Metasearch Services Composition in WS-BPEL
An Application of Metamorphic Testing

M. del Carmen de Castro-Cabrera, Inmaculada Medina-Bulo and Azahara Camacho-Magriñán
Department of Computing Languages and System, University of Cádiz, Cádiz, Spain

Keywords: Metamorphic Testing, Oracle, Testing Cases, Web Services Compositions, WS-BPEL.

Abstract: Nowadays, the impact of Web Services is quickly increasing because of transactions through Internet. The OASIS WS-BPEL 2.0 standard language allows to develop business processes by means of pre-existing Web Services and to offer themselves as a new Web Service. This makes it necessary to pay special attention to testing this type of software and presents a challenge for traditional testing techniques, due to the inclusion of specific instructions for concurrency, fault and compensation handling, and dynamic service discovery and invocation. Metamorphic Testing has proved useful to test and improve the quality of traditional imperative programs. However, it has not been applied to languages for composing Web Services such a WS-BPEL. This work presents an procedure for applying Metamorphic Testing to Web Services compositions, proposes an architecture and analyzes a case study with promising results.

1 INTRODUCTION

Web Services Business Processes language, WS-BPEL 2.0 (OASIS, 2007) was standardized at the request of some TIC companies (HP, IBM, Oracle, Microsoft, and so on.). This language allows us to develop new Web Service (WS) designing more complex business processes from pre-existing WS, and there is a widely support software for them. However its development has not gone along with improve on testing techniques to this type of software (Bozkurt et al., 2010). A deficient testing in a system could cause errors with negative consequences both economical (IDC, 2008) and also humans (Leveson, 1995). Consequently, good testing methods to test compositions for correctness are required. Progress in this sense are described in (Palomo-Duarte, 2011). Furthermore, in (García-Domínguez et al., 2011) two inference algorithms to reduce application cost of a extended methodology based on SOA (with UML and MARTE) are presented.

Metamorphic testing (MT) (Chen, 1998) is a software testing technique that allows to generate test cases automatically to prove programs. It is based on *Metamorphic Relations* (MRs), that is, existing or hope properties defined over inputs and its corresponding output set, to a program under testing. Indeed, that technique has been proved with success in programs written in imperative programming languages (Zhou et al., 2004). As for the efficiency, Zhang in (Zhang et al., 2009) has conducted an experiment in which compares the ability of detecting error and time cost between MT and standard assertion checking. The results reveal that MT detects more faults than standard assertion checking.

This work presents an procedure for applying Metamorphic Testing to compositions with WS-BPEL and proposes an architecture. Furthermore, a case study with encouraging results is added.

This paper is structured as follows. In the section 2 fundamentals are explained. Section 3 presents a general procedure for implementing MT. Section 4 exhibits a general architecture with the description of steps to follow. Afterward, in the section 5 a case study, the Metasearch composition, is described with the MRs and the results obtained. Section 6 shows some works related to testing compositions in WS-BPEL and MT technique applications. Finally, in the section 7 are conclusions and future works.

2 FUNDAMENTALS

Software Testing is a core activity in any software development project. Detecting and correcting errors are critical operations to ensure program reliability. For this reason, diverse techniques have been developed based on different approaches, even though,

Castro C., Medina-Bulo I. and Camacho A..
Metasearch Services Composition in WS-BPEL - An Application of Metamorphic Testing.
DOI: 10.5220/0004026004270432
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 427-432
ISBN: 978-989-8565-19-8

none has proved to be better than others (Beizer, 1990; Myers, G.J. et al., 2004).

2.1 Metamorphic Testing (MT)

MT relies on the notion of MR. In (Andrews et al., 2005), MRs are defined as "existing properties over a different inputs set and its corresponding results to multiple evaluations of a target function". In addition, The use of MT to alleviate oracle problem (Chen, 1998; Chen, 2010).

When the implementation is correct, program inputs and outputs are expected to satisfy some necessary properties that are relevant for underlying algorithms. These properties are traditionally known as *assertions*. In this sense, a MR is a type of assertion that is expressed according to the test case values.

