Exploiting MDE for Platform-Independent Testing of Service Orchestrations

Lucas Leal
Institute of Computing
University of Campinas
Campinas, Brazil
ra163140@students.ic.unicamp.br

Leonardo Montecchi
Institute of Computing
University of Campinas
Campinas, Brazil
leonardo@ic.unicamp.br

Andrea Ceccarelli
Dip. Matematica e Informatica
University of Florence
Florence, Italy
andrea.ceccarelli@unifi.it

Eliane Martins
Institute of Computing
University of Campinas
Campinas, Brazil
eliane@ic.unicamp.br

***Abstract*—Service Oriented Architecture (SOA) is a common design pattern that allows building applications composed of several services. It promotes features as interoperability, scalability, and software reuse. Services composing a SOA system may evolve and change during runtime, often outside the control of the owner of the application, which makes the verification and validation processes complex. Among all the automated techniques to validate the behavior of an SOA application, is Model-Based Testing (MBT). MBT requires an accurate model of the application in order to generate suitable test cases. However, the intrinsic of a SOA application sets significant challenges to MBT effectiveness. In this paper we discuss the challenges in the testing of SOA applications, and we propose the use of Model-Driven Engineering (MDE) to improve the flexibility of testing tools. Finally, we outline our plan for realizing MDE-driven MBT within an existing online testing framework.**

***Index Terms*—model-driven engineering, model-based testing, SOA, orchestration.**

I. Introduction

The architectural paradigm of service-oriented architecture (SOA, [2]) allows building software by assembling independent, loosely coupled services, which are autonomous and platform-independent computational entities. New applications are created by connecting services, which are often controlled and managed by different entities [3]. The SOA paradigm comprises two main composition patterns: web service (WS) orchestration and WS choreography [4]. In this paper, we focus on WS orchestrations, although concepts are broadly extensible to other forms of SOA coordinations.

In an orchestration, there is one participant (the orchestrator) which controls the others services, as opposed to a choreography, which relies on protocols between the composition services to achieve its goal. Service orchestrators are often in charge of coordinating the interactions between the different services that contribute to the target application. The orchestrated services may be unaware of the fact that they take part in a more extensive process. The orchestrator is usually a web service (WS) that belongs to the SOA composition owner [4].

This work was Financed by CAPES Brazilian Federal Agency for Support and Evaluation of Graduate Education within the Ministry of Education of Brazil. This work has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 823788 "ADVANCE".

Orchestrations typically have a central coordinator controlled by the SOA composition owner, which usually facilitates the process of monitoring and controlling. Web services are integrated blind to their participation in compositions, which makes service replacement easier if the required quality of service is not attended [4].

However, orchestrations also raise essential challenges. Applications built as a combination of multiple services strictly depend on successful services interactions and proper services behavior. Further, services can change or evolve in ways not anticipated by the developers of the orchestration and service integration. Thus, it is fundamental to assure through time that the orchestration implements the intended application as expected. Identifying the proper time to execute tests on WS composition is challenging because the chance of a change occurring increases with the number of services involved, and the third parties that provide these services are not under control of the orchestration owner [5].

Testing is usually considered the most viable strategy to mitigate the above challenges and validate the system. However, since dynamic binding postpones the knowledge about the concrete services used, it is often necessary to delay part of tests to the integration and runtime phases. Consequently, runtime testing is essential to service-oriented applications: it ensures that the services, messages, interfaces, and business processes are working as expected [6]. Nevertheless, the unpredictability of third-party services update and their possible unavailability, the ever-changing application requirements, and quick cycles of application deployment create problems to the use of runtime testing on SOA applications [18].

In this paper, we advocate that Model-Driven Engineering (MDE) could be used to tackle some of these problems if applied to a test framework or process. MDE uses model abstractions of complex systems or phenomenon to simplify their management and development. Among the techniques of MDE are methods of model transformation, and model generation, which can be used to improve the interoperability of distributed systems [7].

