2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops

Fifty Shades of Grey in SOA Testing

Franz Wotawa, Marco Schulz
Ingo Pill, Seema Jehan (in reverse a.o.)
Institute for Software Technology
Graz University of Technology, Austria
Email: {wotawa,mschulz,ipill,sjehan}@ist.tugraz.at

Philipp Leitner, Waldemar Hummer,
Stefan Schulte, Philipp Hoenisch, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology, Austria
Email: {p.hoenisch,leitner,s.schulte,dustdar}@infosys.tuwien.ac.at

***Abstract*—Testing is undisputedly a fundamental verification principle in the software landscape. Today's products require us to effectively handle and test huge, complex systems and in this context to tackle challenging traits like heterogeneity, distribution and controllability to name just a few. The advent of Service-Oriented Architectures with their inherent technological features like dynamics and heterogeneity exacerbated faced challenges, requiring us to evolve our technology. The traditional view of white or black box testing, for example, does not accommodate the multitude of shades of grey one should be able to exploit effectively for system-wide tests. Today, while there are a multitude of approaches for testing single services, there is still few work on methodological system tests for SOAs. In this paper we propose a corresponding workflow for tackling SOA testing and diagnosis, discuss SOA test case generation in more detail, and report preliminary research in that direction.**

I. Introduction

Nowadays software is an omni-present concept. Even for simplest appliances like a coffee machine, software-enabled versions promise an enhanced and possibly customizable user experience as well as cheap product/software updates for the vendor. The steady increase of available computation resources enabled complex systems, for example, simulating entire worlds for multi-player online-games, and today's online-shopping, -banking and -stock trading platforms deliver new business opportunities besides enhancing convenience in our everyday's lives. Such a demanding landscape made concepts like modularization and the corresponding reuse of modules and (possibly 3^{rd} party) Intellectual Property cores mandatory in order to be able to compete on the market.

The principles of Service-Oriented Architectures (SOAs) have been gaining attention in this context and are nowadays widely accepted in industry [1]. The inherent advantages in respect of interoperability, reusability and compatibility, as well as the loose (i.e. dynamic) couplings between clients and servers leveraged new R&D in challenging fields such as autonomic computing [2], adaptive systems [3], self-healing distributed systems [4] and cloud computing [5].

Besides formal techniques like model-checking, testing is undisputedly a fundamental verification principle for software development. In the context of SOAs, testing is a complex and demanding task [6] due to inherent traits like heterogeneity, distribution and dynamics that, for example, make controllability and observability severe challenges [7], [8], [9]. Thus, while there are numerous approaches for testing single services [10], [11], [12], [13], [14], [15], there is little work on system-wide tests assessing essential system traits such as performance, stability, robustness, scalability and other functional and non-functional properties.

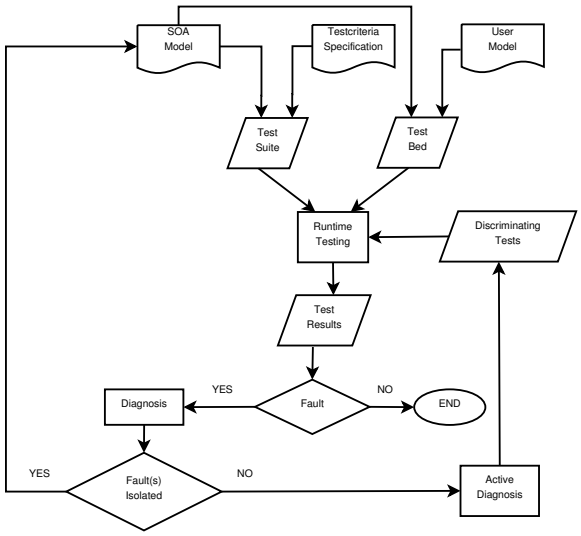

Fig. 1. Overview

We argue that the technology for corresponding SOA test workflows is mandatory for being able to provide established software quality levels also for SOAs. Thus our mission is to take up that challenge, where we depict in Figure 1 an overview of our example test and diagnosis [16] workflow. In this paper our focus is on the testing purpose.

