Business Process Regression Testing

Hehui Liu, Zhongjie Li, Jun Zhu, and Huafang Tan

IBM China Research Laboratory, Beijing 100094, China
{hehuiliu, lizhongj, zhujun, tanhuaf}@cn.ibm.com

Abstract. Business Process Execution Language(BPEL) has been recognized as a standard for the service orchestration in Service Oriented Architecture(SOA). Due to the pivotal role played by BPEL in service composition, the reliability of a business process becomes critical for a SOA system, especially during its evolution.

Regression testing is well known as an effective technology to ensure the quality of modified programs. To reduce the cost of regression testing, a subset of test cases is selected to (re)run, known as regression test selection. Previous work addressing this problem will fail in the presence of concurrent control flow, which is an important and widely used feature of BPEL in describing service orchestration. In this paper, a regression testing approach for BPEL business processes is presented. In this approach, an impact analysis rule is proposed to identify the test paths affected by the change of BPEL concurrent control structures. Based on the impact analysis result and process changes identification, the impacted test paths are classified into reusable, modified, obsolete and new-structural paths. Experiments show that our approach is feasible.

1 Introduction

Service Oriented Architecture (SOA) is continually gaining more application in software industry for the automation of business processes and the integration of IT systems. In SOA, the service orchestration that combines several web services into a more complex one is a crucial building block [2]. Business Process Execution Language(BPEL) is a standard for describing such service orchestration. For the pivotal role played by BPEL in service composition, the reliability of business processes becomes especially critical for a SOA system. More importantly, the dynamic and adaptive nature of SOA also requires the business processes evolve more quickly and meanwhile puts forward more rigorous demand on the quality of the processes during the maintenance of a SOA system.

Regression testing is well known as an effective technology for verifying the behavior of modified programs [5]. After a program has been changed, obviously, the simplest way is to rerun all test cases, which is called as retest-all strategy [5] in regression testing. However, this strategy is expensive for executing unnecessary tests. Another strategy called as selective strategy [5] is applied to select only a subset of test cases to (re)run. Two problems have to be addressed in this

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 157–168, 2007.

strategy: (1) the problem of selecting impacted tests from the test suite of original program and (2) the problem of determining where additional test cases may be required and generating them. A lot of work has been done around the first problem [7, 1, 4, 9, 3], which is known as regression test selection problem. An earlier work proposed a test path comparison approach to select the impacted test cases [1]. In this approach, the test paths of the original and the new programs are compared one by one, test paths not included in the new paths are selected out as impacted paths, and all test cases attached with the impacted paths should be rerun. This path comparison approach could only be used when white box test paths exist. Whereas, in real projects, test cases may be black box and are no associated white box test paths, and the test paths are not always generated. So more generally and widely used approaches are based on the comparison of control flow graphs or source codes [7, 4, 9]. These approaches are based on the assumption that when a node/edge on the control flow or a statement of the source code is changed, only the test cases that could cover this node/edge or statement will be impacted. This assumption is true for programs without concurrent control flow. However, in the presence of concurrent control flow, even a minor change to the synchronization condition may affect many concurrent execution paths that don't contain the changed condition. Furthermore, the impact is not only related with the changed synchronization condition, but also with other synchronization conditions. So, in the presence of concurrent control flow, the traditional work [7, 4, 9] will fail and cannot be applied directly. While in BPEL, the concurrent control flow is used as an important feature and is widely used in business processes.

In previous work [6, 8], the authors have implemented a test path generation tool for a BPEL business process under the path coverage criteria. This tool can generate test paths automatically, and the generated test paths need to be further refined manually into runnable test cases by adding complete test data and so on. The application of this tool in real cases leads to the requirement on the regression testing of a business process under the path coverage criteria. To meet such requirement, in this paper we propose a regression testing approach to select the (re)run test cases for a modified business process. Suppose that we have two sets of test paths, one for the old process, one for the new process, and an old test case set. Our goal is to work out a new test case set and select a test case subset to (re)run. All test paths can be classified into four categories with regression test selection technique:

- **reusable paths:** not impacted by the process changes, representing common test paths of the old and new processes;
- **modified paths:** impacted, in the sense that conditional branches on the test paths are the same but activity attributes or non-conditional activities are somehow changed, representing old and new test paths that have minor differences;
- **obsolete paths:** only valid for the old process, representing old test paths that will not exist for the new process;
- **new-structural paths:** only valid for the new process, representing new test paths having no correspondence in the old process.

