AN APPROACH TO METAMORPHIC TESTING FOR WS-BPEL COMPOSITIONS

Carmen Castro-Cabrera and Inmaculada Medina-Bulo
Department of Computer Languages and Systems, University of Cádiz, Cádiz, Spain

Keywords: Testing, Metamorphic testing, Mutation analysis, Oracle, Follow-up test cases, Web services, WS-BPEL.

Abstract: Nowadays, Web Service (WS) compositions play an important role in business processes. Languages for composing web services, such as the OASIS WS-BPEL 2.0 standard, open a new field for large-scale programming. However, they also present a challenge for traditional quality assurance because of the inclusion of specific instructions for concurrency, fault compensation, and dynamic service discovery and invocation. Metamorphic Testing (MT) has proved useful to test and improve the quality of traditional imperative programs. This paper discusses how to use MT to test WS compositions in WS-BPEL. Although MT has not been previously applied to this area, promising results have been obtained in a number of different applications. A component diagram for a testing framework implementing this approach is included.

1 INTRODUCTION

Web services are having a strong impact in our society because of the increasing number of Internet transactions and the new ways of conceiving and using web applications dealing with them. The Web Services Business Process Execution Language (WS-BPEL), (OASIS, 2007) allows us to develop new Web Services (WS) by modeling more complex business processes on top of preexisting WS. This evolution of business processes has produced the development of specific software to satisfy this demand. However, this development has not come with an advance in testing for this kind of software. In addition, the economic impact of WS-BPEL service compositions has increased (IDC, 2008), and deeper insight on how to test them effectively is therefore required. We present a proposal to apply MT to WS-BPEL compositions.

MT (Chen, 1998) is a software testing technique using *metamorphic relations* (MR). MR are existing or expected relations defined on a set of inputs and their corresponding outputs for multiple executions of a function under test. The underlying concept is simple and its automation is not difficult. In fact, it has proved successful in testing and improving the quality of traditional imperative programs (Zhou et al., 2004). Regarding the cost effectiveness of MT, Zhang (Zhang et al., 2009) conducted an experiment where the fault detection capabilities and time cost of MT were compared to the standard assertion checking method. Results show that MT has the potential to detect more faults than the assertion checking method. In addition, Chan and his collaborators are investigating their applications to different areas. The most related to this paper are the application of MT to service-oriented software (Chan et al., 2006; Chan et al., 2007).

The structure of the rest of the paper is as follows: Section 2 outlines the main concepts about service compositions and MT. Section 3 presents some techniques that are being applied with success to testing WS compositions. In the following Section 4, some applications of MT to different areas are reviewed. Section 5 presents our proposal, its main steps and a component diagram for applying MT to WS-BPEL compositions. Finally, Section 6 presents some conclusions and future research lines.

2 PRELIMINARIES

Next, we introduce the main concepts about service compositions in WS-BPEL and MT.

2.1 Web Services Compositions

This paper focuses in WS compositions, specially when they are implemented using WS-BPEL. WS-BPEL is a programming language based in XML that is used to generate business processes from preexisting services. The resulting business process can be then reused as a WS in higher level compositions. A WS-BPEL composition contains four sections:

1. Declarations of the relationships to the external partners. These include both the client that has invoked the business process and the external partners whose services are required to complete the request of the client.
2. Declarations of the variables used by the process and their types. Variables are used for storing both the messages received and sent from the business process and the intermediate results required by the internal logic of the composition.
3. Declarations of handlers for various situations, such as fault, compensation or event handlers.
4. Description of the business process behavior.

The major building blocks in WS-BPEL are the *activities*. Activities may have both *attributes* and a set of *containers*. These containers can also include elements with their own attributes. Here is an example:

```
<flow>  ← Structured activity
 <links>  ← Container
  <link name="checkFl-BookFl"/> ← Element
 </links>
 <invoke name="checkFlight" ... > ← Basic activity
  <sources>  ← Container
   <source linkName="checkFl-BookFl"/> ← Element
  </sources>
 </invoke>
 <invoke name="checkHotel" ... />
 <invoke name="checkRentCar" ... />
 <invoke name="bookFlight" ← Attribute ...>
  <targets>  ← Container
   <target linkName="checkFl-BookFl" />
  </targets>
 </invoke>
</flow>
```

WS-BPEL provides concurrency and synchronization primitives. For instance, the `flow` activity runs a set of activities in parallel. Synchronization constraints between activities can be defined. In the above example, the `flow` activity invokes three WS in parallel: `checkFlight`, `checkHotel`, and `checkRentCar`. There is another WS, `bookFlight`, that will only be invoked if `checkFlight` is completed. Activities are synchronized by linking them: the target activity of every link will only be executed if the source activity of the link has been completed successfully.

