An Application of Software Fault Injection for Assessment of Quality of Test Sets for Business Processes Orchestrating Web-Services

Damian Grela, Krzysztof Sapiecha and Joanna Strug
Department of Computer Science, Cracow University of Technology, Cracow, Poland

Keywords: Test Sets Quality Assessment, Fault Injection, Mutation Testing, Web-Services, Business Processes, BPEL.

Abstract: The paper presents an experiment of the application of software fault injection to assess quality of test sets for business processes orchestrating web-services. The mutation testing, usually used to this end, suffers from high computational costs of generating and running mutants. In contrast to mutation testing, faults injection can be performed at a run-time. Run-time changes are introduced by a Software Fault Injector for BPEL Processes (SFIBP). SFIBP is implemented as a special service that manipulates invocations of web-services and values of their internal variables. As for time requirements, the experiment proved high superiority of the application of the SFIBP over the mutation testing.

1 INTRODUCTION

Recently, an application of WS-BPEL (Business Process Execution Language for Web-services) has become one of the most promising technologies for developing IT systems. WS-BPEL is a high level language that makes it possible to implement business processes as an orchestration of pre-existing web-services (Oasis, 2007). A developer of an IT system should only select the most appropriate web-services and coordinate them using WS-BPEL language into business processes that cover specification requirements for the system. This leads to a very simple and structured architecture where only a special element of the process called its coordinator and communication links between the coordinator and the services need to be tested. Nevertheless, the testing should be performed with the help of a high quality test set to provide a confidence to system dependability. Thus, the development of tests should be supported by effective techniques for evaluating quality of test sets.

Mutation testing (Offutt and Untch, 2000) (Woodward, 1993) is currently the most effective technique for quality evaluation of tests. In mutation testing faulty versions of an implementation of the object (so called its mutants) are generated, by introducing small syntactic changes into the code, and executed against a test set. Although the technique is very efficient, it suffers from high computational cost of generating and executing mutants.

The paper presents a computational experiment aiming at evaluation of a novel approach that uses fault injection technique (Hsueh, Tsai and Iyer, 1997) to evaluate quality of tests for BPEL processes orchestrating web-services. In contrast to mutation testing, fault injection can be performed at a run-time of the processes. Thus, an application of this technique can significantly reduce the total cost of testing, as there will be no need to create and compile a large number of the mutants. An experiment that compares results of applying tests for mutants of a BPEL process with results of applying the same tests for the process but modified at a run-time by injecting faults is described. Mutants are generated with the help of MuBPEL (MuBPEL - WS-BPEL Testing Tools) and the faults are introduced by Software Fault Injector for BPEL Processes (SFIBP). The experiment shows to what extent the fault injection-based approach can evaluate quality of tests, and how much it costs.

The paper is organized as follows. Section 2 contains a brief description of the background and related work. The problem is stated in section 3. Section 4 describes the experiment, business processes used in the experiment, procedures and results of the experiment. The paper ends with conclusions in section 5.

Grela D., Sapiecha K. and Strug J..
An Application of Software Fault Injection for Assessment of Quality of Test Sets for Business Processes Orchestrating Web-Services.
DOI: 10.5220/0004422000560062
In *Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2013), pages 56-62
ISBN: 978-989-8565-62-4

2 BACKGROUND AND RELATED WORK

A number of papers related to different aspects of testing BPEL processes have already been published (Dong, Yu and Zhang, 2006) (Yan et al., 2006) (Yuan, Li and Sun, 2006). However, the papers do not consider the testing of BPEL processes in which the coordinator orchestrates web-services. A method of generation of test scenarios for validation of the coordinator of a BPEL process was given in (Sapiecha and Grela, 2008a). Tests obtained by means of the method cover all functional requirements for the process and provide high validation accuracy (Sapiecha and Grela, 2008b). Hence, such tests could also be used as a starting set of tests for the process.

