2016 IEEE International Conference on Services Computing

Test Case Selection for Data Flow Based Regression Testing of BPEL Composite Services

Shunhui Ji[1], Bixin Li[2], Pengcheng Zhang[1]
[1]*College of Computer and Information, Hohai University, Nanjing, China*
[2]*School of Computer Science and Engineering, Southeast University, Nanjing, China*
Email: shunhuiji@hhu.edu.cn

***Abstract*—BPEL(Business Process Execution Language) composite service evolves a lot in its lifetime. Regression testing must be performed to ensure the correctness of each evolved version. In this article, an approach is proposed to select test cases for regression testing based on data flow testing criterion. With XCFG(eXtended Control Flow Graph) modeling BPEL composite service, the approach improves the traditional data flow analysis to compute the def-use pairs in BPEL process, and then identifies the affected def-use pairs by comparing the def-use pairs and XCFG model in the evolved version with those in the baseline version, where related WSDL(Web Service Description Language) documents are incorporated for comparison. The data flow paths covering the affected def-use pairs are calculated for regression testing, and some of them can reuse the test cases in the baseline version, which are determined by analyzing the path condition of data flow paths between two versions. The proposed approach can detect three kinds of changes, including process change, binding change and interface change. Experimental study shows the effectiveness.**

***Keywords*-Web composite service; regression testing; data flow testing; test case selection**

I. Introduction

In practice, BPEL(Business Process Execution Language) composite services must evolve to meet various reasons, such as environment changing, bugs fixing, requirements adding, performance improving, and so on. To assure the correctness of each evolved version, regression testing must be performed to check whether the modifications during evolution have brought any negative effect. And test case selection is a key technique to choose as few test cases as possible to cover all areas affected by modifications.

Selecting proper testing criterion is necessary for performing testing. Consider the most powerful structural testing criterion, all-paths. All-paths based testing can't reveal all the errors. And if there is loop structure in the program, there would be infinite paths, which makes it difficult to cover all the paths in testing[1]. The essence of a program is getting expected output for the given input. The relation between input and output is realized by a series of definitions and uses of variables. It can be concluded that the design of control flow serves for realizing correct data flow. So it is very important to perform data flow testing. The all-uses criterion has been proven to be practical and effective for the testing of BPEL composite services[2]. However, little attention was paid on data flow based regression testing for BPEL composite service.

In this paper, we propose a XCFG(eXtended Control Flow Graph) based test case selection approach for all-uses based regression testing of BPEL composite service. All-use criterion requires each def-use pair to be covered in the testing, where a def-use pair (x, n, n') denotes the definition of variable x in statement n is used in n'. With XCFG, the traditional approach of data flow analysis is improved to get the def-use pairs. To perform all-uses based regression testing, def-use pairs affected by modifications must be identified. Different with traditional programs, BPEL-based composite service is composed of a process, an interface described in WSDL(Web Service Description Language) and partner services interacting with the process. Therefore, the evolution of BPEL composite service usually involves three types of changes, i.e., process change, binding change and interface change[3]. Our approach analyzes the def-uses affected by these three types of changes by comparing the def-use pairs and XCFG model in the evolved version with that in the baseline version, with related WSDL documents incorporated. Then we calculate the data flow paths so that the affected def-use pairs are covered and select reusable test cases from the baseline version by analyzing and comparing path conditions of the data flow paths in two versions. We explore three versions of carefully designed BPEL composite service to show the effectiveness of the proposed approach.

The rest of the paper is organized as follows: Section II introduces WSDL and BPEL briefly; Section III gives a formal definition of XCFG for modeling BPEL composite service; Section IV discusses how to perform test case selection in detail; Section V performs some experiment and evaluation of our approach; Section VI compares the related work; Section VII concludes the paper.

II. Background

A. WSDL Summary

WSDL is a XML-based language for describing web service[4], including the interface, message type and access information. A WSDL document is usually classified into abstract part and concrete part. Abstract part is consisted of

978-1-5090-2628-9/16 $31.00 © 2016 IEEE
DOI 10.1109/SCC.2016.77

portType, *operation*, *message* and *type*, where *portType* contains a set of operations, *operation* defines the interface of web service, *message* defines the data structure of exchanged message, and *type* can be built-in type in XML Schema or complex type that user defines. Concrete part is consisted of *service*, *port* and *binding*, where *service* contains a set of ports, *port* describes an end point as a combination of a binding and a network address, and *binding* specifies how services communicate.

B. BPEL Summary

BPEL[5], as the de-facto standard for service composition, is a XML-based language for specifying the interactions between process and partner services and the execution order among these interactions. The process is composed of multiple activities, which are classified into basic activities and structural activities. Basic activities, including *invoke*, *receive*, *reply*, *assign*, *throw*, *wait*, *empty*, etc, describe the elemental steps and can exist independently or in a structural activity. Structural activities, including *sequence*, *if*, *while*, *repeatUntil*, *pick*, *flow*, *forEach*, *scope*, etc, encode the control-flow of process and can contain basic or structural activities. In addition, BPEL defines the relationship between process and partner services with *partnerLink*. And it defines *variable* to record the data generated during the run of process.