The traditional view of white or black box testing is incompatible with the many shades of grey experienced when considering SOAs, as resulting from the accessible information levels for the various services. With model-based testing (MBT) [17], [18] there is yet an attractive approach for our setting: Deriving test cases from models rather than implementations accommodates SOA scenarios where for some service from a 3^{rd} party the implementation is not accessible, while the provider still has to disclose some abstract model offering the necessary details for system integration. Our general approach to deal with these shades of grey in SOA testing is thus as follows: Considering service compositions defined as BPEL processes [19], we can perform white box testing at the top

978-0-7695-4993-4/13 $26.00 © 2013 IEEE
DOI 10.1109/ICSTW.2013.26

level when considering each service as a black box (atomic component). This leaves the dynamics of service compositions to be taken into account, where also further details (gray shades) gained from available data for individual services can augment abstract BPEL data.

In this paper we show our initial research and discuss the resulting perspectives, while arguing that our contribution is an important step for system-wide SOA testing. In the following we discuss two instantiations of SOA testing. These variants demonstrate well the many perspectives of SOA testing that occur in practice. In Section II we show how to derive test cases from BPEL specifications to be leveraged by frameworks like Genesis2 [20]. This control-flow focused approach is complemented by a data-flow based one (see also [21], [22]) discussed in Section III, that takes, for instance, SOA dynamics into account. We conclude the paper discussing perspectives and future work in Section IV.

II. Generating Test Data from BPEL

Academic literature offers various approaches for generating test cases from BPEL specifications, i.e., relying on *symbolic execution* [23], *petri nets* [24], [25], *model checking* [26], [27], and contract based approaches [28], [29], [30]. See Bozkurt and Hassoun [31] for a survey. Those approaches most closely to our variant are Yuan et al. [32] and Yan et al. [33], both exploiting the control flow graph of a BPEL process.

Our contribution is based on model-based testing using symbolic execution of BPEL processes. We adopt a graph based search method for the test generation that relies on a combination of path extraction from control flow graphs (CFGs) and constraint solving. Both, control flow graph and constraints, can be automatically extracted from BPEL source code. The test case generation algorithm (see Figure 2) is search based and requires two inputs: a directed CFG and the maximum path length. The result is a suite of feasible test cases derived via constraint solving. Our basic concept is to first extract all possible paths from the start vertex to the end vertices in the CFG up to the pre-defined length, and then represent each path via constraints (converting each vertex separately considering pre- and postconditions). It is worth noting that our constraint representation makes use of a static single assignment form similar to the one discussed in [34]. Then, we solve the constraint satisfaction problems assessing whether the paths are feasible. For a feasible path, the derived input and output values constitute a test case.

To illustrate our approach we use the Bank Loan example depicted in Figure 3, taken from [35]. The corresponding process starts upon receiving a loan request from a client and is defined as follows: Loan requests below 10.000 credits from low risk clients are approved immediately. Those from high risk clients or with amounts starting at 10.000 credits need thorough assessment before a decision is made.

For this example there are three possible paths: (1) `loanRequest`, `amount >= 10000`, `thoroughAssessment`, `loan[decision]`, (2)`loanRequest`, `amount < 10000`, `calculateRisk`, `risk == low`, `loan[approve]`, and (3) `loanRequest`, `amount < 10000`, `calculateRisk`, `risk != low`, `thoroughAssessment`, `loan[decision]`. Let us consider path (2), making the following assumptions regarding the BPEL components' behavior: Component `loanRequest` has an empty pre-condition and $amount > 0$ as post-condition. Component `calculateRisk`'s behavior is given only partially: up to 1000 credits, the risk is assumed to be low. This partial specification can be formalized using the post-condition $amount < 1000 \rightarrow risk == low$. For `calculateRisk` the pre-condition is assumed to be empty. Taking into consideration the pre- and post-condition as well as the conditions related to other BPEL components, we obtain the following conditions for path (2):

1: `amount_0` > 0
2: `amount_1` = `amount_0`
3: `amount_1` < 10000
4: `amount_1` $< 1000 \rightarrow$ `risk_0` == `low`
5: `risk_0` == `low`
6: `loan_0` == `approved`

Note that these conditions can be easily translated for access by any constraint solver, in our case MINION [36]. We use MINION to check for a variable evaluation that satisfies all constraints. For path (2), the assignment `amount_0`=1, `amount_1`=1, `risk_0`=`low`, `loan_0`=`approved` is such an evaluation. Obviously, `amount_0` $= 1$ with the expected output `loan_0` $=$ `approved` is a valid test case and exactly ensures executing its corresponding path.