Quality of generated test sets is an important issue, as only tests of high quality (high ability to detect faults) can help to provide dependable products (Wagner and Gericke, 2008) (Farooq and Lam, 2009). Several studies have proved validity of the mutation testing as a powerful technique for testing programs and for evaluation of the quality of test sets (Farooq and Lam, 2009) (Estero-Botaro, Palomo-Lozano and Medina-Bulo, 2008). A quality of the test set is determined by a ratio of mutants detected by the tests over all non-equivalent mutants (a mutation score). The higher is the mutation score the higher is the quality of tests. In the paper results of mutation testing were used as the reference when the results of fault injection were evaluated. The mutation testing is a white box testing technique that creates a large number of faulty programs (mutants) by introducing simple flaws (faults) in the original program. If a test case is able to detect the difference between the original program and the mutant, it is said that such test case kills the mutant. On the contrary, if none of the test cases is able to detect a difference, it is said that the mutant keeps alive for all used test cases. The mutants are created by applying so called mutation operators. Each of the mutation operators corresponds to a certain category of errors that the developer might commit. Such operators for various programming languages, including BPEL have already been designed (Offutt and Untch, 2000) (Woodward, 1993) (Estero-Botaro, Palomo-Lozano and Medina-Bulo, 2008).

Fault injection (Hsueh, Tsai and Iyer, 1997) is a popular technique that is mainly used for evaluation of fault-tolerance of computer systems. It consists in injection of deliberate faults into a running system and observation of its behaviour. So called fault coverage (Hsueh, Tsai and Iyer, 1997) for a set of tests is measured. The fault coverage is expressed as a percentage of detected faults to all faults injected into the system. Fault coverage is used as a metric of quality of a set of tests and plays similar role as the mutation score for mutation testing.

Originally fault injection was applied to hardware systems, but currently it is also applied in software and mixed ones. Software fault injection (SFI) is implementation-oriented technique thus it targets computer applications and operating systems. SFI can be performed in near real-time, allowing for a large number of experiments. The technique was already applied for systems based on web services orchestration to emulate SOA faults at different levels (Reinecke and Wolter, 2008) (Juszczyk and Dustdar, 2010). The approaches were built upon existing fault injection mechanisms. However, these solutions are still under development. It is not clear which types of SOA faults are supported, and how the faults are modelled and injected. Moreover, these works do not concern quality of test sets.

3 PROBLEM STATEMENT

Quality of a test set impacts results of the testing, as only such of the sets which detect all faults in a system can answer the question whether the system is fault free or not. For object systems tests are usually evaluated via mutation testing, but this technique is very expensive due to the number of mutant that need to be generated, compiled and executed against the test set.

A BPEL process uses web-services but the process itself is a web-service, too. Thus it needs to be deployed. So this concerns its mutants. The deployment is very time consuming because an application implementing the process and its related files need to be uploaded to a server. Since then the web application becomes available to the testing. This treatment must be repeated for all of the mutants. Thus, in contrast to other kinds of object systems, here the mutation testing seems to be rather complicated and expensive. In contrast to the mutation testing, the software fault injection generates faulty versions of the process at a run-time. No the compilation and the deployment are required. Hence, it seems that not the mutation testing, but the fault injection should be used to evaluate quality of test sets for BPEL processes.

An experiment aiming at providing this claim is presented in the paper. During the experiment, mutation testing and fault injection are applied to evaluate quality of the same sets of tests derived for ten example BPEL processes orchestrating web-services.

4 COMPUTATIONAL EXPERIMENT

The experiment consists of the following three stages:

1. Application of traditional mutation testing to example BPEL processes,
2. Application of software fault injection to the BPEL processes.
3. Comparison of the results of mutation and fault injection.

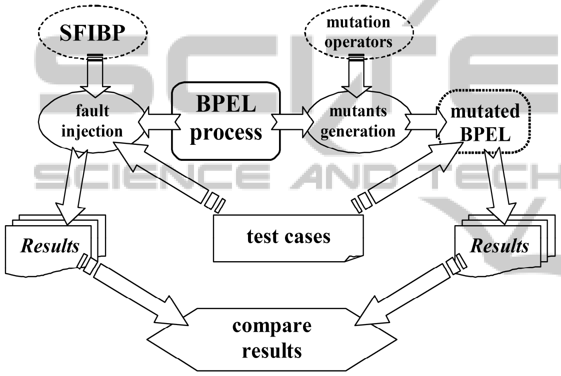

Figure 1: Flow diagram of the experiment.