C. A Motivating Example

We use the loan composite service(*LCS*) deployed in Oracle BPEL PM Server as the subject program. Suppose *LCS* has passed through a continuous evolution and two versions are generated. In *LCS v1.0*, *LoanFlow* receives the loan application from client, and then invokes *CreditRatingService* to confirm the loan rank of client using SSN(Social Security Number) in the application form. Following *UnitedLoanService* and *StartLoanServie* are concurrently invoked to deal with the loan. *LoanFlow* receives the loan application result from the two services and select the one who has smaller APR(Annual Percentage Rate) for the client. In *LCS v1.1*, the service integrator modifies an assign activity in the BPEL process of *v1.0*. In *LCS v2.0*, two additional partner services are imported, where *customerService* and *TaskService* respectively provide the function of SSN querying and manual checking for users. We set *v1.0* as the baseline version of *v1.1* and *v2.0*.

III. XCFG Model

Definition *(XCFG)*. XCFG is defined as a quintuple (N, E, PL, V, F), where N is a set of XCFG nodes, E is a set of XCFG edges, PL is a set of partnerLinks, V is a set of variables, and F is the field of each XCFG element.

$N = N_B \cup N_S \cup N_E \cup N_C$. In BPEL, basic and structural activities can be transformed into the four kinds of XCFG nodes according their semantic.

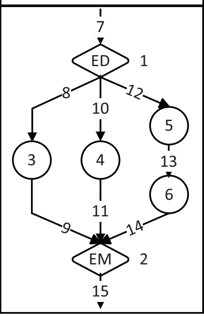
BPEL Activity	**XCFG Construction**	**Field**
<**while** name="A"> <condition> e </condition> <activity name="B"> ...... </activity> </**while**>		The field of branches in while: 1)id=3, outEdges={9}, inEdges={8}, condition=!e 2)id=4, name=B, outEdges={11} inEdges={10}, condition=e 3)id=5, name=B, outEdges={13} inEdges={12}, condition=e 4)id=6, name=B, outEdges={14} inEdges={13}, condition=e The field of edges from ED: 1)id=8, source=1, target=3, type=CHOICE 2)id=10, source=1, target=4, type=CHOICE 3)id=12, source=1, target=5, type=CHOICE

Figure 1. The transformation from while activity to XCFG.

- N_B. It is created for basic activities, such as *receive*, *reply*, *assign*, and so on. Additionally, it is also created for *onMessage* and *onAlarm* in *pick*.
- $N_S = N_{SS} \cup N_{SE}$, where N_{SS} and N_{SE} respectively represent the start and end of *sequence*.
- $N_E = N_{ED} \cup N_{EM}$, where N_{ED} and N_{EM} respectively represent the start and end of choice construct, including *if*, *pick*, *while* and *repeatUntil*.
- $N_C = N_{CB} \cup N_{CM}$, where N_{CB} and N_{CM} respectively represent the start and end of concurrency activity *flow*.

$E = E_C \cup E_L$, where E_C and E_L respectively denote Control Edge and Link Edge.

- E_C. It is created for control flow of activities and has three types, i.e, *Sequence*, *Choice* and *Concurrency*.
- E_L. It is created for links which represent synchronization dependencies among concurrent activities.

F records related information of each XCFG element to support further analysis.

- *id*: records the identification of a node or edge with a natural number. It is unique in XCFG model.
- *name*: records the name of a node, E_L, partnerLink or variable.
- *inEdges/outEdges*: records the set of incoming/outgoing directed control edges of a node.
- *inLinks/outLinks*: records the incoming/outgoing links of a node.
- *readVars/predicateVars/writeVars*: records the c-used (computation used)/p-used(predicate used)/defined variables of a node.
- *condition*: records the execution condition of a node or E_L.
- *partnerLink&portType&operation*: record the interface information about the partner of an interaction activity.
- *endpoint*: records the address of partner service in partnerLink.
- *type*: records the type of a E_C or variable. The types of variables can be WSDL message type, XML Schema