However, a MR should provide a way of generating new test cases from given ones previously. In order to illustrate this, let us consider a sorting program, *sort*, that sorts an input list of integers producing a list with the same integers in ascending order.

For instance, when we have as input the list $l_1 = \langle 4,7,2 \rangle$, the expected result of $sort(l_1)$ is $\langle 2,4,7 \rangle$. Consequently, if a function *perm* that permutes elements from a given list to generate a new test case is deployed, we claim that its result must be the same (the condition over the output is the equal function).

Proceeding with the same example, $l_2 = perm(l_1, \langle (1,2) \rangle) = \langle 7,4,2 \rangle$ (there is only a swap between the first and the second elements), the expected result of $sort(l_2)$ must be the same than $sort(l_1)$. In other words, $sort(l_1) = sort(l_2)$. We could formalize this property, MR_1, as follows:

$$MR_1 \equiv (\exists x \; l_2 = perm(l_1, x)) \rightarrow sort(l_2) = sort(l_1)$$

where l_1 is the initial input and l_2 will be the *follow-up test case*. Summing up, MT is a testing technique using MRs (Chan et al., 2007). It begins with an initial test suite, which is produced with any test case selection strategy, and a set of MRs. Once the program is executed on the test suite, errors are fixed until a *successful test suite* (i.e., consisting of non-failing test cases) is obtained. Then, MRs are used to generate follow-up test cases. These test cases form the *follow-up test suite* and the process is iterated until the specified test criteria are met.

3 MT IMPLEMENTATION

Test case generation can be automated as in traditional techniques. Following with the example of the previous section, replacing the second parameter of *perm* by a random permutation we would obtain the equivalent to traditional random testing for this example.

Of course, a single MR is generally insufficient. In the above example, we could not detect certain faults just with MR_1, as correctness for sorting implies permutation preservation (the result list must be a permutation of the original) and an order constraint (it must be ordered).

Then, MT is a testing technique that begins with an initial test suite, which is produced with any test case selection strategy, and a set of MRs. Once the program is executed on the test suite, we obtain that some test cases detect errors and others not. The first ones force that the program is revised and the errors corrected and, the second ones, named *successful test cases*, will be selected as input to our architecture.

Thereby, the MRs generate the *follow-up test cases* from that successful set. These follow-up test cases will compound the (initial) test suite of the following iteration and the procedure will be repeat.

Therefore, once a initial test suite has been generated (with a generation strategy), we have to follow the next steps to apply successfully MT. First, choose the successful test cases. These constitute the *initial test suite*. Second, select adequate MRs taking into account the problem to solve and the algorithm structure implemented in the program. Third, generate the follow-up test suite applying MRs to initial test suite. Fourth, execute the program with initial and follow-up test cases. Fifth, compare the results. And, finally, improve the program correcting the detected errors, select new test cases and/or new MRs enhanced to successive iterations. You can see more information in (Castro-Cabrera and Medina-Bulo, 2011).

4 PROPOSED ARCHITECTURE

Once diverse aspects related with MT have been analyzed, in this section an architecture for testing web service compositions in WS-BPEL is presented.

The proposed architecture uses open-code systems: ActiveBPEL (ActiveVOS, 2009) as the BPEL execution engine and BPELUnit (Mayer and Lübke, 2006) as the unity testing library.

This approach intends to improve test suite, from MRs, generating new test cases, which allow us to detect a greater number of errors in compositions. Next every steps of the proposed architecture are described:

1. *Analysis and Property Obtainment Step*. This step analyzes the composition to obtain relevant properties and to specify MRs. One of the possibilities is to deploy the free framework Takuan (Palomo-Duarte et al., 2010), that detects properties in a

BPEL composition from test cases. Furthermore, in this step, follow-up test cases are generated through MRs. This phase receives as inputs: a WS-BPEL composition and a test suite randomize generated or through an automatic application. As result and as requirement to the next step, the follow-up test suite (generated from MRs) and the ActiveBPEL specific files to execute the composition are obtained. The main modules are an analyzer and a follow-up test cases generator.