BPEL processes under consideration are described in Section 4.1, later on. Test sets, three for each of the processes, were implemented as test scripts that automate an application of single test cases. Test scripts were prepared using BPELUnit (Mayer and Lubke, 2006), which is an open-source WS-BPEL unit testing framework for BPEL processes. Mutants of a BPEL process were generated (using MuBPEL) by applying mutation operators (Section 4.2) and executed against the tests. SFIBP was employed to inject faults into the correct BPEL process (Section 4.3). The same test sets were used to stimulate the process. Finally, results from applying mutation testing and fault injection were compared.

4.1 Processes and their Test Sets

The experiment was executed on ten example BPEL processes (four our own and six taken from a public repository shared by the University of Cadiz (University of Cadiz WS-BPEL Composition Repository)). For each of the processes, three test sets were provided (first two randomly generated, last one obtained by checking paths method (Sapiecha and Grela, 2008). Table 1 contains the details: identifiers (ID), names of the processes (Name), the number of web-services used by the processes (WS) and the number of test cases provided in each of the test sets (TS). No fault tolerance mechanisms were used.

Table 1: BPEL processes used in the experiment.

ID	Name	WS	TS
PDO	Planning Distribution of Orders helps its users to distribute orders among stores.	5	5/4/3
FRS	Football Reservation System allows its users to book tickets for football games, hotels to stay during the games and plane or train tickets to arrive at the games.	5	7/5/3
OB	Order Booking receives orders placed by users, it verifies the user and routes each order to two suppliers to get quotes and chooses the supplier that provided the lower quote.	8	12/10/4
PES	Project Evaluation System allows its users, students and teachers, to submit and received projects for evaluation, it control the evaluation sums the results.	6	9/14/8
LA	Loan Approval concludes whether a certain request for a loan will be approved or not (it was published within the specification of the WS-BPEL 2.0 Standard.	2	6/9/5
SS	Squares Sum computes the value of sum (i=1 to n) for a certain value of n.	0	3/3/2
TS	Tac Service inverts the order of the lines in a file.	0	4/6/4
MP	Market Place receives a price and offer from two partners: buyer and seller, and compares if the price offered by the buyer is equal or higher that set for the seller to sell.	2	9/12/8
TI	Trade Income models the behaviour of managing a supermarket, controls the total profits that were generated by the different establishments that the supermarket has, checks stock, etc.	7	12/6/6
MS	Meta Search implements a meta-search engine, which queries mockups of the Google and MSN search engines, interleaves their results and removes duplicates.	2	7/9/7

4.2 Mutation Testing

Mutation testing was performed with a help of MuBPEL. The MuBPEL is a mutation testing tool for BPEL that automatically generates mutants of a BPEL process, executes the mutants against provided test set and finds the difference in output of both (mutated and original) BPEL processes. The

MuBPEL generates mutants with exclusion of the equivalent ones. A user only needs to prepare a BPEL process and a set of its tests. The tests need to be created as test scripts using BPELUnit.

Only 12 out of 26 operators defined in (Estero-Botaro, Palomo-Lozano and Medina-Bulo, 2008) were used in the experiment. The remaining 14 were skipped, as they refer to features of BPEL processes that are not supported by current version of SFIBP. All 12 operators listed in Table 2 have been implemented in the MuBPEL.

Table 2: Mutation operators used in the experiment.