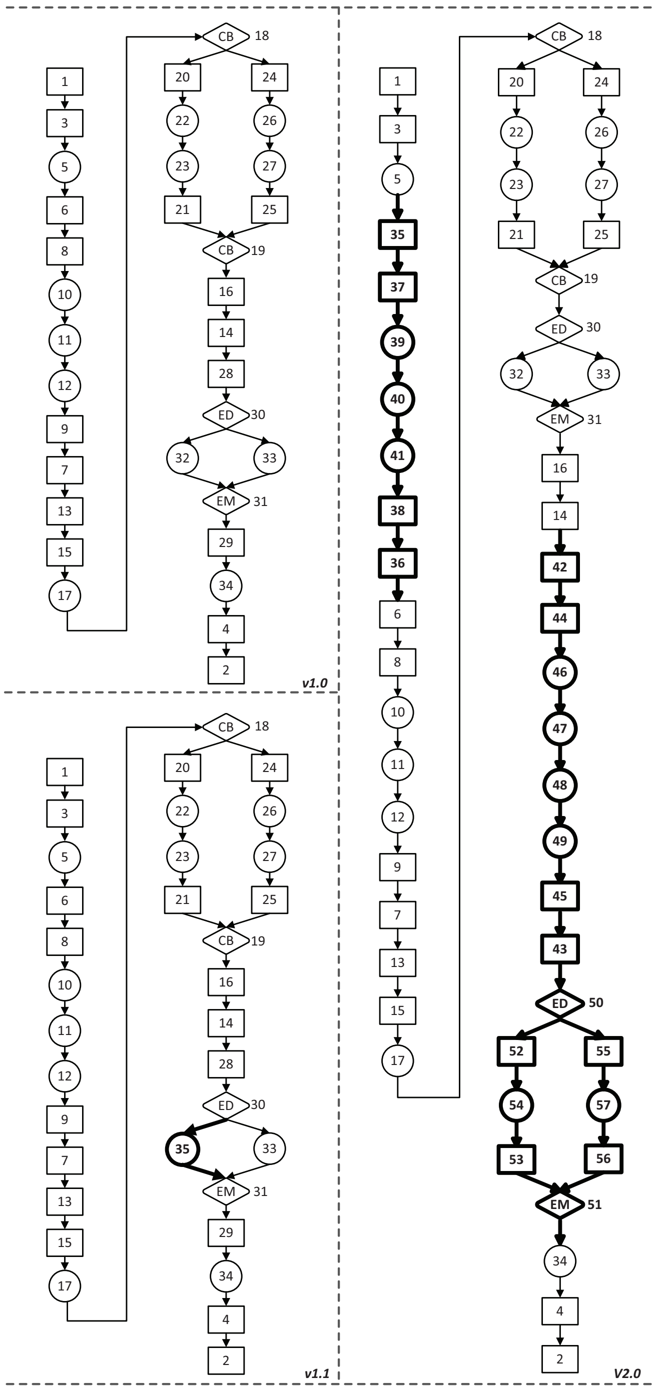


Figure 2. XCFG models of three versions of *LCS*.

built-in type, or user-defined complex type.

- *concreteType*: records the concrete type of a variable. It only exists in the variables of WSDL message type.

Compared with the XCFG model in our earlier work[6], not only the information of partner service is described here, but also the model of loop structure is improved. To simplify the model and keep the full data flow of BPEL process, we model the child activities in the loop structure to execute for zero time, one time and two times. Take *while* whose condition is e as example, the transformation is shown in Fig.1. First a pair of N_{ED} and N_{EM} is created to respectively represent the start and end of while, and then three branches are constructed between N_{ED} and N_{EM}. The first branch describes the situation that child activity *B* executes for zero time, where the leftmost edge outgoing from N_{ED} connects to a node corresponding to empty activity with the $condition$ field set as $!e$. The second branch describes the situation that *B* executes for one time, where the middle edge outgoing from N_{ED} connects to a node corresponding *B* with the $condition$ field set as $!e$. The third branch describes the situation that *B* executes for two times by constructing two nodes representing *B* and linking the two nodes with a *Sequence* typed edge.

The XCFG models of the three versions of *LCS* are shown in Fig.2, where the modifications are marked in bold.

IV. Test Case Selection

Suppose we are going to operate all-uses criterion based regression testing on BPEL composite service S_2 against the baseline version S_1, with the XCFG models constructed which are respectively denoted as G_1 and G_2, the steps of performing test case selection are as follows:

(1) Data Flow Analysis. In the all-uses based data flow testing, every definition to its use of each variable should be covered. So data flow should be analyzed based on XCFG model to get the def-use pairs of BPEL composite service. The main task is to calculate the def-use pairs of S_1 and S_2: DU_1 and DU_2.

(2) Affected def-use pairs identification. The main task is to analyze the impacts that process change, binding change and interface change bring and recognize the affected def-use pairs DU^s. With the related WSDL documents incorporated, the identification of DU^s can be performed by comparing DU_1 with DU_2 and G_1 with G_2, .

(3) Data flow path computation. The main task is to compute the complete XCFG paths P^s covering DU^s for regression testing. It is feasible to find all paths P by traversing XCFG . Then P^s can be obtained by choosing the minimum set of P to cover DU^s. Also paths P_1 that cover DU_1 can be calculated in the same way.

(4) Path condition analysis. The main task is to generate path condition for P_1 and P^s. If $\exists p_2 \in P^s$ and $\exists p_1 \in P_1$ where the path condition of p_2 is the same as that of p_1, then the test cases attached to p_1 can be reused to test p_2.

A. Data Flow Analysis

In the traditional data flow analysis, reaching definition analysis technique can be used to compute the def-use pairs of program, which is carried out based on CFG(Control Flow Graph) to get the definitions of variables that reach each node. For definition $d = (x, n)$, which denotes variable x is defined in node n, if there is a path that comes from n to node n' such that x is not redefined by any other node along the path, d is regarded as reaching n'. Let $In(n)$ and $Out(n)$ denote the reaching definitions at the point before and after each node n, the computation formulas are as follows[7]:

$$Out(n) = Gen(n) \cup (In(n) - Kill(n)) \qquad (1)$$

$$In(n) = \bigcup_{p \in pred(n)} Out(p) \qquad (2)$$

where $Gen(n)$ is the set of definitions generated by the statement in n, $Kill(n)$ is the set of all other definitions of the defined variable in the program, and $pred(n)$ is the set of predecessor nodes of n.