This short paper is structured as follows: Section 2 introduces basic concepts; Section 3 presents the research idea and experiment design; Section 4 presents the selected framework, with a brief description of its components, how they interact,

and its current limitations; Section 5 shows a comparison of available MBT frameworks. Lastly, Section 6 brings the main conclusions and expectations for the proposed research.

II. Concepts

A. Model-based testing

Model-based testing (MBT) is a variant of black-box testing, which implies that the test cases are generated without knowledge of the system under test (SUT) source code. Therefore, the tests target the interfaces, giving inputs and assessing the outputs. MBT relies on models that reflect the behavior of the SUT and its environment. The models are used to generate test cases, which are executed on the original system [8].

Many algorithms can be used to generate the test case, for example, random or shortest path [16], depending on the test designers or any other that better fits the test designers needs. Since MBT can generate an infinite number of test cases from a single model, test designers often define criteria to stop the test case generation process and limit this number [16]. Usually, the test cases produced by MBT tools are high-level sequences of actions or events on the system under test (SUT), similar to high-level test sequence designed by a tester [8].

B. Model-driven engineering

MDE is a software development methodology that relies on model abstractions to represent knowledge from a specific domain. These models can be applied to system simulation, automation, code transformation, and many other applications depending only on the user's need [15]. MDE paradigm can be explained as a division of levels of conceptualization and organized in levels of implementation. The conceptualization levels are usually organized in application-level (M1), application-domain (M2), and meta-level (M3) [7].

Figure 1 presents the MDE paradigm using as an example a model transformation process, divided into conceptualizations levels from bottom up, and application levels from left to right. M1 focuses on the model definition, transformation mechanism, code and script generation. M2 is responsible for the specification of the modeling language, and the transformation rules that will guide the mapping process. M3 settles the languages and facilities (metametamodels) to which the metamodels, models, and transformations should conform [7].

C. Self-adaptabilty

A system able to modify its behavior and/or structure during runtime in response to the dynamic operational conditions, with little or no human interference, is called self-adaptive [9]. There are two main components of a Self-adaptive system: a managed system and the managing system [13]. The managing system is responsible for handling the evolution of the managed systems, and to respond to such changes [9]. Self-adaptive systems have several implementation paradigms, as the MAPE-K control loop. The MAPE-K feedback control loop is an engineering approach to realize self-adaptation, usually applied in dynamic adaptive systems, used to overcome problems related to dynamicity [13]. MAPE-K is a sequence

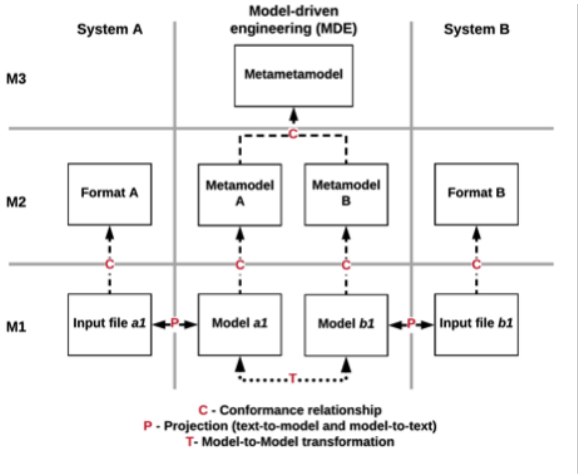

Fig. 1. MDE applied to input file transformation

of four independent computational phases: Monitor, Analyze, Plan, and Execute, that share and manipulate the same Knowledge [13].

III. Research and Experiment

A. Research idea

Among the applications of MDE there are platform-neutral abstractions, used to improve the interoperability between different languages/tools/platforms. Although MBT approaches are in the opposite situation regarding interoperability, according to Utting et al. [16], there are six dimensions of MBT approaches, all use different modeling standards, syntaxes, and maintain a certain level of independence.

