowledge of XML to understand the WS-BPEL messages. Moreover, the tool has some gaps, e.g. it reports only tested paths without untested web services (Fig. 2). From the aforementioned gap, it makes the test operation difficult to determine whether the existing test cases cover all web services in the business process. Thus, the tester has to know the BPEL node that interacts with the web services, in order to trace the input and output messages between web services, and also require knowledge of XML to read these trace messages.

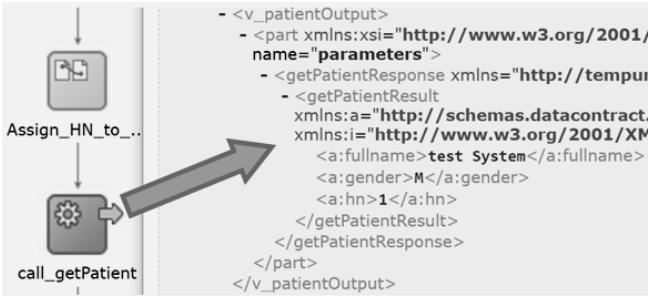

Fig. 1. The Input and Output Messages from the Web Services

Fig. 2. The Result of Testing from BPEL Engine

C. XSD (XML Schema Definition)

XSD is a document that contains the definition of data schema in XML format. XSD is also used in WSDL and WS-BPEL to describe the data schema definition [9]. XSD depicts the data types and constraints of each variable.

IV. Proposed Approach

We present an approach for monitoring partner link invocation under WS-BPEL, in order to trace the input and output messages between web services, and also calculate the test coverage of web services. Our approach also handles the test case generation based on path conditions when untested web services are found, which contains six steps:

A. Create the BPEL flow graph

B. Code Instrument

C. Test the BPEL with test cases

D. Explore the untested web services

E. Create the new test cases

F. Generate the test reports

Fig. 3 represents the overview of the proposed approach and identifies inputs and outputs in each step.

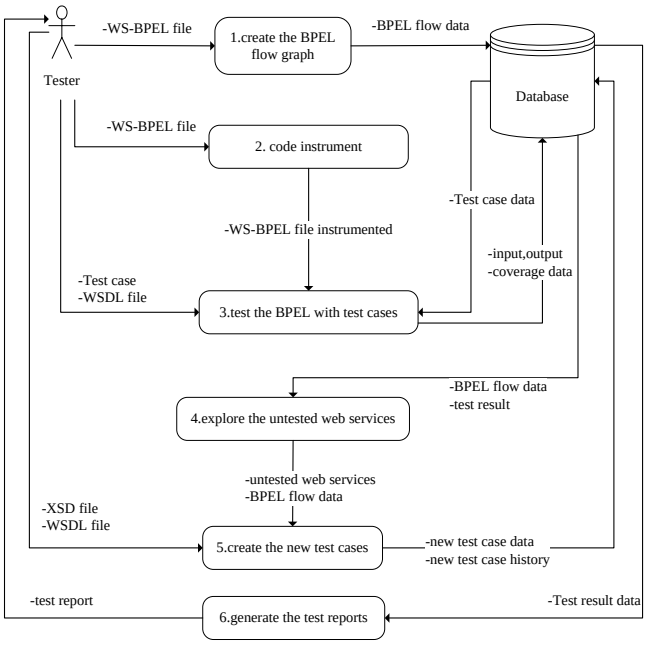

Fig. 3. The Overall Methodology

A. Create the BPEL flow graph

A BPEL flow graph is stored into the database by reading a WS-BPEL file that is uploaded by the tester (The example of the WE-BPEL file is shown in Fig. 4). Then, a BPEL flow graph is generated as shown in Fig. 5, which follows the below tasks:

1) Read a WS-BPEL file from the tester.
2) Analyze the WS-BPEL file to generate the BPEL flow graph by converting each BPEL node from XML format to a node object, which contains attributes such as node id, node name, node type, list of previous node, list of next node, and interaction web service.
3) Store node objects from task 2 into the database.
4) Repeat task 2 to 3 until the end of WS-BPEL file
   The BPEL flow graph data stored in the database are used for comparing with the test web services and untested web services in step D.

```
<invoke name="callPatient_Info"
    partnerLink="patientService"
    portType="ns1:IService1"
    operation="getPatient"
    inputVariable="Invoke1_getPatient_InputVariable"
    outputVariable="Invoke1_getPatient_OutputVariable"
    bpelx:invokeAsDetail="no"/>
```