Compared with traditional sequential program, BPEL specification has new characters, such as concurrency and synchronization dependency between concurrent activities. XCFG adds corresponding elements to CFG for modeling these characters. Considering the newly added elements in XCFG, we modify Eq.(2) to Eq.(3).

$$In(n) = \bigcup_{p \in pred(n)} Out(p) - \bigcup_{m \in K} Kill(m) \qquad (3)$$

where K is the set of nodes whose corresponding activities are contained in the concurrency structure and will always be executed before n if n is the end node of concurrency structure or the target node of link edge, and K is $\emptyset$ for the other nodes.

If an activity in the concurrency structure will always be executed, which means it doesn't exist in any choice structure, then the definitions it kill must not reach the end node of this structure. So was it for the target node of link edge. Using the improved equations to analyze the data flow of XCFG can get the correct definitions for all the XCFG nodes except the nodes that correspond to the child activities in the concurrency structure. However, this flaw does not affect correctly calculating the def-use pairs in BPEL process because there is no data flow dependency between the concurrent activities.

The classic solution to the equations of data flow is MFP(Maximal Fixed Point) method[6], which iteratively computes Eq.(1) and Eq.(3) with $In()$ and $Out()$ initialized as $\emptyset$ until the convergence occurs. With the reaching definitions of each XCFG node calculated by MFP solution, the def-use pairs of variable x can be computed as follows:

$$du(x, n) = \{n' | (x, n) \in In(n') \wedge x \in use(n')\} \qquad (4)$$

$$du - pairs(x) = \{(x, n, n') | x \in def(n), n' \in du(x, n)\} \qquad (5)$$

where $def()$ and $use()$ are the set of variables defined and used by the statements. $use()$ consists of computation used variables and predicate used variables, which are represented by $c\text{-}use()$ and $p\text{-}use()$ respectively. To support the further analysis, the def-use pairs are classified into two types: *Computation* and *Predicate*, depending on whether the variable x belongs to $c\text{-}use(n')$ or $p\text{-}use(n')$ for def-use pair (x, n, n').

B. Affected Def-use Pairs Identification

Process change, binding change and interface change will bring direct or indirect effects to the data flow of BPEL composite service. Affected def-use pairs are mainly classified into three types[8]:

- Directly affected def-use pairs. They are newly created or deleted def-use pairs in the evolved version compared with the baseline version.
- Value affected def-use pairs. They are def-use pairs whose computed value has been changed.
- Dependency condition affected def-use pairs. They are def-use pairs whose executuion conditions has been changed.

Process change denotes the modification of the structure of composite service, including addition or deletion of the partner services, modification of activities, and modification of execution order of activities. It will cause all the three kinds of affected def-use pairs.

1) Directly affected def-use pairs by process change. Directly affected def-use pairs of *Computation* type can be recognized by comparing DU_1 with DU_2 and should be added to DU^s, where the comparison between two def-use pairs is performed following Eq.(6). The directly affected def-use pairs of $Predicate$ type will be computed when analyzing dependency condition.

$$(x1, n1, n2) == (x2, n1', n2') \Leftrightarrow$$
$$(x1 == x2) \wedge (n1 == n1') \wedge (n2 == n2') \qquad (6)$$

2) Value affected def-use pairs by process change. We search the nodes whose definition values are changed and record them in the set $valueChangeSet$. For $\forall (x1, n1, n2) \in DU^s$, the value of c-used variable in $n2$ has been changed. So if $\exists (x2, n2, n3) \in DU_2$, where $DU_2 = DU_2 - DU^s$, then $n2$ should be added into $valueChangeSet$ since the definition value of $x2$ changes, and $(x2, n2, n3)$ should be added into DU^s. The value changes in $valueChangeSet$ propagate forward following the nodes c-use the values of them. By analyzing the propagation, we can get all the value-changed nodes and value affected def-use pairs of $Computation$ type, as shown in Alg. 1. With $valueChangeSet$, we can also get the value affected def-use pairs of $Predicate$ type.

3) Dependency condition affected def-use pairs by process change. Dependency condition is recorded in the $condition$ field of some node. We record the nodes whose $condition$s change in the set $conditionChangeSet$. For value affected def-use pairs of $Predicate$ type, the use node should be added into $conditionChangeSet$ because its $condition$ field is changed. Then the condition change

Algorithm 1 valueImpact()

```
Input:
  $valueChangeSet$: the set of nodes whose definition value changes;
  $DU_2$: the def-use pairs in the evolved version;
  $DU^s$: the def-use pairs affected by the changes;
Output:
  $DU^s$: the def-use pairs affected by the changes;
1: copy $valueChangeSet$ to $tempSet$;
2: while $tempSet! = \emptyset$ do
3:   take an element $node$ from $tempSet$ and delete $node$ from $tempSet$;
4:   for $du - pair \in DU_2$ do
5:     get the definition node and the use node of $du - pair$: $defNode$ and $useNode$;
6:     if $du - pair.type == "Computation" \wedge defNode == node$ then
7:       $DU^s = DU^s \cup \{du - pair\}$;
8:       if $useNode.writeVars! = \emptyset \wedge useNode \notin valueChangeSet$ then
9:         $valueChangeSet = valueChangeSet \cup \{useNode\}$;
10:        $tempSet = tempSet \cup \{useNode\}$;
11:      end if
12:    end if
13:  end for
14: end while
15: return $DU^s$;
```