Once we have this classification performed on the old and the new test paths, we can take actions to get the new test cases. Old test cases of the reusable paths are added to the new test case set; old test cases of the modified paths are updated into new test cases; and new-structural test cases are derived from the new-structural paths. Then all modified and new-structural test cases are selected to (re)run. In order to do the test path classification, in this paper, the differences between the old and the new processes are firstly identified. An impact analysis rule is proposed to analyze the affected test paths by the changes of concurrent control structures. Based on the process change information and impact analysis result, a test path selection algorithm is used to select the test paths, (re)run test cases. Different with previous work [1], we select the impacted test paths based on the business processes comparison instead of path comparison. Therefore, even if the test paths of the business processes are not generated, our approach could still be applied to select the impacted test cases via the linkage of source codes with test cases. The rest of this paper is organized as follows: section 2 gives some background of BPEL. Section 3 introduces our regression testing approach for business processes. Section 4 presents related work and section 5 closes this paper with conclusion and future work prediction.

2 Business Process Execution Language

A BPEL process is composed of BPEL activities, which can be either atomic or structured. Atomic activities define constructs for web service interactions and data handling, such as receive (wait to receive an event), reply (return an response to its caller), assign (assign a value to a variable). Like other programming languages, BPEL has typical control structures including sequence, switch, while, etc. In addition, BPEL uses the flow construct to provide concurrency and synchronization. These are called structured activities, which will be the container for atomic activities. A BPEL activity could have some attributes like name, invoked service and variable name.

Inside a flow construct, synchronization between concurrent activities is provided by means of links. Each link has a source activity and a target activity. Furthermore, a transition condition (a boolean expression) is associated with each link and is evaluated when the source activity terminates. Only after the source activity has terminated, the transition condition is evaluated. And only if the transition condition is true, the target activity could be executed. In this paper, activities that allow control logic divergence (e.g. switch, link with a transition condition) are called decision points. As the link activity has special meaning for regression analysis, we'll take it as a special kind of activity, and call all the other activities "general activities".

Figure 1 gives an example loan process. This process begins by receiving a loan request. The *InvokeAssessor* and *InvokeApprover* are two invoke activities to invoke risk assessment and loan approval services respectively. All the activities of this process are contained within a flow, and their (potentially concurrent) behavior is staged according to the dependencies expressed by links. The

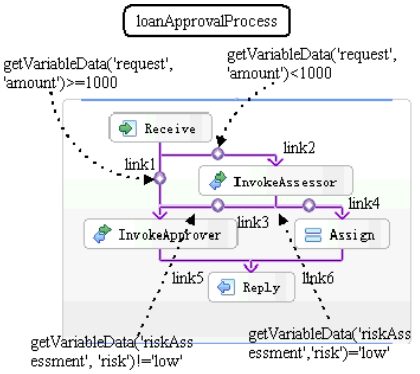

Fig. 1. The Loan Process

transition conditions on the links determine which links get activated. Finally the process responds with either a "loan approved" or a "loan rejected" message.

3 BPEL Regression Testing

The objective of regression test selection of a business process is to identify the impact of business process changes to test cases, and then take proper test case update actions accordingly and determine the subset of test cases to (re)run.

In this paper, we classify the test paths for the original process and the new process into four categories: reusable, obsolete, modified and new-structural, as is shown in figure 2. The exact meaning of this classification has been explained in the introduction section. Take the processes in figure 2 as an example, for the three test paths in the old process, path 1 is reusable as it is not impacted by the process changes, path 2 is obsolete, path 3 is modified into a new test path by adding a new activity 10 and modifying the activity 6. The new process introduces two new test paths: path 4 and path 5.