Our implementation currently focuses on synchronous executable BPEL processes, handling most of the basic activities including receive, reply, assign, invoke, if, while, and sequence. While we currently do not handle flow activities

1: **procedure** Test Case Generation($DCFG, L$)
2: Compute all paths π up to length L from start to end vertices.
3: **for** each π **do**
4: Convert π to a set of constraints
5: Solve the constraint satisfaction problem
6: **if** satisfiable **then**
7: Save π's in-/output values as test case
8: **end if**
9: **end for**
10: Return all derived test cases as test suite
11: **end procedure**

Fig. 2. Bpel Test Case Generation algorithm

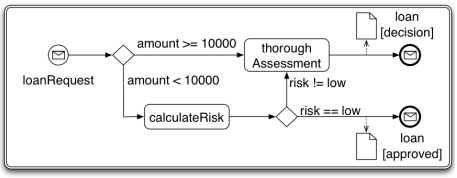

Fig. 3. The Bank Loan BPEL Process [35]

(which provide means for concurrent execution), an extension is under development where we assume that concurrent paths must not interfere. In order to evaluate our implementation we performed some experiments using the Bank Loan example, an ATM example[1], and a simple program using a while loop (While). For all examples constraint solving never took more than 11 milliseconds, where the path length varied from 8 to 50. It is worth noting that the number of paths, especially for larger examples, can grow drastically (from 3 to 16,300). Hence, there is a strong urge for reducing the number of paths in order to keep test case generation time low. Apparently, our constraint solving step seems not to be a limiting factor.

III. Data-Flow Based Testing using the k-Node Data Flow Criterion

While the testing approach discussed in Section II helps verifying the control flow of the BPEL process under test, it does not explicitly deal with the problem of dynamics in BPEL processes. More concretely, when instantiating an abstract BPEL process, it is unclear which service implementations can be chosen simultaneously, without breaking the functionality of the composition because of service incompatibilities. In the following, we discuss the TeCoS (Test Coverage for Service-based systems) approach for dealing with these issues. TeCoS has initially been introduced in [21], [22], and complements the approach sketched in Section II.

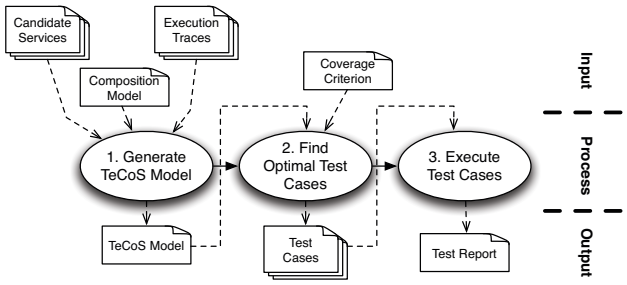

Fig. 4. Simplified TeCoS Approach

Figure 4 depicts the overal TeCoS approach. Essentially, TeCoS can be split into three phases. Firstly, the abstract service composition (e.g., the BPEL code) is enriched with existing execution traces (if available), and information about potential implementation services for each abstract activity in the process. These service candidate information can be retrieved, for instance, from a service registry [37]. From this information, a list of all potential service combinations that require testing is generated. However, taking into account the generally very large size of service compositions, and assuming that there may be many alternatives for each abstract activity in the composition, it is usually infeasible to test all potential combinations. Hence, in the second phase, the list of test cases is filtered (using some coverage criterion discussed later on) in order to find those cases most relevant to test.

[1] http://docs.jboss.com/jbpm/bpel/v1.1/userguide/tutorial.atm.html

Finally, in the third phase, the selected most important test cases are executed, similar to what we described in Section II.

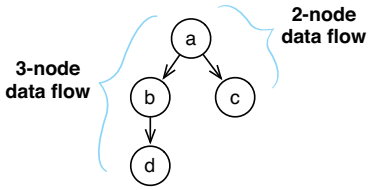

Fig. 5. Example K-Node Data Flows

From this description, it is evident that the coverage criterion used to select the most important test cases is key in TeCoS. Regarding potential candidates, we propose the use of a metric we call k-node data flow criterion. The k-node data flow criterion is grounded in the assumption that services that have direct data dependencies (i.e., the output of one service is used as input to another) have a high probability of leading to incompatibilities, and that the probability of incompatibility decreases when the distance of the services in the data flow graph increases. Hence, it seems reasonable to only test combinations which are relatively "close" in the data flow graph. We refer to this as k-node data flows: two services which have a direct data dependency are defined to produce a 2-node data flow (the minimum), services with an indirect dependency with one intermediary service produce a 3-node data flow, and so on. This is illustrated in Figure 5. $a \rightarrow b \rightarrow d$ is a 3-node data flow, while $a \rightarrow c$ is a direct dependencies (a.k.a. a 2-node data flow). Our goal when filtering test cases in Phase 2 (as per Figure 4) is to find a minimal set of test cases that still covers all k-node data flows for a selected k.