caused by $condition$ itself can be recognized by comparing $condition_2$ of n_2 in $S2$ with $condition_1$ of corresponding n_1 in $S1$. If $condition_2! = condition_1$, n_2 should be added into $conditionChangeSet$, and for $\forall x \in \{x|x \in p-use(n2) \wedge x \notin p-use(n1)\}$, the def-use pairs of $Predicate$ type of x with n_2 as the use node in DU_2, where $DU_2 = DU_2 - DU^s$, are newly added and should be added into DU^s. If $condition_2 == condition_1$, finding the def-use pairs control depending on $condition_2$ but not $condition_1$ so that they are added into DU^s since their dependency conditions have changed. If there is no $condition_1$ in $S1$ corresponding to $condition_2$, which means $condition_2$ is a newly added dependency condition, then it should be added into $conditionChangeSet$ and any def-use pair of $Prediate$ type with n_2 as the use node should be added into DU^s because it is a newly added $Predicate$ typed def-use pair. With the $conditionChangeSet$, we can get the def-use pairs that control depend on the $conditon$s of them and put them into DU^s.

Binding change denotes the replacement of a partner service by another candidate service with the same functionality. It will cause value and dependency condition affected def-use pairs. Let PL_1 and PL_2 denote the partnerLinks of S_1 and S_2. The partnerLinks $plChangeSet$ whose binding addresses have changed can be recognized by comparing PL_2 with PL_1. $\forall pl_2$ that satisfies Eq.(7) should be added into $plChangeSet$. If node $n2$ in S_2 interacts with $pl_2 \in plChangeSet$ and $\exists x \in def(n2)$, then the definition value of $n2$ is regarded as changed since the new partner service hasn't been checked whether it can provide expected result and $n2$ should be added into $valueChangeSet$. The value change in $valueChangeSet$ will cause value affected def-use pair of both $Computation$ and $Predicate$ type. Further the value affected $Predicate$ def-use pairs will cause dependency condition affected def-use pairs. These affected def-use pairs can be analyzed with the same method shown above.

$$\exists pl_2 \in PL_2, \exists pl_1 \in PL_1 \rightarrow (pl_2.name == pl_1.name) \wedge (pl_2.endpoint! = pl_1.endpoint) \quad (7)$$

Interface change denotes the modification of the interfaces of services, including composite service and partner services, including change of elements in WSDL documents, such as type, message, operation and port. The changes of operation and port will cause passive changes in the composition process, which have been dealt with in the analysis of process change. The changes of type and message will cause the changes of concrete types of WSDL message typed variables, which cannot be seen in BPEL composite service. So the def-use pairs affected by the changes of type and message are analyzed by incorporating WSDL documents. These affected def-use pairs need new test cases generated since the concrete types of variables have been changed when retested in the regression testing. Let V_1 and V_2 denote the variables in S_1 and S_2, by pair-wise comparison between variables of V_1 and V_2, we can get the variables $typeChangeSet$ whose concrete types are changed. $\forall v_2$ that satisfies Eq.(8) should be added into $typeChangeSet$. Any def-use pair of the variable x with $x \in typeChangeSet$ should be added into DU^s.

$$\exists v_2 \in V_2, \exists v_1 \in V_1 \rightarrow (v_2.name == v_1.name) \wedge (v_2.type == v_1.type) \wedge (v_2.concreteType! = v_1.concreteType) \quad (8)$$

C. Data Flow Path Computation

A complete XCFG path is a set of finite XCFG nodes, $p = (n_1, n_2, ..., n_k)(k \geq 2)$, where n_1 and n_k are the start node and the end node of XCFG, and $\forall n_i(1 \leq i \leq k-1)$, $(n_i, n_{i+1}) \in E$. If there is a child path $p' = (n, n_i, n_{i+1}, ..., n_j, n')$ contained in p, where $x \in def(n) \wedge x \in use(n')$ and no node among $n_i, n_{i+1}, ..., n_j$ defines x, then we say p is a path that covers the def-use pair (x, n, n').

To calculate the paths for covering def-use pairs DU, there are two steps:
1) Generating all the complete XCFG paths P of BPEL composite service;
2) Selecting the minimum set P' from P to cover every def-use pair in DU.

In XCFG, the $inEdges$ and $outEdges$ fields are attached to each nodes to record the incoming and outgoing control edges, and the $source$ and $target$ fields are attached to each edge to record the source and target nodes. Using these information, XCFG can be traversed to generate complete XCFG paths P. In the process of traversing XCFG, various types of XCFG nodes can be encountered and different type needs different dealt. If the current node $n \in N_{ED}$ and it has k branches, then excepting adding the nodes outgoing from one of its branches to the current XCFG path, another $k-1$ paths are copied to describe the other $k-1$ branches. If $n \in N_{CB}$, since the execution order of the child activities of the concurrency structure is non-deterministic and the same outputs are expected for the possible multi execution sequences, so we treat them as one path and add all the nodes outgoing from N_{CB} to the current path. Later when its end node $n' \in N_{CM}$ is encountered, n' cannot be added into the current path until all the source nodes of n' are included. If $n \in N_B \cup N_S \cup N_{EM}$, just adding it to the current path. The process for constructing a XCFG path terminates when the end node of XCFG is added.