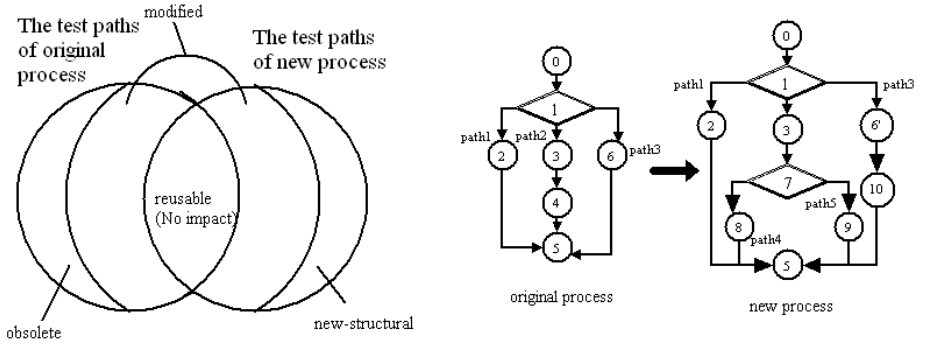

Fig. 2. The test paths classification and process change scenario

3.1 Regression Test Selection Problems Introduced by Concurrent Control Flow

Based on the path classification, the mission of path selection is to identify the reusable, obsolete, new-structural and modified test paths after a business process has been changed. For different types of activities, the change impact to test paths is different. For a general activity change, only test paths containing this activity will be impacted. According to the activity type and change information, the category(modified, obsolete, and new-structural) of impacted test paths could be determined. For example, if a while activity is deleted, all test paths containing this activity in the original path set will become obsolete. However, for a link activity, the problem becomes complex and this simple rule is not true any more. Once a link activity is changed, the impacted test paths will not be limited to those containing the changed link. For example, if the transition condition of link2 in the process of figure 1 is modified to the condition showed in the process of figure 3(1), the path with request.amount >= 1000 not containing link2 in figure 1 will be modified(the condition of this path becomes request.amount >= 2000). At the same time, two new-structural test paths are introduced into the new process(the path with 1000 <= request.amount < 2000 and riskAssessment.risk != 'low' and the path with 1000 <= request.amount < 2000 and riskAssessment.risk = 'low'). Whereas, if the transition condition of link2 is changed to request.amount >= 1000, as showed in the process in figure 3(2), all the test paths in the original process will become obsolete, and two new-structural paths will be generated in the new process.

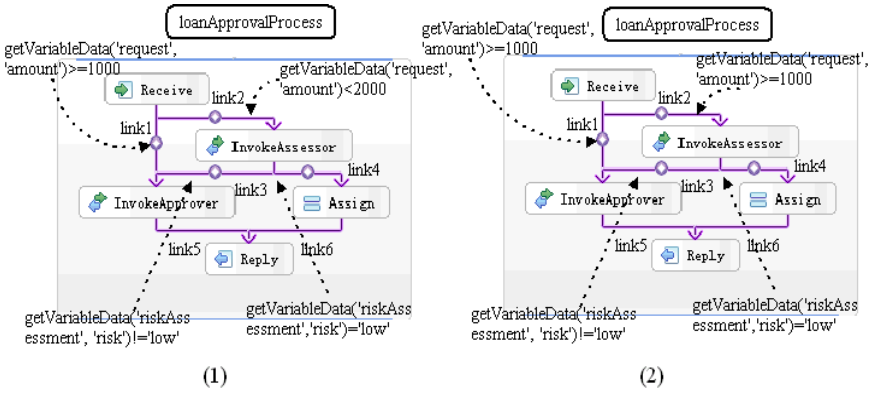

Fig. 3. The modified processes of loan process

Actually, the problems here are caused by the concurrent characteristic of the flow activity. Potentially, all the elements contained in a flow could execute concurrently. Just for the existence of link activities, some activities are prohibited to run, consequently not included in some test paths. For the test path with request.amount >= 1000 in figure 1, as the condition of link 2 is false, both

InvokeAssessor and *Assign* activity are prohibited to be executed, not including in this test path. Further more, the activation of link 2 is related with link 1, consequently the change of link 1 could affect both the test paths covering link 1 and those covering link 2. So, in order to select the modified, obsolete and new-structural paths impacted by a link activity change, we have to solve two problems: 1, analysis the impact of the link activity changes to test paths; 2, select the relevant test paths(modified, obsolete, new-structural paths) from the impacted test paths.