Operator	Description
Identifier replacement operators	
ISV	Replaces a variable identifier by another of the same type
Expression operators	
EAA	Replaces an arithmetic operator (+,-,*, div,mod) by another of the same type
EEU	Removes the unary minus operator from an expression
ERR	Replaces a relational operator (<,>,>=,<=,=,!=) by another of the same type
ELL	Replaces a logical operator (and,or) by another of the same type
ECC	Replaces a path operator (/,//) by another of the same type
ECN	Modifies a numerical constant incrementing or decrementing its value in one unit, adding or removing one digit
Activity operators (concurrent)	
ASF	Replaces a sequence activity by a flow activities
Activity operators (non-concurrent)	
AEL	Deletes an activity
AIE	Deletes an elseif element of the else element from an if activity
AWR	Replaces a while activity by repeat-until and vice versa
ASI	Exchanges the order of throw sequence child activities

Mutation testing was performed accordingly to the following scenario:

1. Generation of mutants of the BPEL process by applying the operators given in Table 2,
2. Execution of the mutants against both test sets and comparison of results produced by the mutants with values calculated by fault-free process,
3. Calculation of the mutation score for each test set,

Steps 2, 3 and 4 were repeated for every group of mutation operators described in Table 2.

Tables 3 and 4 summarize the results of the mutation testing. It gives the number of mutants generated, mutants killed by each of the test sets (Table 3) and the mutation scores (*MS*) for each of the test sets (Table 4). A set of test cases is ***mutation adequate*** if its *MS* is 100%.

$$MS(T) = \frac{M_K}{M_T - M_E} \cdot 100\% \text{, where} \qquad (1)$$

T - denotes a test set
M_K – is the number of mutants killed by the test set
M_T – is total number of generated mutants
M_E – is the number of equivalent mutants

Table 3: Results of mutation testing.

BPEL process	mutants generated	mutants killed		
		TS1	TS2	TS3
PDO	229	184	186	190
FRS	219	193	182	191
OB	639	586	547	597
PES	687	492	552	596
LA	28	20	23	23
SS	45	42	41	43
TS	53	43	47	48
MP	29	25	27	27
TI	557	551	528	556
MS	525	411	471	479

Table 4: Mutation score.

BPEL process	mutation score MS [%]			
	TS1	TS2	TS3	average
PDO	80,34	81,22	82,97	81,51
FRS	88,13	83,11	87,21	86,15
OB	91,70	85,60	93,43	90,24
PES	71,61	80,34	86,75	79,57
LA	71,43	82,14	82,14	78,57
SS	93,33	91,11	95,56	93,33
TS	81,13	88,67	90,56	86,79
MP	86,21	93,10	93,10	90,80
TI	98,92	94,79	99,82	97,85
MS	78,28	87,81	91,24	85,77

4.3 Fault Injection

Fault injection was executed with a help of a Software Fault Injector for BPEL Processes (SFIBP). The SFIBP is an execution-based injector (Benso and Prinetto, 2003), that is able to inject faults into the BPEL processes at a run-time, thus it simulates effects of the faults. Such approach helps to reduce costs of the experiment, as the faults are injected without changing the implementation of a process. The SFIBP is implemented as a special local service that is invoked between or instead of the proper web-service invocation.

The SFIBP generates the following four types of faults:

- disturbances of web-service output parameters (OP),
- disturbances of values of web-service input parameters (IP),
- replacing requested web-service with another one (WS),
- disturbances of a value of the variable (RV).

Fault injection was performed accordingly to the following scenario:

1. configuration of the SFIBP,
2. execution of the BPEL process, run-time injection of faults and comparison of results (against values calculated by fault-free process),
3. calculation of the fault coverage for each test set.

Configuration of the SFIBP includes setting of fault types, probability of their occurrence and of predefined web-services and values which are used when faults are injected. Information about the injected faults is stored in a log file. Steps 2 and 3 were repeated for each type of the faults. The total number of faults injected for a process always equals the number of mutants generated in the previous stage of the experiment.

Tables 5 and 6 summarize the results of the fault injection. It reports, for each of the processes, total numbers of faults injected, faults detected by each of the test sets (Table 5) and fault coverage (*FC*) for each of the test sets (Table 6). *FC* for a test set is defined as a percentage of detected faults to all injected faults. *FC* should be 100%.

$$FC(T) = \frac{F_D}{F_I} \cdot 100\% \text{, where} \quad (2)$$

F_D – is the number of faults detected by the test set,
F_I – is total number of injected faults.