To decide whether a XCFG path covers the def-use pair $(x, n, n') \in DU$, we must check whether any node n'' that defines x and kills the definition of x in n exists. Such nodes should be avoided in the data flow path of (x, n, n'), denoted as the set $avoidSet$. With the $avoidSet$, a XCFG path p is the data flow path that covers (x, n, n') if and only if the following condition is satisfied:

$$(p \in P) \wedge (n \in p) \wedge (n' \in p) \wedge !(avoidSet \subset p) \quad (9)$$

Selecting the minimum set P' from P so that all def-use pairs in DU are covered is the problem of minimum set selection. We make a list of all sub-sets of P and check them in the order from the sub-set with the least number to that with the largest number.

D. Path Condition Analysis

To make full use of test cases of the baseline version and avoid redundant test case generation, we adopt the principle of predicate logic and compare path conditions between the data flow paths in P^s and P_1. If the path conditions of $p1 \in P_1$ and $p2 \in P^s$ are proved to be identical, the test cases required for $p2$ can be selected from the baseline version, which are those attached to $p1$. Most paths in P^s can be analyzed to select test cases with this method, except the paths of def-use pairs that are affected by interface change.

The predicate constraints of a XCFG path come from the $condition$ field of two kinds of nodes, where one is the node whose direct precedent node is N_{ED}, and the other is the target node of E_L. Formally, predicate constraint is defined as a triple $prc =< EP, PT, F >$, where EP is the constraint expression, PT is the predicate type and $PT = \{Boolean, Numeric, String\}$, F denotes how prc is combined in path condition and $F = \{AND, OR\}$. Path condition pac is a vector containing predicate constraints of this XCFG path. As predicate constraints are bound with XCFG nodes, path condition of XCFG path p is defined as follows:

$$pac = \bigcup_{i=1}^{k} \{n_i.prc | n_i \in p\} \quad (10)$$

where k denotes the number of nodes in p.

Two path conditions pac_1 and pac_2 of XCFG path p_1 and p_2 are identical if and only if for $\forall prc \in pac_2$ and corresponding $prc' \in pac_1$, $prc == prc'$. The comparison between two predicate constraints are decided by the three elements, including EP, PT and F.

V. Experimental Study

A. Experimental Design

Suppose the actual numbers and covered numbers of process changes are denoted as $\triangle_{pc}$ and num_{pc} respectively. Process change mainly includes the changes of activities, links and execution order, which are modeled as nodes, E_L and E_C in XCFG. For the three kinds of XCFG elements, let n_n, n_m and n_d denote the new elements, modified elements and deleted elements, since we have analyzed the impact of all these changes, the coverage rate of process changes is evaluated as follows:

$$\rho_{pc} = \frac{num_{pc}}{\triangle_{pc}} \times 100\% = \frac{n_n + n_m + n_d}{n_n + n_m + n_d} \times 100\% = 100\% \quad (11)$$

Suppose the actual numbers and covered numbers of binding changes are denoted as $\triangle_{bc}$ and num_{bc} respectively. Binding change can be reflected in XCFG too. Let b_n, b_m and b_d denote the number of new, modified and deleted bindings, since we have analyzed the impact of new bindings and modified bindings, the coverage rate of binding changes is evaluated as follows:

$$\rho_{bc} = \frac{num_{bc}}{\triangle_{bc}} \times 100\% = \frac{b_n + b_m}{b_n + b_m + b_d} \times 100\% \quad (12)$$

Suppose the actual numbers and covered numbers of interface changes are denoted as $\triangle_{ic}$ and num_{ic} respectively. As for the interface change, WSDL document is composed of the definitions of message, type, portType, operation, binding and port. Let M_d, T_d, PT_d, O_d, B_d and P_d denote the number of corresponding element. Some of the changes of portType, operation, binding and port can be reflected in BPEL activities, which have been analyzed in the analysis of process change. And some of the changes of message and type are analyzed in the analysis of interface change. Let M_u, T_u, PT_u, O_u, B_u and P_u denote the changes of corresponding elements that can be covered by our approach, then the coverage rate of interface changes is evaluated as follows:

$$\rho_{ic} = \frac{num_{ic}}{\triangle_{ic}} \times 100\% = \frac{M_u + T_u + PT_u + O_u + B_u + P_u}{M_d + T_d + PT_d + O_d + B_d + P_d} \times 100\% \quad (13)$$

B. Experimental Result

A prototype tool, named XCFG4BPEL, has been developed for implementing the automatic regression test case selection. According to the experimental result and the evaluation equations for process change, binding change and interface change, we will present the coverage rate of the regression testing of *LCS v1.1* and *v2.0*.

In *v1.1*, only the process change happens compared to *v1.0*, where an activity is changed. One data flow path whose test cases can be selected from *v1.0* needs to be retested in the regression testing. Reflected in corresponding XCFG model, one node and two connecting edges are changed. So $\rho_{pc}[1.1] = 3/3 = 100\%$.