3.2 BPEL Diff

In order to select the impacted test paths, we need firstly identify the changes of a business process upon modification, and by the inclusion relationship between activities and test paths, the impacted test paths could be selected and classified. A change table is used to record all the changed activities here, as is shown in table 1. Therein, each row represents a change item. *IsDecisionPoint* indicates whether the changed activity is a decision point. *ChangeType* indicates the type of the change action: *M*, *D*, and *A*. *M* is modification action, *D* is deletion action, *A* is addition action. *Activities in old process* refers to the changed activity in the original process, and *Activities in new process* refers to the changed activity in the new process. For the deletion action, because the deleted activity does not exist in the new process, the previous activity of the deleted activity is put in *Activities in new process* column. Similarly, for the addition action, *Activities in old process* will point to the previous activity of the added activity. Such as for the process in figure 2, we could get the change table showed as table 1.

Table 1. The structure of change table

Activities in old process	**IsDecisionPoint**	**Change Type**	**Activities in new process**
4	false	D	3
6	false	M	6’
3	true	A	7

As a BPEL process is represented in XML format, we could use an XML parser to get a model that contains all the activities and their structure information of this process. In the model of original process and that of new process, in order to identify the same activity, the activity name is used as the unique identifier in this paper. In the BPEL process model coming from two processes, by comparing the activities in the original and the new process, we could get all the change information, and fill them into the change table. *Link* activity is special as it connects source and target activities. We use the following rules for the comparison of *link* activities. Only when the source activity name, the target activity name and the transition condition of two *links* are all the same, the *link* activities are considered as same. If either the source or the target activities are

different, the *link* activities are considered as different entities (this case will be broken down into a *link* delete action and a *link* addition action); if the transition condition is modified, this *link* is considered as modified.

3.3 Path Selection

For the new business process, based on our previous work [8], its test paths can also be generated automatically. So, in this paper, our test path selection algorithm is applied on these two test path sets to classify the reusable, modified, obsolete and new-structural paths. In order to record the relationship between test paths and activities, a test path table is used, as is shown in table 2. Therein, the value in column j and row i represents whether activity j is on test path i. When this value is 1, the activity j is on the test path i; when the value is 0, the activity j is not on the test path i. We call this value as indicator in this paper. Table 2 is the test path table of the loan process in figure 1.

Table 2. The test path table of the loan process

Test Path	Receive	Link1	Link2	Invoke Assessor	Link3	Link4	Invoke Approver	Assign	Link5	Link6	Reply
path 1	1	1	0	0	0	0	1	0	1	0	1
path 2	1	0	1	1	1	0	1	0	1	0	1
path 3	1	0	1	1	0	1	0	1	0	1	1

Impact Analysis for Concurrent Control Structure Change. From the change table, we could get all the change information of a business process, and for the different types of activities, the impact to test paths is definitely different. For a decision point, its deletion will cause all test paths passing this activity in original test path set become obsolete, and at the same time introduces new test paths that could cover the previous activity of this decision point into new path set. The addition of a decision point will generate new paths covering this activity in the new path set, and at the same time could make all test paths containing its previous activity in the original test path set become obsolete paths. For a branch activity of a decision point, its deletion could also make the test paths passing it in the original path set become obsolete, and its addition could generate new-structural paths passing it in the new path set. For a non-decision-point activity, its addition, deletion and modification could only make all test paths passing it become modified paths.

For the change of a *link* activity, the test paths covering the changed *link* activity are only a subset of the impacted test paths. Just as explained in section 3.1, its impact to the test paths is far beyond this. In fact, based on the characteristic of the target activity of a changed *link* activity, there are two types of impact results. One is that the target activity of a changed *link* activity is a start activity of a *flow* activity in the opposite process (we say the original and new process as opposite process), we call this change as first type of *link* change;

the other type is that the target activity is not a start activity, and we call this change as second type of *link* change.