Table 5: Results of fault injection.

BPEL process	faults injected	faults detected		
		TS1	TS2	TS3
PDO	229	179	181	188
FRS	219	187	177	190
OB	639	582	541	595
PES	687	486	547	591
LA	28	20	22	23
SS	45	39	40	42
TS	53	42	45	48
MP	29	18	21	23
TI	557	546	521	553
MS	525	405	456	474

Table 6: Fault coverage.

BPEL process	fault coverage FC [%]			
	TS1	TS2	TS3	average
PDO	78,16	79,04	82,09	79,77
FRS	85,39	80,82	86,76	84,32
OB	91,08	84,66	93,11	89,62
PES	70,74	79,62	86,03	78,80
LA	71,43	78,57	82,14	77,38
SS	86,67	88,89	93,33	89,63
TS	79,24	84,90	90,57	84,90
MP	64,28	75,00	82,14	73,81
TI	98,02	93,53	99,28	96,94
MS	77,14	86,86	90,29	84,76

4.4 Comparison

Results of the fault injection are close to the results of the mutation testing for all evaluated test sets. As it can be observe in Table 3 and 4 average fault coverage differs from average mutation score from 0,62% (for OB) to 4,76% (for LA). Higher consistency of results was observed in the case of larger systems. For such systems (OB, TI or MS), the difference did not exceed 2%

Each technique uses its own fault model, thus changes made by mutation operators and faults injected by SFIBP are completely different kind of faults. Despite the lack of dependency between mutants and the injected faults, the results of both approaches are similar (the behaviour of a process differs from the expected).

Another notable feature is the time overhead. Tables 7, 8 and 9 present the total execution time of mutation testing (Table 7) and fault injection (Table 8). All BPEL processes were executed on the same hardware configuration (Intel Core2Duo 1.2GHz processor, 2GB RAM).

Table 7: Execution time of mutation testing.

BPEL process	Mutation Testing time MIt [s]		
	TS1	TS2	TS3
PDO	2311	1963	1444
FRS	1434	1176	918
OB	7517	6796	4049
PES	17193	25490	16554
LA	239	309	214
SS	241	245	169
TS	474	667	481
MP	1463	1995	1394
TI	59089	29128	29396
MS	11276	14418	11387

Table 8: Execution time of fault coverage.

BPEL process	Fault Injection time FIt [s]		
	TS1	**TS2**	**TS3**
PDO	1652	1304	941
FRS	937	679	493
OB	4679	3959	2147
PES	9874	14939	9574
LA	194	245	175
SS	174	178	119
TS	359	548	364
MP	1238	1775	1174
TI	36932	16815	16932
MS	7583	9953	7606

Table 9: MT and FI execution time ratio.

BPEL process	MTt / FIt			
	TS1	**TS2**	**TS3**	**average**
PDO	1,40	1,50	1,53	1,479
FRS	1,53	1,73	1,86	1,708
OB	1,61	1,72	1,88	1,736
PES	1,74	1,71	1,73	1,725
LA	1,23	1,26	1,22	1,239
SS	1,38	1,37	1,42	1,394
TS	1,32	1,22	1,32	1,286
MP	1,18	1,12	1,19	1,164
TI	1,59	1,73	1,74	1,689
MS	1,49	1,45	1,50	1,478

The results proved that the fault injection is much faster than the mutation testing (Table 9) for all the test sets (about 1,5 times faster). Fault injection-based approach is particularly cost effective for large systems (e.g. OB, TI) due to the lack of deployment of huge number of mutants. For smaller systems (e.g. MP, LA), the results are less effective thus for such systems the selection of test method is arbitrary.

5 CONCLUSIONS

Cost effective testing extensive software systems requires specific approaches and different technologies adjusted to specific architectures. The experiment proved that testing based on SFI might be attractive for service oriented architectures (SOA) implemented with the help of BPEL. This is almost as effective as mutation testing but does not need elaboration of mutants. Moreover, it is much faster because can be performed at a run-time of the process. Hence, this might be much more cost effective.