2.2 Metamorphic Testing

Software testing is a key activity in any software development project. Fault detection and correction are key activities to ensure program reliability. For this reason, a number of software testing techniques have been developed following different approaches (Beizer, 1990; Myers, G.J. et al., 2004).

One of the main challenges for most testing techniques is the *oracle problem*. Because of its nature, some programs are inherently difficult to test. For example, the output can be difficult to verify because the precise result is not known *a priori*,[1] or because the size of the output makes it unfeasible to apply standard techniques. In these cases, we should use a different approach to help with the verification and validation process.

Traditionally, human testers have been used to check the results manually, though it is clear that this is both expensive and error-prone. An alternative is to develop a simpler program implementing the same functional requirements that the original program, sacrificing efficiency and other non-functional requirements. However, this is out of the question for complex systems and reduces the problem to the correctness of the simpler program. Any mechanism for checking whether a program under test behaves correctly for any given input is called an *oracle*. As we have argued, there are practical situations where oracles can be unavailable or too expensive to apply. MT has been proposed to alleviate the oracle problem (Chen, 1998; Chen, 2010).

MT relies on the notion of *metamorphic relation* (MR). In (Andrews et al., 2005), they are defined as "existing or expected relations over a series of distinct inputs and their corresponding results for multiple evaluations of a target function". When the implementation is correct, program inputs and outputs are expected to satisfy some necessary properties that are relevant for the underlying algorithms. These properties are traditionally known as *assertions*. In these sense, a MR is a kind of assertion. However, a MR should provide a way of generating new test cases from given ones. 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 example, when the list $l_1 = \langle 4,7,2\rangle$ is used, the expected result of $sort(l_1)$ is $\langle 2,4,7\rangle$. Then, if we use the permutation $l_2 = perm(l_1, \langle(1,2)\rangle) = \langle 7,4,2\rangle$ as the *follow-up test case*, the expected result is the same. In other words, $sort(l_1) = sort(l_2)$, obviously because any permutation of a list has the same ordered list. Therefore, we could formalize this property, MR_1, as follows:

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

[1]Partial knowledge or properties of the result can, indeed, be available.

where l_1 is the original input, l_2 is a follow-up test case (a permutation of l_1 in this particular case) and *perm* is the function applying a permutation to a list (just one swap in the example; the first and the second elements). If the metamorphic property is not satisfied then the program is faulty.

Please, notice that *perm* is the test generation function associated to MR_1. It receives an input list (the given test case) and a permutation (a list of index pairs), and produces a new list (the follow-up test case). Test case generation can be automated as in traditional techniques. For example, 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 resulting list must be a permutation of the original) and an order constraint (it must be sorted). On the other hand, a MR is usually devised by an expert in the application domain.

Summing up, MT is a testing technique using MR (Chan et al., 2007). It begins with an initial test suite, which is produced with any test case selection strategy, and a set of MR. 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, MR 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 TESTING WEB SERVICES COMPOSITIONS

In this section, we describe some tools and techniques applied to test this kind of software. We focus in those which are implemented in WS-BPEL, specially the automated ones. Most of them are referred in bibliography to test case generation of those compositions (García-Fanjul et al., 2007; Yan et al., 2006; Zheng et al., 2007). There are more complete works including the measure of test case quality and optimizing test case generation through mutation testing. GAmera (UCASE Research Group, 2010a) is a mutation testing framework to WS-BPEL that uses genetic algorithms to reduce the number of mutants required. It classifies mutants as killed (output is different than original composition), alive (output is the same as the original composition against the entire test case set performed), equivalents (output is always the same as the original composition), and erroneous (result in failure of the unfold) (Domínguez-Jiménez et al., 2009). Other testing framework called Takuan (UCASE Research Group, 2010b) is based on dynamic generation of invariants for WS-BPEL compositions. Lastly, a recent work describes different technologies developed in web services and web services testing (Bozkurt et al., 2010).

4 METAMORPHIC TESTING IN SOME ENVIRONMENTS AND PROGRAMS

The first documented work about MT belongs to Weyuker (Weyuker, 1982). In this paper she proposed a new perspective for software testing based on using alternative programs to the original code sharing the same objective to prove the original program. However, she only focused on numerical functions and the relations were of equality, such as $(a+b)^2 = a^2 + 2a*b + b^2$.

That original notion was later adopted by Chen (Chen, 1998), who defined the term as *Metamorphic Testing* and extended it, including non-numerical functions to be tested and MR but not necessarily equality as previously.