First type of link change: in this type, because in current process, the target activity of the changed *link* activity is a start activity of a *flow* in the opposite process, all test paths passing this flow activity will include the activity. While in current process, for the existence of the *link* activity, only some test paths contain the target activity and it can not be a start activity in the flow activity. So all test paths passing this *flow* in the original and the new process will be impacted by the change. See the example in figure 4, which is another changed process of the loan process. In this process, *link 4* is deleted from the original process, and *Assign* activity becomes the start activity of the *flow* activity (although in semantics, this cannot happen for this process, here we just use it as an example). It could be seen that all the test paths passing the flow activity in the new process have the *Assign* activity as a start activity of this *flow*. While in the original process, no matter for which path, *Assign* activity is not a start activity. That is to say, no matter in the original or the new process, all the paths passing the *flow* activity are impacted.

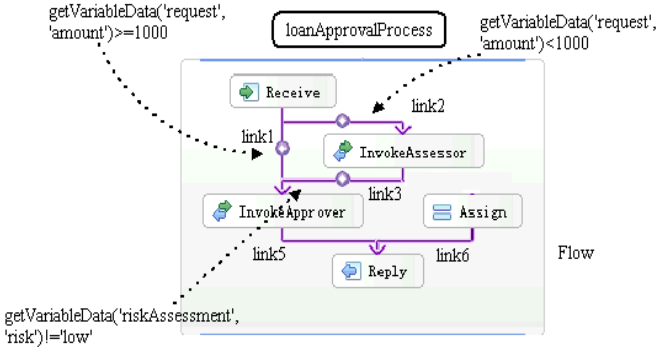

Fig. 4. Another changed process of the loan Process

Second type of link change: in this type, suppose in the original process, the *links* using the source activity of the changed *link* as source activity are link_1, link_2, ... , link_n, and the corresponding transition conditions are C_1, C_2, ... , C_n, the intersections they generate could be expressed as $C_1 \bigcap C_2 \bigcap ... \bigcap C_n$, it is obviously that each branch of this source activity is a region of this expression. For a condition C_i, if it has no intersection with the other conditions $C_j(i!=j)$, its deletion or addition will not impact other regions of this expression, but only reduce or increase one region. See the process showed in figure 1, the transition conditions of link3 and link4 have no intersection, so the deletion of link3 could only reduce the branches of activity *InvokeAssessor* by one. In this case, only the test paths containing the changed *link* activity are impacted. At the other extremity, if C_i has intersection with all the other conditions $C_j(i!=j)$, its deletion or addition definitely can impact all other intersections, such as C_i == true. In

this case, all the test paths passing the source activity will be impacted. In general cases, when the *link* transition condition is changed, the impacted test paths will have a scope between the above two extremity cases. In this paper, for this change type, we consider all the test paths covering the source activity as impacted paths.

So, for the changes of *link* activities, an impact analysis rule could be applied to select out all the impacted test paths. This rule can be described as follows. Firstly, based on the information of a changed *link* activity, judge which change type it belongs. Secondly, following the conclusion of relevant impact analysis, select out the impacted test paths caused by the change of this *link* activity. By this rule, we could solve the first problem introduced by concurrent control flow.

Path Selection Based on Test Path Table. Based on the impact analysis, we could get the regression test selection process as follows. Firstly, based on the change table, the non-impacted test paths could be selected as reusable paths. Secondly, the obsolete and new-structural paths caused by general activity changes could be selected out from the original and the new path set. Thirdly, the impacted test paths caused by *link* activity changes could be selected respectively from the original and the new path set. Finally, the remaining paths is modified paths. In this paper, we call the test paths impacted by link activity changes **special path sets**.