From the experiment it results that even random testing enables detection a lot of faults in the processes. Usually these faults are easy detectable ones. Using validation test sets seems to be more effective than random testing. The more complex is the process the higher are benefits from the fault injection and using validation test set, especially for time requirements. This last one is derived at the very beginning of the development of the system running the process, and thus need not any extra effort while testing. The results are promising. However, other object oriented architectures have to be taken into account to answer the question to what extent and when the fault injection may be an alternative for the mutation testing. In our future research more experiments on various types of SOA will be performed to strengthen the conclusions.

REFERENCES

OASIS, 2007, *Web Services Business Process Execution Language 2.0*, http://docs.oasis-open.org/wsbpel/2.0/. Organization for the Advancement of Structured Information Standards.

W.-L. Dong, H. Yu, Y.-B. Zhang, 2006. Testing BPEL-based web service composition using high-level Petri nets. In *EDOC 2006: Tenth IEEE International Enterprise Distributed Object Computing Conference. Hong Kong, China: IEEE Computer Society*, 2006, pp. 441–444.

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

Y. Yuan, Z. Li, W. Sun, 2006. A graph-search based approach to BPEL4WS test generation. In *ICSEA 2006: International Conference on Software Engineering Advances*. Papeete, Tahiti, French Polynesia: IEEE Computer Society, p. 14.

K. Sapiecha, D. Grela, 2008a. Test scenarios generation for certain class of processes defined in BPEL language. In *Annales UMCS - Informatica*, vol.8, number 2/2008, pp.75-87

K. Sapiecha, D. Grela, 2008b. Automating test case generation for requirements specification for processes orchestrating web services. In *Information Systems Analysis and Specification vol.1*, 10th International Conference on Enterprise Information Systems (ICEIS), Barcelona, pp. 381-384.

S. Wagner, J. Gericke, M. Wiemann, 2008. Multi-Dimensional Measures for Test Case Quality. In *ICSTW '08*. IEEE International Conference on Software Testing Verification and Validation Workshop.

U. Farooq, C. P. Lam, 2009. *Evolving the Quality of a Model Based Test Suite*. In *ICSTW '09*. International Conference on Software Testing, Verification and Validation Workshops.

A. J. Offutt, R. H. Untch, 2000. Mutation testing for the new century. *Norwell*, Massachusetts, USA: Kluwer Academic Publishers, 2001, ch. Mutation, Uniting the Orthogonal, pp. 34–44.

M. R. Woodward, 1993. Mutation testing — its origin and evolution. In *Information and Software Technology*, vol. 35, no. 3, pp. 163–169

M. C. Hsueh, T. K. Tsai, R. K. Iyer, 1997. Fault Injection Techniques and Tools. In *IEEE Computer*, vol. 30, no. 4, pp. 75-82.

P. Reinecke, K. Wolter, 2008. Towards a multi-level fault-injection test-bed for service-oriented architectures - requirements for parameterisations. In *27th International Symposium on Reliable Distributed Systems*, Napoli, Italy.

L. Juszczyk, S. Dustdar, 2010. Programmable fault injection testbeds for complex SOA. In *8th International Conference on Service Oriented Computing* (ICSOC'10), San Francisco, USA.

A. Estero-Botaro, F. Palomo-Lozano, I. Medina-Bulo, 2008. Mutation operators for WS-BPEL 2.0. In *ICSSEA 2008: 21th International Conference on Software & Systems Engineering and their Applications*, Paris, France.

P. Mayer, D.Lubke, 2006. Towards a BPEL unit testing framework. In *TAV-WEB'06: Proceedings of the workshop on Testing, analysis, and verification of web services and applications*, pp. 33–42. ACM, New York.

A. Benso, P. Prinetto, 2003. Fault injection techniques and tools for embedded systems reliability evaluation. *Kluwer Academic Publishers*, Holland.

MuBPEL - WS-BPEL Testing Tools, http://neptuno.uca.es/redmine/projects/sources-fm/wiki/MuBPEL

University of Cadiz WS-BPEL Composition Repository. http://neptuno.uca.es/redmine/projects/wsbpel-comp-repo