MBT approaches require a model of the SUT to perform the test-case generation process. Therefore, it is critical that the model generation is correct and reflects the actual behavior of the SUT or its environment. MDE could extend the functionalities of models, enlarging the frontiers of MBT. Its techniques could enlarge the frontiers of MBT. MDE defines a metamodel that can hold information about business processes of SOA applications, thus enabling model transformations and generations according to the needs of any model-based test case generation tool. It could also store any information that testers might establish necessary, just like the adoption of Extended Finite State Machines (EFSM) did to MBT [17], MDE could enable the solution of open problems and the study of new applications of MBT.

B. Research Questions

We aim to apply the MDE paradigm into an MBT framework, guided by the following research questions.

1) **Is it possible to leverage the automation of MBT approaches using a metamodel?** What are the current challenges for the automation of MBT approaches? What are, and how can we measure, the benefits of using metamodel in an automated test process?
2) **Is it possible to use the same metamodel to model systems of alternative domains besides SOA applications?** For example, what do SOA application and

cyber-physical systems have in common? Is it possible to explore the same MBT approach to perform tests in cyber-physical systems?

C. Proposed methodology

It is necessary to design or adapt an MBT framework to work with the MDE paradigm to try to answer the research questions. An available MBT framework that would meet the requirements of the experiment is the SAMBA Framework [1].

SAMBA (Self-Adaptive Model-BAsed) generates online regression tests for SOA orchestrations, and executes them at runtime. Test cases are created from an up-to-date model of the target orchestration by a model-based test case generation tool, like Graphwalker [1].

SAMBA is a great candidate to be adapted to the MDE paradigm. The model generation process is a SAMBA framework core feature since it defines i) what kind of orchestrations can be tested, ii) the model-based testing tool that can be used, and iii) its operational parameters. Figure 2 illustrates the central role a SAMBA metamodel would have in the model conversion process. This metamodel would allow feature would make SAMBA flexible, a metamodel would allow the framework to be easily extended to further models without the need to implement the conversion to different MBT tools. Framework updates would require minimum effort, and the metamodel would improve the automation level of the testing process.

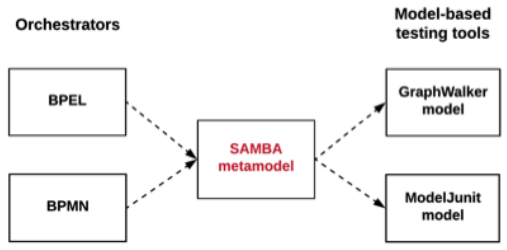

Fig. 2. SAMBA metamodel applications

Furthermore, defining a metamodel allows to store this information in an abstract notation, enabling SAMBA to generate specific models required by any model-based testing tool. Figure 3 exhibits a possible SAMBA workflow using MDE model transformation process.

Another aspect on which SAMBA would benefit from the application of MDE techniques is the test case execution. By defining a customized metamodel for the SAMBA framework, we could include the information about inputs and outputs of the web service operations in an abstract notation, thus enabling SAMBA to automate the test case execution and assessment in completely distinct environments.

The proposed research plan is to solve the limitations of SAMBA's current implementation using MDE, focusing on improving its features and making it more flexible to test different applications, thus enabling the possibility to answer the research questions.