In the special path sets, for two test paths p and $p^{'}$ that come from the original path set and the new path set respectively, if $p^{'}$ is modified from p, they must execute the same *link* activities. If in the original path set, there is no test path that could execute the same *link* activities with $p^{'}$, $p^{'}$ must belongs to new-structural path set. Oppositely, if there is no test path that could execute the same *link* activities with p in the new path set, p must belong to obsolete path set. So, for a test path $p^{'}$ in the special path set coming from new path set, following the rule that whether there is a path p in the original path set has the same *link* activities with it, we could decide $p^{'}$ as a modified path or new-structural path. If $p^{'}$ is a modified path, p should also be a modified path. Finally, all the remaining paths in the special path set coming from the original path set are obsolete paths. Consequently, the test paths impacted by the changes of *link* activities could be classified into modified, obsolete, and new-structural paths, solving the second problem introduced by concurrent control flow.

In summary, the path selection algorithm is shown as below. In this algorithm, each path is labeled with a symbol from (*R, M, O, N, R*). In the symbol for path p_i, R represents p_i is reusable, M represents p_i is modified, O represents p_i is obsolete, N represents p_i is new-structural, and S represents p_i belongs special path sets.

PathSelection(Change Table: C, Test Path Table of Original Process: T, Test Path Table of new Process: $T^{'}$)
1 Label all paths in original and new path set as R
2 **for** each modified activity a in C **do**
3 get impacted test paths I and I' from T and $T^{'}$

4 **if** a is a decision point or a branch activity of a decision point
5 label the paths in I as O and the paths I' as N
6 **else if** a is not a link activity
7 for the paths in I and I' are labeled with R, label them as M
8 **else**
9 **if** the change type of a is addition or deletion
10 label all paths containing a in I as O and those in I' as N
11 **end if**
12 label other impacted test paths being labeled with R or M in I and I' as S
13 **end if**
14 **end if**
15 **end for**
16 classify the test paths are labeled with S into modified, obsolete and new-structural paths.

Table 3. The test path table of modified loan process

Test Path	Receive	Link1	Link2	Invoke Assessor	Link3	Link4	Invoke Approver	Assign	Link5	Link6	Reply
path 1	1	1	1	1	1	0	1	0	1	0	1
path 2	1	1	1	1	0	1	1	1	0	1	1

In the identification of the special test path sets, because the paths executing the added or deleted *link* activities have been selected out(shown in the 9 and 10 lines of **PathSelection** algorithm), in the special test path sets, all the *link* activities contained in the paths are modified or non-changed, which exist in original and new process at the same time. We represent this *link* activities set as S_{link}. In the original path table, for a test path in the special test path set, from top to down, all the indicator values of *link* activities contained in S_{link} could form a 0 and 1 string, which in deed indicates the *link* activities that a test path contains. In the new path table, from top to down, by tuning the order of *link* activities contained in S_{link} to keep the same order with that of original path table, for a test path in the special test path set of new path set, all the indicator values of *link* activities contained in S_{link} also could form a 0 and 1 string. By judging whether this string is contained in the 0 and 1 string set of original path set, we could select this path into relevant path set. Finally, all the remaining paths in the special path set in the original path set are obsolete paths. Such as for the original process in figure 1 and the new process in figure 3(2), table 3 shows the test paths of the process in figure 3(2). By impact analysis, we could learn that all test paths in test path table 2 and 3 are special paths. Firstly, we could get the 0 and 1 strings for the special path set of original path set as {path 1: 100010, path 2: 011010, path 3: 010101}(from left to right, the link activities are: link1, link2, link3, link4, link5, link6). The 0 and 1 strings for the special path set of this new process are: {path 1:111010, path 2: 110101}.

Because 111010 and 110101 are neither contained in {100010, 011010, 010101}, the test paths of table 3 are new-structural paths, and the remaining paths in table 2 are obsolete.

After the test paths are selected, the regression testing actions are taken as follows: the test cases of reusable paths are added to new test case set, those of modified paths are updated, and new test cases are generated for the new-structural paths. Then the updated and new test cases are selected to (re)run.

The tool of this paper is built as an Eclipse plugin tool, making it could easily integrated with other SOA develop or testing tool, such as WebSphere Integration Developer or Rational Architectural Develop.