2. *Execution Step*. In this step the composition is executed with initial and follow-up test cases. Every test case encloses an initial message that triggers a WS-BPEL process instance in the server, as well as the responses that the external services will provide. As input specific files to ActiveBPEL engine are received and initial and follow-up test cases obtained from the previous step. As output the results from test cases execution are obtained. The principal modules are a WS-BPEL engine and a WS-BPEL unity testing library.
3. *Comparison Step*. This step receives the results of test cases (initial and follow-up) execution. It compares the results of initial and follow-up test cases execution, with the target of detecting errors in composition. Essentially, it is checking if MRs hold, since these MRs are necessary properties. In the case that another MR was not satisfied, an error in the composition has been detected. The comparison results allow to decide the number of process iterations.

5 A CASE STUDY

In this section we apply the proposed architecture to the MetaSearch composition defined in (OASIS, 2007). Its behavior is in the figure 1. This consists of the search done by a client in Google search and MSN search. Each browser returns its results, first all the results of Google and then all the results of MSN. If there are no results of Google, the client will only see the results of MSN, and vice versa.

The first phase consists of analyzing the composition and the original test cases to design suitable MRs: study the original test cases and design and implement adequate MRs to these test cases. We apply one of these MRs to one of the original test cases and we obtain a new test case. It should be noted that the test cases extend to the responses of services, that is, we have to consider not just inputs and outputs, but also the responses of the involved services.

Now, we show the most important elements involve in the test case that we study: *Qury*, keywords of the client's query; *Lang*, language that the client uses for searching; *C*, country where the search is done; *Cult*, string formed by the concatenation between *Lang* and *C* (if one of the elements is null (''), the value of *Cult* will be 'en-US'); *Max_Res*, maximum number of results that the client wants; *N*, total number of results that the client obtains with both browsers - this number has to be less or equal than Max_Res ($N \Leftarrow Max\_Res$); *Google_Res*, this represents all of the results of Google; and, finally, *MSN_Res*, this represents all of the results of MSN. There are other elements that is not relevant to the study of this paper.

The structure of a theorical test case is this: (*Qury, Lang, C, M_Res, N, Google_Res, MSN_Res*)

- One of the original test cases that we have studied is the following: *Cult = de-De*

  (Philipp, de, DE, 3, 3, *Google_Res*, *MSN_Res*)

  We do not consider the Google results and the MSN results because the MRs only use the *Lang* and *C* elements. This test case is a valid one because the client indicates a query to search, a maximum number of results and finally, the client obtains a total result of the browsers that is less or equal than this maximum.

  We apply the following MR to the previous test case where the elements that end with '1' are the original elements, and the elements that end with '2' are the new elements:

  Precondition: $Lang1 \neq$ "" **MR1**:
  $Lang2 \neq Lang1 \wedge\ Lang2 =$ "" $\Rightarrow\ Cult2 \neq Cult1$

  The precondition of this MR indicates that the original Language is no null. And as the original test case satisfies this precondition, we obtain the new follow-up test case: *Cult = en-US*

  (Philipp, , DE, 3, 3, *Google_Res*, *MSN_Res*)

  The new test case is valid because satisfies all the requirements of the composition that we have commented before.

  Another original test case that we have studied is the following: *Cult = de-De*

  (Philipp, de, DE, 4, 4, *Google_Res, MSN_Res*)

  The structure is similar as the previous original test case, but in this case the client wants to obtain one more result.

  We apply this MR to the test case:

  Precondition: $C1 \neq$ ""
  MR2: $C2 \neq C1 \wedge\ C2 =$ "" $\Rightarrow\ Cult2 \neq Cult1$