In *v2.0*, all the three kinds of change types, including process change, binding change and interface change, happen compared to *v1.0*. Two data flow paths who require new test cases generated need to be retested. The coverage of different change types are given as follows:

- Process change: 49 XCFG elements are added or modified and the experiment covers all 32 elements, so $\rho_{pc}[2.0] = 49/49 = 100\%$.
- Binding change: 2 bindings are added and the experiment covers both 2 bindings, so $\rho_{bc}[2.0] = 2/2 = 100\%$.
- Interface change: 268 WSDL elements are added in the new 3 WSDL documents. Some of these elements are used in BPEL composite service, by statistics we have $M_u = 5$, $T_u = 29$, $PT_u = 3$, $O_u = 3$, $B_u = 3$, $P_u = 3$, so $\rho_{ic}[2.0] = 46/268 = 17.16\%$.

From the evaluation result we can see that the change coverage of interface change is on the low side compared to the other two coverages. This is mainly due to the reason that only WSDL elements that are related to BPEL composite service are considered in our approach. In *v2.0*, most elements in *TaskServiceWSIF.wsdl* and *TaskServiceInterface.wsdl* are irrelative to BPEL process. Since the added WSDL elements are not used in *LCS*, deleting these elements won't affect the execution of BPEL process. So we perform 3 experiments to see whether the coverage rate of interface change will increase, where the interfaces of the new WSDL documents are changed. In the experiment Ep1, we delete 10 definitions of irrelative type. The result is still 46 changed WSDL elements are covered, so $\rho_{ic} = 46/258 = 17.83\%$. In the experiment Ep2, we delete another 10 definitions of irrelative message. The result is still 46 changed WSDL elements are covered, so $\rho_{ic} = 46/248 = 18.55\%$. In the experiment Ep3, we delete another 10 definitions of irrelative operation. The result is still 46 changed WSDL elements are covered, so $\rho_{ic} = 46/238 = 19.33\%$. As shown in Fig.3, the experiments show that the coverage rate of interface change keeps increasing when the number of irrelative WSDL elements becomes smaller.

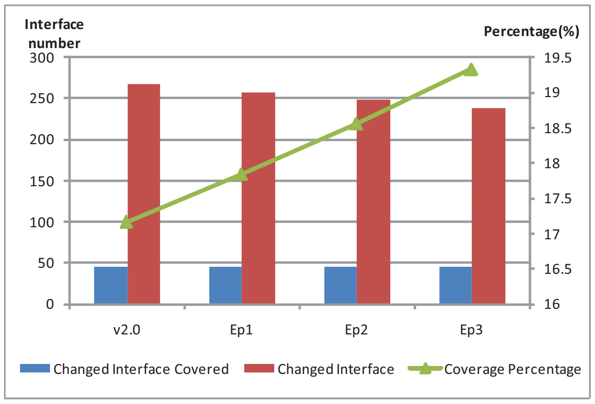


Figure 3. The coverage rate of interface change.

VI. Related Work

In the regression testing of Web composite service, there are mainly three techniques, including test case prioritization[9], test suite minimization[10], and test case selection[3,11-19]. And in the research of test case selection, majority of the techniques are modification-aware[20].

Liu et al. proposed a BFG(BPEL Flow Graph) based path analysis technique to perform test case selection for BPEL composite service[11]. The work focused on the changes of BPEL concurrency structures and can not identify the newly added structural activities in the processes. The limitation was overcame in the work[12]. Li et al. proposed XBFG(eXented BPEL Flow Graph) model to modeled not only the BPEL process but also the interactions between the process and its partner services[3], and WSDL was also required so that it was capable of handling binding changes.

Lin et al. proposed a safe regression test selection for Java-based Web Service[13]. JIG(Java Interclass Graph), a CFG extension for java program, was used to model Web composite service. JIG requires the input of WSDL to be transformed to a local java program that simulate the related functionalities and behaviors. Then they used graph walk technique to perform test case selection.

Ruth et al. composed CFGs of participating services to construct the global CFG of Web composite service[14,15]. They took graph walk technique to identify the dangerous edges by comparing the global CFGs of two version and select test cases to be rerun from the test suite. [15] improved the approach of [14] by considering the three possible scenarios of concurrent changes on services to ensure the test consistency. They then further perfected the the approach by considering the privacy issue of shared services during test selection process[16,17].

Tarhini et al. modeled Web composite service with TLTS(Timed Labeled Transition System)[18], where internal behaviors of the participating services were also included. Using modification-based technique, they identified modifications from generated TLTS and selected test cases that would cover the modified part.

Khan et al. took TGTS(Typed Graph Transformation System) to model Web composite service[19], and based on the analysis of the dependencies and conflicts between visual contracts which specify the preconditions and effects of operations, dependency-based technique was used to select test cases.

Among the above researches, none of them is carried out based on the data flow testing criterion. Most techniques can handle process change, 3 techniques can handle interface change[3,18,19], and only 2 techniques can handle binding change[3,18]. In this paper, based on the all-uses data flow testing criterion, three types of changes, including process change, binding change and interface change, can be identified and handled in the process of test case selection.