	Small	**Medium**	**Large**
$k = 2$	25	100	400
$k = 3$	125	1000	8000
$k = 4$	125	10000	160000

TABLE I
# of Test Cases for Different Values of k

Table I summarizes the minimal number of test cases that are sufficient to reach a 2-node, 3-node or 4-node data flow coverage in three different exemplary service compositions. In this example, the small composition consisted of only 6 abstract services and 5 alternatives for each service. The medium composition consisted of 10 abstract services and 10 alternatives. Finally, the large composition sported 20 services, with 20 alternatives each.

IV. Conclusions

SOA testing is a very hard challenge due to limited observability and controllability, and the dynamic nature of SOAs where a service's actual implementation used at runtime might not be known in advance. As a consequence, SOA testing requires us to adopt a Grey Box testing strategy. Moreover, the adaptiveness of SOAs has to be tested separately. Our contribution in this paper is twofold: Besides discussing the

challenges of SOA testing, we also give two variants of testing SOAs, i.e., one that relies on more or less classical model-based testing, and one that deals with testing dynamic changes in the service implementation at runtime. In future work, we will improve the empirical evaluation of our approaches, and furthermore aim at extending them, i.e., combining the BPEL testing approach with mutation testing and enhancing the approaches via introducing fault localization and diagnosis capabilities.

ACKNOWLEDGMENT

This work is partially supported by the Austrian Science Fund (FWF): P23313-N23, P22959-N23.

REFERENCES

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: a research roadmap," *International Journal of Cooperative Information Systems*, vol. 17, no. 2, pp. 223–255, 2008.

[2] K. Verma and A. P. Sheth, "Autonomic web processes," in *International Conference on Service-Oriented Computing (ICSOC)*, 2005, pp. 1–11.

[3] G. Denaro, M. Pezzè, D. Tosi, and D. Schilling, "Towards self-adaptive service-oriented architectures," in *Workshop on Testing, analysis, and verification of web services and applications (TAV-WEB)*, 2006, pp. 10–16.

[4] R. B. Halima, K. Drira, and M. Jmaiel, "A qos-oriented reconfigurable middleware for self-healing web services," in *IEEE International Conference on Web Services (ICWS)*, 2008, pp. 104–111.

[5] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[6] G. Canfora and M. D. Penta, "Testing services and service-centric systems: Challenges and opportunities," *IT Professional*, vol. 8, no. 2, pp. 10–17, 2006.

[7] L. Cacciari and O. Rafiq, "Controllability and observability in distributed testing," *Information & Software Technology*, vol. 41, no. 11-12, pp. 767–780, 1999.

[8] R. M. Hierons and H. Ural, "The effect of the distributed test architecture on the power of testing," *The Computer Journal*, vol. 51, no. 4, pp. 497–510, 2008.

[9] R. Hierons and H. Ural, "Overcoming controllability problems with fewest channels between testers," *Computer Networks*, vol. 53, no. 5, pp. 680–690, 2009.

[10] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping performance and dependability attributes of web services," in *IEEE International Conference on Web Services (ICWS)*, 2006, pp. 205–212.

[11] M. D. Barros, J. Shiau, C. Shang, K. Gidewall, H. Shi, and J. Forsmann, "Web services wind tunnel: On performance testing large-scale stateful web services," in *International Conference on Dependable Systems and Networks (DSN)*, 2007, pp. 612–617.

[12] E. Martin, S. Basu, and T. Xie, "Websob: A tool for robustness testing of web services," in *Companion to the proceedings of the 29th International Conference on Software Engineering (ICSE)*, 2007, pp. 65–66.

[13] J. Zhang and L.-J. Zhang, "Criteria analysis and validation of the reliability of web services-oriented systems," in *IEEE International Conference on Web Services (ICWS)*, 2005, pp. 621–628.

[14] W. Xu, J. Offutt, and J. Luo, "Testing web services by xml perturbation," in *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2005, pp. 257–266.

[15] H. Huang, W.-T. Tsai, R. A. Paul, and Y. Chen, "Automated model checking and testing for composite web services," in *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, 2005, pp. 300–307.

[16] R. Reiter, "A theory of diagnosis from first principles," *Artificial Intelligence*, vol. 32, no. 1, pp. 57–95, 1987.

[17] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A survey on model-based testing approaches: a systematic review," in *ACM international workshop on Empirical assessment of software engineering languages and technologies (WEASELtech)*, 2007, pp. 31–36.