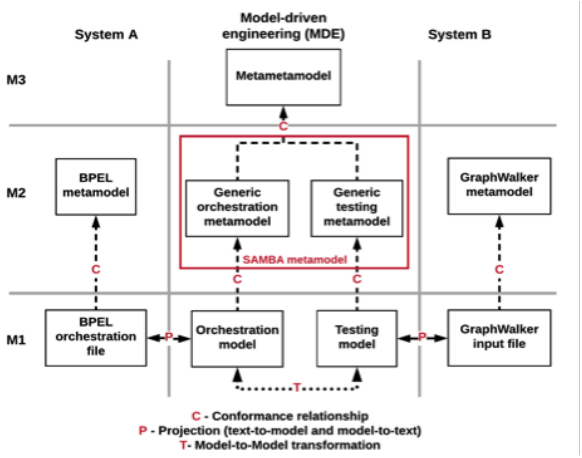

Fig. 3. Proposed SAMBA workflow with MDE

IV. Concrete Design: SAMBA Review and Criticalities

A. SAMBA overview

SAMBA (Self-Adaptive Model-BAsed) is an online test framework designed to perform service composition regression tests. SAMBA generates and executes online regression tests on orchestrated SOA applications. It uses the information available in BPEL files to extract the business sequence of an orchestrated SOA application and generates a model from it, which is used in the test case generation process [1]. SAMBA was designed to perform as a MAPE-K control loop. Figure 4 presents the framework components and their role in the control loop. SAMBA is implemented with the following functionalities:

1) update the model when required or when changes in the orchestration are detected.
2) Extract a model from the orchestration's description files.
3) Automatically generate test cases from the models.
4) Automatically execute tests from the generated test cases.
5) Automatically generate test reports.

SAMBA is composed of four main components: service assemble monitor (SAM), model generator (MG), model-based online test case generator (MOT), and test service (TS). The components are organized as MAPE-K stages and have distinct functions. SAM is responsible for monitoring the evolution of the target SOA application, and it informs the MG which orchestration changed, MG analyzes the evolution of target SOA application and generates updated models, MOT plans the test cases and updates the test report, and TS executes the runtime test [1].

B. Implementation issues

SAMBA's current implementation is working only with business process execution language (BPEL) compositions. A metamodel would enable the framework to work with different

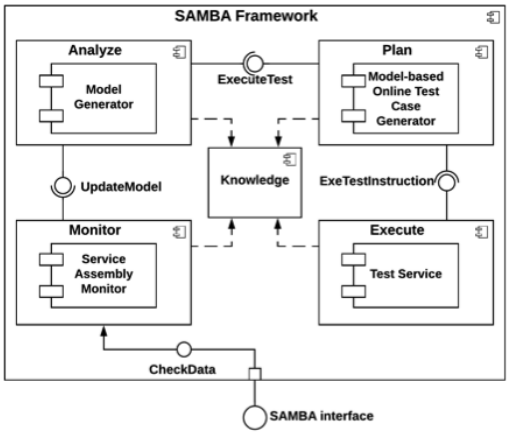

Fig. 4. SAMBA's MAPE-K components

orchestration descriptors. The model-based test tool used by the current SAMBA implementation is Grapwalker [1]. Graphwalker has built-in REST API with methods to load models, fetch data from the generated test cases, restart or abort the test case generation. The model transformation techniques of MDE would decouple SAMBA from Graphwalker, and allow the framework to use different MBT tools and asses additional aspects of an application. The TS component uses an oracle, which is responsible for storing the parameters and assertions necessary to execute and assess the generated test cases. The Oracle holds the proper inputs and outputs of each operation of every web service consumed by the SUT. MDE metamodels can be designed to store any information, making it easier accessed by the framework and updated by the users.

V. Related works

There are many works that face MBT testing in SOA, but here we review the closest to the approach we are proposing. To the best of our knowledge, no work explicitly applies MBE and meta-modeling to MBT of SOAs. The work of Bentakouk et al. (2009) [10] presents an orchestration testing framework. The service composition is translated to a symbolic transition system (STS), then a symbolic execution tree is generated from the STS. The tester inputs the coverage criteria for the test case generation process and the results are a set of execution paths, that later are executed using a test oracle. The framework described by Cao et al. (2010) [11], uses a gray-box approach, since the tester knows about the interactions between the web services that compose the orchestration, but does not have access to the services source code. The framework automatically generates and executes online tests for orchestrated services. The orchestration is converted to a timed extended finite machine (TSFM) model, which can represent time constraints and data variables. The main differences between these approaches compared to SAMBA is that neither addresses the problem of dynamicity of the SOA environment, and the frameworks are not available for the community to experiment.