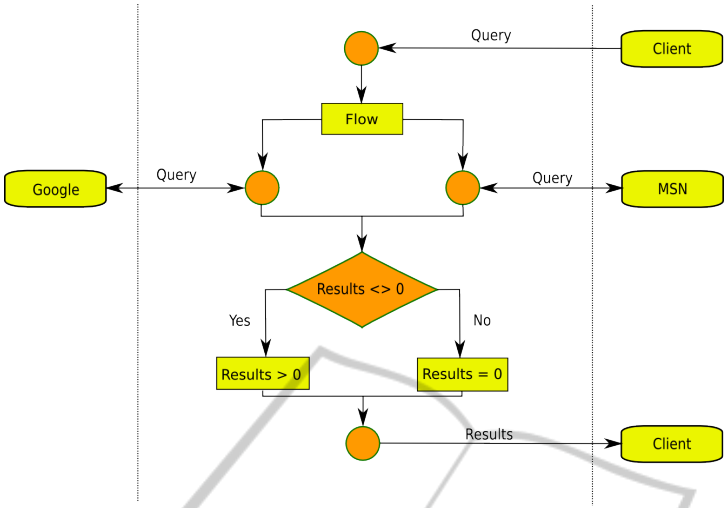

Figure 1: The MetaSearch composition.

The precondition of this MR indicates that the original Country is no null. And as the original test case satisfies this precondition, we obtain the new follow-up test case: *Cult = en-US*

(Philipp, de, , 4, 4, *Google_Res, MSN_Res*)

The new test case is valid because satisfies all the requirements of the composition.

And one last original test case that we have studied is the following: *Cult = en-De*

(Philipp, en, DE, 3, 3, *Google_Res, MSN_Res*)

The structure is the same as the first original test case, but in this case the *Lang* is English. Therefore, the value of the *Cult* element is 'en-De'.

We apply again the first MR for this test case because it satisfies the precondition and we obtain the new follow-up test case: *Cult = en-US*

(Philipp, , DE, 3, 3, *Google_Res, MSN_Res*)

It is the same test case that we obtained in the first example. Therefore, we can see that we not only can obtain new test cases with these MRs, but also we can obtain repeated test cases.

5.1 Follow-up Test Case Generation

We have implemented the previous MRs with others that we have also obtained of the complete study. These MRs allow us to obtain new test cases because they apply logical and numerical relations to the original test cases. This improves the initial test suite.

As we could see in the previous section, the application of MRs to certain test cases can obtain repeated test cases. To prevent this, we implemented an application that automates the generation of follow-up test cases from an original test suite, deletes the follow-up test cases that are repeated and writes all the follow-up test cases in a file with BPELUnit format.

But this effort is rewarded because the application generates new valid test cases. This saves us the hard work of generating follow-up test cases by hand. In fact, we could iterate this process if we run it to these follow-up test cases improving the test suite.

5.2 Result Evaluation

An overall result of this study is that we have improved the error detection technique completing the initial test suite and detecting new errors. A particular test case is the following, one of the test cases that we have studied in this section: *Cult = de-De*

(Philipp, de, DE, 3, 3, *Google_Res, MSN_Res*)

And the new test case generated by the MR1 is:

Cult = en-US
(Philipp, , DE, 3, 3, *Google_Res, MSN_Res*)

Based on the MetaSearch composition logic, the source code fragment that evaluates the final value of the *Cult* element is the following:

```
<bpel:condition>
   (($inputVariable.payload/client:country != '')
     and
   ($inputVariable.payload/client:language != ''))
  </bpel:condition>
```

As we explained at the beginning of this section, if the *Lang* and *C* elements are different to null (") the value of *Cult* will be the string formed by the concatenation between both elements. In other case, the value will be 'en-US'.

If there are an error in the source code of the com-

position, like replacing the relation 'and' by the relation 'or', the source code fragment of this error composition is the following:

```
<bpel:condition>
   (($inputVariable.payload/client:country != '')
     or
   ($inputVariable.payload/client:language != ''))
  </bpel:condition>
```

With the original test case we do not detect the invalid composition because it satisfies the condition too. But, if we use the new test case generated by the MR1, we will detect the invalid composition because it does not satisfy the condition.

In the valid composition the value of the *Cult* element of the new test case is 'en-US' because one of the elements (in this case, the *Lang* element) is equal to null. However, in the invalid composition the value of *Cult* is '-DE' because the test case satisfies the condition that one of the elements is different to null. The values are different therefore, the error is detected.