[18] K. Stobie, "Model based testing in practice at microsoft," *Electronic Notes in Theoretical Computer Science*, vol. 111, pp. 5–12, 2005, proceedings of the Workshop on Model Based Testing (MBT 2004).

[19] "Web services business process execution language version 2.0," 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[20] L. Juszczyk and S. Dustdar, "Script-based generation of dynamic testbeds for SOA," in *IEEE International Conference on Web Services (ICWS)*, 2010, pp. 195–202.

[21] W. Hummer, O. Raz, O. Shehory, P. Leitner, and S. Dustdar, "Test Coverage of Data-Centric Dynamic Compositions in Service-Based Systems," in *IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST 2011)*. IEEE Computer Society, 2011, pp. 40–49.

[22] W. Hummer, O. Raz, and S. Dustdar, "Towards efficient measuring of web services api coverage," in *3rd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS), co-located with ICSE'11*, 2011, 2. Conference Proceedings. [Online]. Available: www.infosys.tuwien.ac.at/staff/hummer/docs/2011_pesos_coverage.pdf

[23] J. C. King, "Symbolic execution and program testing," *Commun. ACM*, vol. 19, no. 7, pp. 385–394, Jul. 1976. [Online]. Available: http://doi.acm.org/10.1145/360248.360252

[24] C. Ouyang, E. Verbeek, W. M. P. van der Aalst, S. Breutel, M. Dumas, and A. H. M. ter Hofstede, "Formal semantics and analysis of control flow in ws-bpel," *Sci. Comput. Program.*, vol. 67, no. 2-3, pp. 162–198, 2007.

[25] Y. Yang, Q. Tan, and Y. Xiao, "Verifying web services composition based on hierarchical colored petri nets," in *Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, ser. IHIS '05. New York, NY, USA: ACM, 2005, pp. 47–54.

[26] Y. Zheng, J. Zhou, and P. Krause, "A model checking based test case generation framework forweb services," *Information Technology: New Generations, Third International Conference on*, vol. 0, pp. 715–722, 2007.

[27] J. Z. andYongyan Zheng, , and P. Krause, "An automatic test case generation framework for web services," *Journal of Software*, vol. 2, no. 3, 2007.

[28] R. Heckel and M. Lohmann, "Towards contract-based testing of web services," *Electron. Notes Theor. Comput. Sci.*, vol. 116, pp. 145–156, Jan. 2005.

[29] G. Dai, X. Bai, Y. Wang, and F. Dai, "Contract-based testing for web services," in *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, vol. 1, july 2007, pp. 517 –526.

[30] Z. Zakaria, R. Atan, A. A. A. Ghani, and N. F. M. Sani, "Unit testing approaches for bpel: A systematic review," in *Proceedings of the 2009 16th Asia-Pacific Software Engineering Conference*, ser. APSEC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 316–322. [Online]. Available: http://dx.doi.org/10.1109/APSEC.2009.72

[31] M. H. Mustafa Bozkurt and Y. Hassoun, "Testing web services: A survey," Department of Computer Science, King's College London, Tech. Rep. TR-10-01, January 2010.

[32] Z. L. Yuan Yuan and W. Sun, "A graph-search based approach to bpel4ws test generation," in *Software Engineering Advances, International Conference on*, oct. 2006, p. 14.

[33] J. Yan, Z. Li, Y. Yuan, W. Sun, and J. Zhang, "Bpel4ws unit testing: Test case generation using a concurrent path analysis approach." in *Software Reliability Engineering, 2006. ISSRE '06. 17th International Symposium on*. IEEE Computer Society, 2006, pp. 75–84.

[34] N. Mihai, N. Simona, and W. Franz, "Does testing help to reduce the number of potentially faulty statements in debugging?" in *Testing AI Practice and Research Techniques*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6303, pp. 88–103.

[35] "Eclipse bpel designer tutorial." [Online]. Available: http://servicetechnologies.wordpress.com/exercises/

[36] I. P. Gent, C. Jefferson, and I. Miguel, "Minion: A fast scalable constraint solver," in *In: Proceedings of ECAI 2006, Riva del Garda*. IOS Press, 2006, pp. 98–102.

[37] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "End-to-end support for qos-aware service selection, binding, and mediation in vresco," *IEEE Trans. Serv. Comput.*, vol. 3, no. 3, pp. 193–205, Jul. 2010. [Online]. Available: http://dx.doi.org/10.1109/TSC.2010.20