Fig. 4. The Example of a WS-BPEL File

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

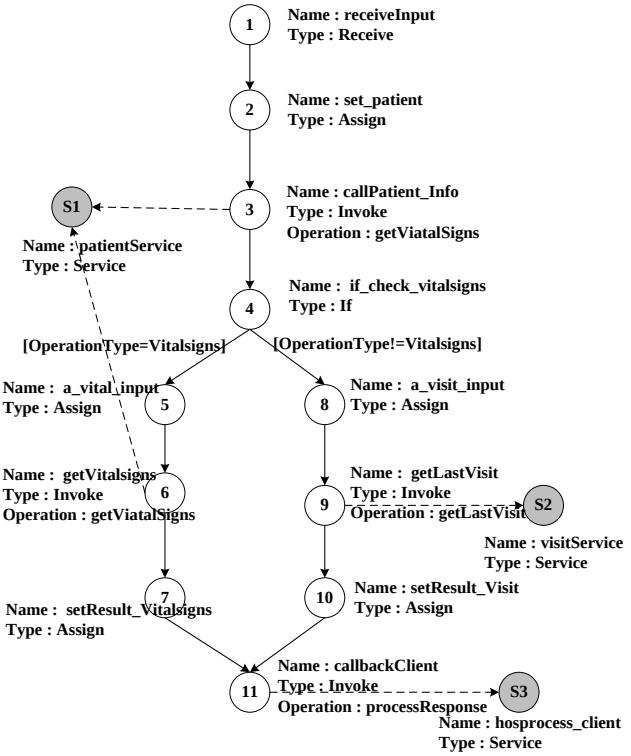

Fig. 5. The Example of the BPEL Flow Graph

B. *Code instrument*

This step uses a code instrumentation technique to collect the test data. When the BPEL engine executes the instrumented line, the BPEL engine will send and store the test data into the database. We have classified the instrumentation node into two kinds of the partner links: invocation nodes and predicate nodes.

1) Invocation node
   For the invocation node, we collect test data using code instrumentation technique. To find the invocation node, every node that contains ‘invoke’, ‘receive’, and ‘reply’ tags, are identified. Then, the code instrumentation is performed by adding a trace node for collecting test data in the WS-BPEL file. This step has the following tasks:
   1.1) Find invocation nodes.
   1.2) Find input variables of web services.
   1.3) Convert web service input variables into XML format.
   1.4) Perform code instrumentation by adding a trace node before the invocation node to call the web service initially by sending input values in XML format.
   1.5) Convert the web service output variables into XML format.
   1.6) Perform code instrumentation by adding a trace node after the invocation node to call the web service again by sending output values in XML format.
   1.7) Repeat task 1.1 - 1.6 until the end of WS-BPEL file.
2) Predicate node
   For the invocation node, we collect test data using code instrumentation technique. To find the invocation node, every node that contains ‘invoke’, ‘receive’, and ‘reply’ tags, are identified. Then, the code instrumentation is performed by adding a trace node for collecting test data in the WS-BPEL file. This step has the following tasks:
   2.1) Find predicate nodes.
   2.2) Read conditions in each predicate node.
   2.3) Perform code instrumentation by adding a trace node after the predicate node to call the web service by sending conditional variables and execution status in object format.
   2.4) Repeat task 2.1 - 2.3 until the end of WS-BPEL file.

After instrumenting all nodes in WS-BPEL file, testers may use this instrumented WS-BPEL file to deploy. Testers may test the deployed WS-BPEL with their existing test cases through the BPEL engine. From the example in Fig. 5, node 1, 3, 6, 9, and 11 are instrumented as invocation nodes type to trace inputs and outputs of web services and node 5, and 8 are instrumented as predicate nodes.

C. *Test the BPEL with existing test case*

In this step, testers can use the instrumented WS-BPEL file from step B to deploy and execute through the BPEL Engine. The execution of the instrumented WS-BPEL file allows tester to collect the test data. Then, testers can test the WS-BPEL using existing test cases. Consequently, the test data are stored into the database when the instrumented lines are executed following the corresponding node type, e.g. the BPEL Engine will send the input and output into the database in XML format when the invocation node type is executed, and the BPEL Engine will send execution status in object format when the predicate node type is executed. The example in Fig. 5 is illustrated as a test case input of ‘operationType’ tag “Vitalsigns” in Fig. 6, and the BPEL Engine sends test data from node 1, 3, 5, 6, and 11.

```
<inputVariable>
 <part xmlns:xsi="http://www.w3.org/2001/XMLSchema-
 instance"name="payload">
    <ns1:process xmlns:ns1="http://xmlns.oracle.com/sample_HOS
    /sampleHos/HosProcess">
       <ns1:hn>5900001</ns1:hn>
       <ns1:operationType>Vitalsigns</ns1:operationType>
    </ns1:process>
 </part>
</inputVariable>
```
Fig. 6. The Example of a Test Case

D. *Explore the untested web services*

The tested nodes (from step C) are compared with the BPEL flow graph (from step A) by finding all web services contained in the BPEL flow graph and comparing it with the tested web service nodes from the test results. From the example in Fig. 5, the web service nodes are 1, 3, 6, 9 and 11. The tested web service nodes are 1, 3, 6 and 11. The example shows the untested web service “getLastVisit” called by node 9 to the tester.