This new test case allows us to detect an error that the original test case did not detect. In addition to generating a significant number of follow-up test cases automatically, some of these test cases allow us to detect new errors and to improve the initial test suite. Besides, new MRs can be implemented that allow us to generate another test cases to improve the results.

6 RELATED WORKS

In this section, we describe some tools and techniques applied to test WS compositions. We focus in those ones implemented in WS-BPEL and most of them are referred to test case generation (García-Fanjul et al., 2007; Zheng et al., 2007). GAmera (UCASE Research Group, 2010) is based on mutation testing and that uses genetic algorithms.

The first documented work related with MT belongs to Weyuker (Weyuker, 1982). In this paper Weyuker proposed a new perspective for software testing based on using alternative programs to the original one that deploys the same function. This idea was later adopted by Chen in (Chen, 1998), where the name *metamorphic testing* firstly appears. Since then, numerous papers have been published with different visions as (Gotlieb and Botella, 2003), where a possible automation of this technique is presented.

Chen and his colleagues in (Chen et al., 2004) describe how the adequate selection of MRs is a critical issue in this technique. Alternatively, Chan and his collaborators have some works related with this topic, but the more interesting for our research is (Chan et al., 2007), in that applies MT to SOA, however it is only applied over services, not compositions. Recently, other interesting work has been published to obtain MRs and automatize test cases generation integrated into BeTTy (Segura et al., 2012).

A research group of Columbia University has implemented in (Murphy et al., 2008) a framework that automatize partly the testing process. Later, they have implemented *Corduroy*, an application to automatize the process (Murphy et al., 2009). Lately, advances in this research have been published in (Xie et al., 2011).

7 CONCLUSIONS

WS-BPEL business processes are considerably increasing in last years. For this reason, it is important the development of techniques that allow to test this type of software. Due to the language nature, it is necessary to adapt traditional testing techniques to test compositions implemented by those languages.

In addition, MT has been implemented in different languages efficiently and it is being actually under consideration by some research groups. Selection of adequate MRs is an important issue to this technique, so we ought to consider the problem context and the structure of the program under test.

A testing framework architecture to apply MT in WS-BPEL compositions has been proposed. The steps and design of this have been specified supported by open software systems. Further, a test case study over an specific composition, Metasearch, has been included. Its specification, design and implementation description, as well as results have also been presented. This indicates how it is possible to improve a test suite detecting errors in the composition.

As future work, the complete development of the architecture proposed is considered. Once the system was implemented, it could be compared these results with other obtained through different techniques using diverse compositions. However, unfortunately it does not exist up to now, any standardized testing suite to WS-BPEL (Palomo-Duarte, 2011), anyhow the results could be depending on the cases study.

ACKNOWLEDGEMENTS

This work is supported by the MoDSOA project (TIN2011-27242) under the National Program for Research, Development and Innovation of the Ministry of Science and Innovation (Spain) and by the PR2011-004 project of the University of Cádiz.

REFERENCES

ActiveVOS (2009). ActiveBPEL WS-BPEL engine. http://sourceforge.net/search/?q=ActiveBPEL.

Andrews, J. H., Briand, L. C., and Labiche, Y. (2005). Is mutation an appropriate tool for testing experiments? In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, pages 402–411. ACM Press.

Beizer, B. (1990). *Software Testing Techniques, 2nd Edition*. International Thomson Computer Press, 2 sub edition.

Bozkurt, M., Harman, M., and Hassoun, Y. (2010). TR-10-01: testing web services: A survey. Technical Report TR-10-01, King's College, London.

Castro-Cabrera, C. and Medina-Bulo, I. (2011). An approach to metamorphic testing for ws-bpel compositions. In *Proceedings of the International Conference on e-Business (ICE-B 2011)*.

Chan, W. K., Cheung, S. C., and Leung, K. R. (2007). A metamorphic testing approach for online testing of service-oriented software applications. *International Journal of Web Services Research*, 4(2):61–81.

Chen, T. Y. (1998). Metamorphic testing: A new approach for generating next test cases. *HKUSTCS98-01*.