Since then, other works have been written, each one focusing on one or more different aspects of the theory. For example, (Zhou et al., 2004) describes diverse programs to apply MT. (Gotlieb and Botella, 2003) relates to automatization of this technique and where it is applicable. Moreover, a research group from Columbia University has implemented part of the process to apply MT using JML execution environmental (Murphy et al., 2009). Furthermore, Chen and his colleagues in (Chen et al., 2004) describe the importance of choosing MR suitable with problem domain and algorithm structure of the program to prove. In addition, Chan and his collaborators have different approaches about this subject. Although, the most interesting for us is how to apply MT to SOA software (Chan et al., 2006; Chan et al., 2007). Furthermore, there is other interesting issue based on the analysis of feature model to obtain MR and automate the test data generation (Segura et al., 2010).

4.1 MT with Numerical Problems

MT may be applied to resolve issues related to oracle problems with partial differential equations where the solution is unknown. A related study case (Chen et al., 2002) addresses how to successfully apply MT techniques to a thermodynamic problem to obtain certain properties representing MR.

4.2 MT with Non-numerical Problems

This technique is not limited to numerical problems and may be applied to programs designed to solve non-numerical problems (Zhou et al., 2004). For example, it has been proven in programs implementing algorithms that solve graph problems, with known properties that may be proven and must be met. MT has also been applied in computer graphics software. Additionally it may be used in testing compilers when it becomes too difficult to prove the equivalence between source and object code.

Another approach is to use interactive software where inputs to program are user inputs, instead of data. As example, test cases to testing a browser are HTML files and action sequences like this:

$$EnterURL \to ClickItem1 \to$$
$$SelectMenufile \to SelectOpen...$$

There may be several action sequences for a specific user defined requirement. By relating all inputs to their suitable outputs for every matching action we obtain MR. If we select the appropriate MR set we may apply MT to this kind of software.

4.3 MT with Learning Machine Programs

A group of Columbia University have implemented a framework (Murphy et al., 2008) automating part of the testing process. They have analyzed programs that implement learning machines applications to automize how to obtain properties for this type of software. In (Murphy et al., 2008) categorized six types of metamorphic properties that such applications may have. Then implemented a tool called Corduroy to automate the process by allowing developers to specify individual functions of metamorphic properties using the specification language JML; these properties could then be checked using JML Runtime Assertion Checking (Murphy et al., 2009). This way they obtain and check Metamorphic Relations in order to apply MT.

5 AN APPROACH OF MT IN WEB SERVICES COMPOSITIONS

Once the different testing aspects to web services and MT have been analized, we propose to apply MT to web services compositions in WS-BPEL and implement MT by integrating well-tested open-source systems: ActiveBPEL as the WS-BPEL engine and BPELUnit as the unit test library. ActiveBPEL is a WS-BPEL 2.0 compliant open-source engine. Compared to other engines, it is quite lightweight, reducing the time needed to run a test suite. It is maintained by ActiveVOS (ActiveVOS, 2009), which offers commercial products based on it. BPELUnit is a WS-BPEL unit test library (Mayer and Lübke, 2006) which can use any WS-BPEL 2.0 compliant engine. It uses XML files to describe test suites. It can replace external services with mockups providing predefined responses.

Our approach is based in the two following ideas:

1. Based in Murphy and his collaborators (Murphy et al., 2008; Murphy et al., 2009), the first idea consists of achieving program properties from learning machines software, automating MR selection.
2. The second idea is to specify, design and implement a new web service called *metamorphic service* that wraps the service to prove with MR. This approach is proposed by Chan and his colleagues in (Chan et al., 2006; Chan et al., 2007)

A generic diagram of our approach is as follows: Firstly, a *metamorphic composition* will be specified to encapsulate each composition to be proven, to include MR previously selected by analyzing the composition to obtain properties. This way it will allow us to automate part of the process. Furthermore, MR will allow to generate follow-up test cases and prove results sound respect properties and, therefore, prove the original composition. Finally, metamorphic composition will issue a report with results to be analyzed. In our proposal (Fig. 1) is necessary for each composition:

- Select adequate MR.
- Generate initial test cases.
- Get follow-up test cases.
- Multiple executions of composition.
- Analyze results.

To begin, we take a composition example. We comment briefly this approach with the classical WS-BPEL example of the *Loan Approval Service* included in the WS-BPEL 2.0 specification (OASIS, 2007). To begin, we have to develop a program to ingest a file containing a BPELUnit Test Suite (BPTS) using templates to generate initial test cases and return other BPTS with follow-up test cases (from initial ones applying MR previously selected). To achieve this, we will do the following:

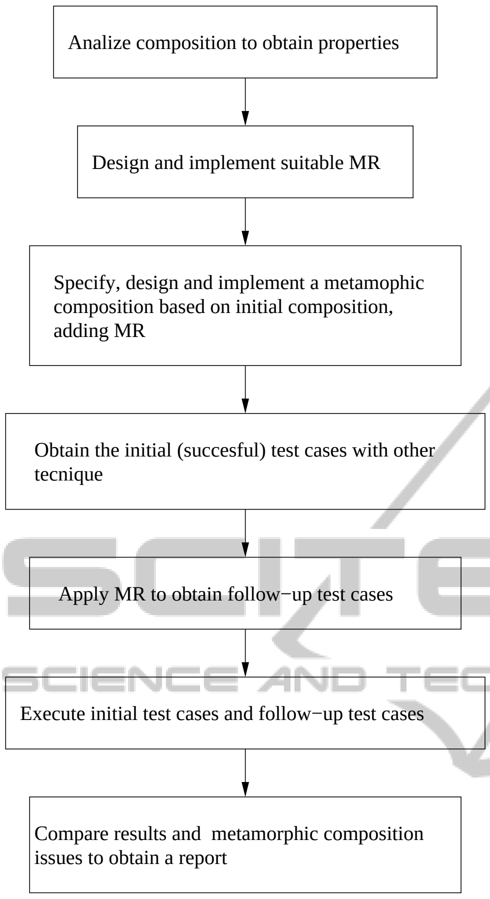
Analize composition to obtain properties
Design and implement suitable MR
Specify, design and implement a metamophic composition based on initial composition, adding MR
Obtain the initial (succesful) test cases with other tecnique
Apply MR to obtain follow−up test cases
Execute initial test cases and follow−up test cases
Compare results and metamorphic composition issues to obtain a report

Figure 1: A component diagram.

- Hand write a BPTS for the example composition using templates. This is already completed.
- Write the data file to generate the initial test cases by the BPTS. This is completed as well. It is a CSV file.
- Design the MR.
- Implement a program that receives the data file(CSV) and generates a new file (CSV) using BPTS to generate one or more test cases using the MR designed. We have to associate the initial test cases with the new ones.
- Execute the sample composition with the initial and new test cases.
- Analize and compare the results in both executions.

Another option would be to develop a program to ingest a BPTS (initial test cases written by hand) and return other BPTS once MR are applied. by completing the following:

- Load BPTS in memory (this would be done by BPELUnit).
- Conver the BPTS in memory using MR.
- Use XMLBeans to save the new BPTS from memory to a file in disk.

We are currently working on the first option of the proposed approach. The proposed Metamorphic composition includes all the tools and files used to generate initial and follow-up test cases, as well as, MR code implemented besides original composition.

6 CONCLUSIONS AND FUTURE WORK

Business processes based in WS-BPEL compositions are rapidly becoming commonplace in recent years, so it is important to devote more attention to testing in this context.

Therefore, it is required more innovative and efficient techniques in software testing to WS compositions.

The WS-BPEL programming language is oriented to business processes, and therefore its peculiarities must be considered. MT is proven efficient on different applications, and research by several groups continues. Moreover, it is a technique that can be implemented independently of program features to prove. Selection of adequate MR is a critical aspect in this technique, therefore, problem knowledge and program structure must be considered.

We have proposed a diagram of a test framework to apply MT to service compositions in WS-BPEL, specifying steps and ways to design it.

Regarding future work, the proposed framework must be formally specified, designed, and implemented, developing every step until the system is completed.

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.

Chan, W. K., Cheung, S., and Leung, K. (2006). Towards a metamorphic testing methodology for service-oriented software applications. In *Quality Software, 2005.(QSIC 2005). Fifth International Conference on*, pages 470–476.

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., Feng, J., and Tse, T. H. (2002). Metamorphic testing of programs on partial differential equations: A case study. In *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, COMPSAC '02, pages 327–333, Washington, DC, USA. 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.

Domínguez-Jiménez, J. J., Estero-Botaro, A., García-Domínguez, A., and Medina-Bulo, I. (2009). GAmera: An automatic mutant generation system for WS-BPEL compositions. In *ECOWS 2009: Seventh IEEE European Conference on Web Services*, pages 97–106, Eindhoven, The Netherlands. IEEE Computer Society.

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.

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, New York, NY, USA. 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.

Segura, S., Hierons, R. M., Benavides, D., and Ruiz-Cortes, A. (2010). Automated test data generation on the analyses of feature models: A metamorphic testing approach. *Software Testing, Verification, and Validation, 2008 International Conference on*, 0:35–44.

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

UCASE Research Group (2010b). Takuan home site. https://neptuno.uca.es/redmine/projects/takuan-website.

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

Yan, J., Li, Z., Yuan, Y., Sun, W., and Zhang, J. (2006). BPEL4WS unit testing: Test case generation using a concurrent path analysis approach. In *ISSRE 2006: 17th International Symposium on Software Reliability Engineering*, pages 75–84, Raleigh, North Carolina, USA. IEEE Computer Society.

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.