E. *Create the new test cases*

From the previous step, untested web services are notified to testers. Then, the testers can generate additional test cases

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

to test the untested web services. To generate additional test cases, we use a path condition technique, by checking the untested condition nodes and trying to change the condition variables of the existing test cases. This step contains the following tasks:

1) Receive the untested web services and existing test cases from the previous step.
2) Find all feasible paths to each untested web service.
3) Find untested predicate nodes contained in the feasible paths: Untested predicates contained in the feasible paths are checked for identifying the variables in the conditions, to generate new test cases that variable value are received from initial test case and the value is remained the same in WS-BPEL.
4) Change variables from existing test cases based on conditions from the previous task by requesting XSD and WSDL file from the tester.
5) Insert new test cases into database.
6) Repeat task 2-5 until the end of untested web services list.

From the previous example (Step D): The untested web service "getLastVisit" and existing test cases are received. The example test case contains a test of 'operationType' tag "Vitalsigns" for testing predicate node "5". In this case, we found that there is only a feasible path: "1-2-3-4-8-9-10-11". Then, we can identify that the untested condition of node "8" is 'operationType != "Vitalsigns" '. Therefore, the 'operationType' is the variable received from an existing test case that its value in WS-BPEL is remained the same. The additional test case is generated by requesting XSD (Fig. 7) and WSDL (Fig. 8) files from the tester. In the example test case, 'operationType' data type is string. An additional test case is generated by replacing the "test" value into the "Vitalsigns" value (Fig. 9). Finally, an additional test case is inserted into the database.

```
<element name="process">
    <complexType>
      <sequence>
        <element name="hn" type="string"/>
       <element name="operationType" type="string"/>
      </sequence>
    </complexType>
  </element>
```

Fig. 7. The Example of XSD File Describing the Process Schema

```
<wsdl:portType name="HosProcess">
    <wsdl:operation name="process">
      <wsdl:input message="client:HosProcessRequestMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="HosProcessCallback">
    <wsdl:operation name="processResponse">
      <wsdl:input message="client:HosProcessResponseMessage"/>
    </wsdl:operation>
  </wsdl:portType>
```

Fig. 8. The Example of WSDL File Describing Data Type of the Input Messages.

```
<inputVariable>
 <part xmlns:xsi="http://www.w3.org/2001/XMLSchema-
 instance"name="payload">
    <ns1:process xmlns:ns1="http://xmlns.oracle.com/sample_HOS
    /sampleHos/HosProcess">
      <ns1:hn>5900001</ns1:hn>
      <ns1:operationType>Test</ns1:operationType>
    </ns1:process>
 </part>
</inputVariable>
```

Fig. 9. The Example of a New Test Case.

F. Generate the test reports

Two summary reports are generated using the test data from all of the previous steps, which are:

1) Untested web services report
   This report shows the untested web services and notifies the results to testers. This report makes the testers aware of the untested web services and enforce to generate additional test cases for the coverage of integration testing.
2) Test result report
   This report shows the input and output messages received from each web service. This report helps tester to trace the data sent between web services explicitly.

V. Conclusion and Future Work

This paper presents an approach for monitoring partner link invocation under WS-BPEL, to enable testers to trace the input and output messages sent between web services, and also help testers by providing awareness of the untested web services, by notifying the results, and allow testers in generating additional test cases based on path conditions.

For future work, we will develop a monitoring tool that applies the proposed approach in order to evaluate the correctness and applicability in the production environment.

References

[1] D. Lübke, L. Singer, and A. Salnikow, "Calculating BPEL test coverage through instrumentation," *Proc. 2009 ICSE Work. Autom. Softw. Test, AST 2009*, pp. 115–122, 2009.

[2] B. Li, D. Qiu, S. Ji, and D. Wang, "Automatic test case selection and generation for regression testing of composite service based on extensible BPEL flow graph," *IEEE Int. Conf. Softw. Maintenance, ICSM*, no. 3, 2010.

[3] C.-A. Sun, Y. Zhao, L. Pan, H. Liu, and T. Chen, "Automated Testing of WS-BPEL Service Compositions: A Scenario-Oriented Approach," *IEEE Trans. Serv. Comput.*, vol. XXX, no. XXX, pp. 1–1, 2015.

[4] "OASIS Web Services Business Process Execution Language (WSBPEL) TC | OASIS." [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel. [Accessed: 26-Nov-2016].

[5] "OASIS WS-BPEL Extension for People (BPEL4People)." [Online]. Available: http://docs.oasis-open.org/bpel4people/bpel4people-1.1-spec-cs-01.html.

[6] OASIS Web Services Business Process Execution Language (WSBPEL), D. Jordan, and A. Alves, "Web Services Business Process Execution Language Version 2 . 0," *Language (Baltim).*, vol. 11, no. April, pp. 1–264, 2007.

[7] M. B. Juric and D. Weerasiri, *WS-BPEL 2.0 Beginner's Guide*. .

[8] D. Ings *et al.*, "WS-BPEL Extension for People Committee Specification," no. August, pp. 1–57, 2010.

[9] "XML Schema Tutorial." [Online]. Available: http://www.w3schools.com/Xml/schema_intro.asp. [Accessed: 19-Dec-2016].

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)