Chen, T. Y. (2010). Metamorphic testing: A simple approach to alleviate the oracle problem. In *Proceedings of the 5th IEEE International Symposium on Service Oriented System Engineering*. IEEE Computer Society.

Chen, T. Y., Huang, D. H., Tse, T. H., and Zhou, Z. Q. (2004). Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004)*, pages 569–583.

García-Domínguez, A., Medina-Bulo, I., and Marcos-Bárcena, M. (2011). Model-driven design of performance requirements with UML and MARTE. In *Proceedings of the 6th International Conference on Software and Data Technologies*, Seville, Spain.

García-Fanjul, J., Tuya, J., and de la Riva, C. (2007). Generación sistemática de pruebas para composiciones de servicios utilizando criterios de suficiencia basados en transiciones. In *JISBD 2007: Actas de las XII Jornadas de Ingeniería del Software y Bases de Datos*.

Gotlieb, A. and Botella, B. (2003). Automated metamorphic testing. *Computer Software and Applications Conference, Annual International*, 0:34–40.

IDC (2008). Research reports. http://www.idc.com.

Leveson, N. G. (1995). *Safeware: system safety and computers*. ACM.

Mayer, P. and Lübke, D. (2006). Towards a BPEL unit testing framework. In *TAV-WEB'06: Proceedings of the 2006 workshop on Testing, Analysis, and Verification of Web Services and Applications*, pages 33–42. ACM.

Murphy, C., Kaiser, G., Hu, L., and Wu, L. (2008). Properties of machine learning applications for use in metamorphic testing. In *Proc. of the 20th international conference on software engineering and knowledge engineering (SEKE)*, pages 867–872.

Murphy, C., Shen, K., and Kaiser, G. (2009). Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles. In *Software Testing Verification and Validation, 2009. ICST'09. International Conference on*, pages 436–445.

Myers, G.J., Sandler, C., Badgett, T., and Thomas, T. M. (2004). *The Art of Software Testing, 2nd ed.* Wiley - Interscience.

OASIS (2007). Web Services Business Process Execution Language 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. Organization for the Advancement of Structured Information Standards.

Palomo-Duarte, M. (2011). Service composition verification and validation. In Jonathan Lee, S.-P. M. and Liu, A., editors, *Service Life Cycle Tools and Technologies: Methods, Trends and Advances*, pages 200–219. IGI Global.

Palomo-Duarte, M., García-Domínguez, A., Medina-Bulo, I., Álvarez-Ayllón, A., and Santacruz, J. (2010). Takuan: a tool for WS-BPEL composition testing using dynamic invariant generation. In et al., B. B., editor, *ICWE*, volume 6189 of *Lecture Notes in Computer Science*, pages 532–535. Springer.

Segura, S., Galindo, J., Benavides, D., Parejo, J., and Ruiz-Corts, A. (2012). Betty: Benchmarking and testing on the automated analysis of feature models. In Eisenecker, U., Apel, S., and Gnesi, S., editors, *Sixth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'12)*, pages 63–71, Leipzig, Germany. ACM.

UCASE Research Group (2010). GAmera home site. http://neptuno.uca.es/˜gamera.

Weyuker, E. (1982). On testing Non-Testable programs. *The Computer Journal*, 25(4):465–470.

Xie, X., Ho, J. W. K., Murphy, C., Kaiser, G., Xu, B., and Chen, T. Y. (2011). Testing and validating machine learning classifiers by metamorphic testing. *J. Syst. Softw.*, 84:544–558.

Zhang, Z.-Y., Chan, W. K., Tse, T. H., and Hu, P.-F. (2009). An experimental study to compare the use of metamorphic testing and assertion checking. *Journal of Software*, 20(10):2637–2654.

Zheng, Y., Zhou, J., and Krause, P. (2007). An automatic test case generation framework for web services. *Journal of software*, 2(3):64–77.

Zhou, Z. Q., Huang, D. H., Tse, T. H., Yang, Z., Huang, H., and Chen, T. Y. (2004). Metamorphic testing and its applications. In *Proceedings of the 8th International Symposium on Future Software Technology (ISFST 2004)*. Software Engineers Association.