4 Related Work

Regression testing has been recognized as an effective technology to ensure the quality of software after a system has been changed. Lots of previous work has been done around the regression test selection problem, and the test selection based on control flow is widely applied [7, 4, 9]. In these approaches, after a program has been changed, the control flow graphs of the original and the new program are obtained by analyzing the source codes of original and new program. Based on the control flow graphs, a graph comparison algorithm is used to identify the changed nodes or edges. Consequently, the test cases covering the changed nodes or edges are selected as impacted cases. In the face of the new characteristic introduced by the object-oriented programs and aspect-oriented programs, the control flow graph is extended by [4, 9] respectively to support the new characteristic of new programming language. Then based on the extended control flow graph, the test selection algorithm is applied to select the impacted test cases [4, 9]. So far as we know, [1] is the only work that also selects the impacted test cases under the path coverage criteria. In this paper [1], all test paths are represented by a special expression-algebraic expression. Based on the representation, a test path comparison approach was proposed to select the impacted test cases. However, this approach is limited by the expression capability of test path model, and only could be applied to selected the impacted test cases when the test paths are generated. While the approach of our paper not only could be applied to the select the (re)run test cases via test paths, but also could be applied to select the (re)run test cases when test paths are not generated.

5 Conclusions and Future Work

Service Oriented Architecture (SOA) is recognized as a good solution for the integration of diverse IT systems. BPEL as a standard for the service orchestration has been widely used in SOA to compose multiple services to accomplish a business process. The pivotal role of BPEL in a SOA system makes its reliability become especially critical in the maintenance of a SOA system. Regression testing has been recognized as an effective technology to ensure the quality of

modified programs. In this paper, to address the special problems introduced by the concurrent control structure of BPEL, a regression test selection approach for BPEL is proposed. In this approach, the changed activities are identified by BPEL Diff. The impact of concurrent control structure changes to test paths is classified into two types. Based on the analysis for these two types, an impacted analysis rule for the changes of concurrent control flow is proposed. By considering all the *link* activities on a test path, the test paths impacted by a *link* activity changes are classified into modified, obsolete and new-structural paths. Consequently, the reusable, modified, obsolete and new-structural path sets are selected out. Finally the related test cases are selected and updated. In future, we will study a more precise selection algorithm to select the test paths impacted by *link* activity changes, and further validate our technology in more real cases.

References

[1] Benedusi, P., Cimitile, A., Carlini, U.D.: Post-maintenance testing based on path change analysis. In: ICSM' 88. Proceedings of the Conference on Software Maintenance, Scottsdale, AZ, USA, pp. 352–361 (October 1988)
[2] Chen, L., Wassermann, B., Emmerich, W., Foster, H.: Web service orchestration with bpel. In: ICSE'06. Proceeding of the 28th International Conference on Software Engineering, Shanghai, China, pp. 1071–1072 (May 2006)
[3] Vokolos, F.I., Frankl, P.G.: Pythia: A regression test selection tool based on textual differencing. In: ENCRESS' 97. Porceedings of the 3th International Conference on Reliability, Quality, and Safety of Software Intensive Systems, Athens, Greece, pp. 3–21 (May 1997)
[4] Harrold, M.J., Jones, J.A., Li, T., Liang, D.: Regression test selection for java software. In: OOPSLA'01. Proceedings of the ACM Conference on OO Programming, Systems, Languages, and Applications, Tampa Bay, FL, USA, pp. 312–326. ACM Press, New York (October 2001)
[5] Li, Y., Wahl, N.J.: An overview of regression testing. ACM SIGSOFT Software Engineering Notes 24(1), 69–73 (1999)
[6] Li, Z., Sun, W.: Bpel-unit: Junit for bpel processes. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 415–426. Springer, Heidelberg (December 2006)
[7] Rothermel, G., Harrold, M.J.: A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology 6(2), 173–210 (1997)
[8] Yuan, Y., Li, Z., Sun, W.: A graph-search based approach to bpel4ws test generation. In: ICSEA'06. Proceedings of the International Conference on Software Engineering Advances, Papeete, Tahiti, French Polynesia, p. 14 (October 2006)
[9] Zhao, J., Xie, T., Li, N.: Towards regression test selection for aspectj programs. In: WTAOP06. Proceedings of the 2nd workshop on Testing Aspect-Oriented Programs, Portland, Maine, pp. 21–26 (July 2006)