VII. Conclusion

In this paper, we proposed a XCFG based test case selection approach for the regression testing of BPEL composite service. Different from the other methods, our work is based on the all-uses data flow testing criterion. And the proposed approach can cover the affected data flow caused by process change, binding change and interface change. Experimental study shows that our approach has a high coverage rate for the changes. Besides test case case selection, our approach can also be used for the all-used criterion based test case generation for BPEL composite service of single versions.

Our research represents an initial work on the data flow based regression testing for Web composite service. There are a lot of work to do in the future, such as performing more large-scale experimental study, using the proposed approach to Web composite service described in other languages, and so on.

Acknowledgment

This work is supported by the Fundamental Research Funds for the Central Universities (No. 2015B12714) and the National Natural Science Foundation of China (Nos. 61572171 and 61202097).

References

[1] Rapps S and Weyuker E.J. *Selecting Software Test Data Using Data Flow Information*. IEEE Transaction on Software Engineering, 1985, SE-11(4):367-375.

[2] Mei L, Chan W.K., Tse T.H., and Kuo F.C. *An Empirical Study of the Use of Frankl-Weyunker Data Flow Testing Criteria to Test BPEL Web Services*. In Proc. of the 33th Annual IEEE International Conference on Computer Software and Application, Seatlle, WA, 2009:81-88.

[3] Li B, Qiu D, Leung H, and Wang D. *Automatic Test Case Selection for Regression Testing of Composite Service Based on Extensible BPEL Flow Graph*. The Journal of Systems and Software, 2012, 85(6):1300-1324.

[4] Roberto C et al. *Web Services Description Language (WSDL) Version 2.0*. W3C Recommendation, http://www.w3.org/tr/wsdl20/, 2007.6.

[5] Alves A et al. *Web Services Business Process Execution Language Version 2.0*. OASIS Standard, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2007.4.

[6] Li B, Ji S, Qiu D, Leung H, and Zhang G. *Verifying the Concurrent Properties in BPEL Based Web Service Composition Process*. 3rd ed. IEEE Transactoin on Network and Service Managemen, 2013, 10(4):410-424.

[7] Aho A.V, Sethi R, and Ullman J.D. *Compilers, Principles, Techniques, and Tools*. New York: Addision-Wesley, 2006:620-631.

[8] Gupta R, Harrold MJ, and Soffa ML. *Program Silicing-Based Regression Testing Techniques*. Journal of Software Testing, Verificatingt and Reliability, 1996, 6(2):83-111.

[9] Mei L, Cai Y, Jia C, Jiang B, Chan W.K and Tse T.H. *A Subsumption Hierarchy of Test Case Prioritization for Composite Services*. IEEE Transactions on Service Computing, 2015, 8(5):658-673.

[10] Bozkurt M. *Cost-Aware Pareto Optimal Test Suite Minimisation for Service-Centric Systems*. In Proc. of the 15th Annual Conference on Genetic and Evolutionary Computation, Amsterdam, Netherlands, 2013:1429-1436.

[11] Liu H, Li Z, Zhu J, and Tan H. *Business Process Regression Testing*. In Proc. of the 5th International Conference on Service-Oriented Computing, Vienna, Austria, 2007:157-168.

[12] Li ZJ, Tan HF, Liu HH, Zhu J, and Mitsumori NM. *Business-Process-Driven Gray-Box SOA Testing*. IBM System Journal, 2008, Vol.47, No.3, pp.457-472.

[13] Lin F, Ruth M, and Tu S. *Applying Safe Regression Test Selection Techniques to Java Web Services* . In Proc. of International Conference on Next Generation Web Service Practices, Seoul, 2006:133-142.

[14] Ruth M and Tu S. *Concurrency Issues in Automating RTS for Web Services*. In Proc. of IEEE International Conference on Web Services, Salt Lake City, UT, 2007:1142-1143.

[15] Ruth M.E. *Concurrency in A Decentralized Automatic Regression Test Selection Framework for Web Services*. In Proc. of 15th ACM Mardi Gras Conference, Baton Rouge, LA, 2008, No.7.

[16] Ruth M.E. *Employing Privacy-Preserving Techniques to Protect Control-Flow Graphs in A Decentralized, End-to-end Regression Test Case Selection Framework for Web Services*. In Proc. of IEEE 4th International Conference on Software Testing, Verification and Validation Workshop, Berlin, 2011:139-148.

[17] Ruth M.E and Rayford C. *A Privacy-Aware, End-to-End, CFG-Based Regression Test Selection Framework for Web Services Using Only Local Information*. In Proc. of 4th International Conference on the Applications of Digital Information and Web Technologies, Stevens Point, WI, 2011:13-18.

[18] Tarhini A, Fouchal H, and Mansor N. *Regression Testing Web Services-based Applications*. In Proc. of the 2006 IEEE International Conference on Computer Systems and Applications, Dubai/Sharjah, UAE, 2006:163-170.

[19] Khan T.A and Hechel R. *On Model-Based Regression Testing of Web-services Using Dependency Analysis of Visual Contract*. In Proc. of 14th International Conference On Fundamental Approaches to Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software, Saarbrcken, Germany, 2011:341-355.

[20] Yoo S and Harman M. *Regression Testing Minimization, Selection and Prioritization: A Survey*. Software Testing, Verification and Reliabitliy, 2012, 22(2):67-120.