VI. Conclusion

By applying the MDE methodology, we aim to improve the compatibility and integration of MBT solutions when applied to different orchestrations and MBT tools. As future works, we plan to apply our strategy along with other V&V processes such as anomaly-based error detectors, which rely on unsupervised algorithms to suit dinamicity of SOAs [19], and evaluate the possibility of improving the automated test case execution, by using the metamodel as a source for the information needed by the test case executioner to perform and assess the results.

References

[1] Leal, Lucas. "Self-adaptive model-based online testing for dynamic SOA". Master's thesis, Institute of Computing, UNICAMP (2017).
[2] Erl, Thomas. Service-oriented architecture. Vol. 8. New York: Prentice hall, 2005.
[3] Paik, Hye-young, et al. "Web Service Composition: Overview." Web Service Implementation and Composition Techniques. Springer, Cham, 2017. 149-158.
[4] Peltz, Chris. "Web services orchestration and choreography." Computer 36.10 (2003): 46-52.
[5] Belli, Fevzi, et al. "A holistic approach to modelbased testing of Web service compositions." Software: Practice and Experience 44.2 (2014): 201-234.
[6] Bozkurt, Mustafa, Mark Harman, and Youssef Hassoun. "Testing and verification in serviceoriented architecture: a survey." Software Testing, Verification and Reliability 23.4 (2013): 261-313.
[7] Topu, Okan, et al. Distributed simulation: A model driven engineering approach. Springer, 2016.
[8] Dias Neto, Arilo C., et al. "A survey on model-based testing approaches: a systematic review." Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies, in conjunction with the 22nd International Conference on Automated Software Engineering (ASE) 2007. ACM, 2007.
[9] Krupitzer, Christian, et al. "A survey on engineering approaches for self-adaptive systems." Pervasive and Mobile Computing 17 (2015): 184-206.
[10] Bentakouk, Lina, Pascal Poizat, and Fatiha Zadi. "A formal framework for service orchestration testing based on symbolic transition systems." Testing of Software and Communication Systems. Springer, Berlin, Heidelberg, 2009. 16-32.
[11] T.-D. Cao, P. Felix, R. Castanet, and I. Berrada. Online testing framework for web services. In Third International Conference on Software Testing, Verification and Validation (ICST), pp. 363-372. IEEE, 2010.
[12] Standard, O.A.S.I.S. (2007). Web services business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
[13] Arcaini, P., Riccobene, E., & Scandurra, P. (2015, May). Modeling and analyzing MAPE-K feedback loops for self-adaptation. In Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (pp. 13-23). IEEE Press.
[14] Dijkman, R. M., Dumas, M., & Ouyang, C. (2008). Semantics and analysis of business process models in BPMN. Information and Software technology, 50(12), 1281-1294.
[15] Schmidt, D. C. (2006). Model-driven engineering. COMPUTER-IEEE COMPUTER SOCIETY-, 39(2), 25
[16] Utting, M., Pretschner, A., & Legeard, B. (2012). A taxonomy of model-based testing approaches. Software Testing, Verification and Reliability, 22(5), 297-312.
[17] Anand, S., et al. (2013). An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software, 86(8), 1978-2001.
[18] Sogeti. 2018. World quality report 2017-18 9th edition (2017). https://www.sogeti.com/explore/reports/world-quality-report-2017-2018/. Retrieved Jun 10, 2019.
[19] Zoppi, T., Ceccarelli, A., & Bondavalli, A. (2019). MADneSs: a Multi-layer Anomaly Detection Framework for Complex Dynamic Systems. IEEE Transactions on Dependable and Secure Computing, page(s): 1-14.