Online Testing of Service-Oriented Architectures to detect State-based Faults

Michaela Greiler
supervised by Hans-Gerhard Gross and Arie van Deursen

Delft University of Technology, Delft, The Netherlands,
{m.s.greiler|h.g.gross|arie.vanDeursen}@tudelft.nl

Abstract. Service-oriented architectures have found their way into industry to enable better business-to-business cooperations. With this software architecture new challenges for software development and testing appeared. In this proposal we discuss the problem of testing these complex, and distributed systems in dedicated test environments. We argue that state and configuration of the production system can influence system behavior in an unexpected way, and that test environments do not reflect the final system adequately. Therefore we propose the development of an online testing method to detect state-based faults, and discuss related research challenges and solutions.

1 Introduction

A service-oriented architecture (SOA) is a software architecture that aims at reusability and interoperability by exposing functionality as loosely coupled services, that can be composed to form business processes. Testing such systems is aggravated by stakeholder separation: third party services are black-boxes for integration testers, and influence in their evolution and maintenance is not permitted. Dynamic features, like ultra-late binding or dynamic composition of services, allow a flexible and dynamic way of composing a concrete system just at runtime, but restrict the ability to test a priori. The context in which a service will be used is often unknown at the service's development time, imposing problems for service testers to predict and foresee possible requirements and usage scenarios. The limited testability of service-oriented systems imposes the need of reconsidering and redesigning traditional, and inventing new testing methods and processes, as mentioned in [5, 7, 9, 10].

On the other hand, high availability requirements force business critical systems to be continuously available, and direct them to evolve at runtime as new requirements come up. The system cannot be shutdown for the deployment process, and erroneous behavior due to software reconfiguration has to be detected while the system is operational, i.e. online. State comprises in our context information about previous executions, the concrete set of installed and active programs, as well as history of preceding configuration activities.

This proposal suggests the development of a method to identify state-based faults in online reconfiguration of service-oriented systems. Reconfiguration includes evolution of business processes, services, business logic and changes in the execution environments.

The proposal is structured as following: First, related work on SOA testing is summarized. Section 3 states the problem and formulates the research hypothesis. The related research challenges and possible solutions are discussed subsequently. The research plan is outlined in Section 5, followed by the evaluation plan in Section 6. The expected outcomes are exposed in Section 7.

2 Related Work

Influenced by the need of new testing methods and techniques for SOA, many approaches on unit testing of atomic or composite services, integration and interoperability, as well as regression testing can be found in literature [6]. We are focusing especially on related work concerning integration and online testing.

Bucchiarone et al. discuss in [4] integration testing and mention especially the lack of control of integrated services e.g., to access the service code or to put a service in test mode, and the lack of information of integrated services e.g., to generate stubs, as difficulties for effective testing. Tsai et al. [12] introduce the Coyote framework that allows to specify and execute test scenarios and cases.

Research in service-oriented testing has also looked at the field of online testing, such as by Bertolino et al. for interoperability testing [1, 2]. The problem of online testability is only briefly mentioned, with the remark that test invocations should be directed to stubs or proxies. How services can provide runtime-testability has been addressed by Brenner et al. [3]. An architecture for online testing of component-based software has been outlined in [8].

Tsai et al. propose in [11] several testability evaluation criteria for test support of service-oriented architectures. Especially service integration testability, including service composition, runtime service re-composition, service controller and adaptive controller testability evaluation criteria are of matter for the design of a SOA integration testing framework.

All approaches abstract from the complex and heterogeneous environments the services are operated in, and do not mention their influence on systems' behavior.

3 Problem Statement and Research Hypothesis

Testing of SOA systems means testing of organization-spanning systems-of-systems. Setting up an accurate test environment, that represents the final environments of all involved parties in which services are used is in most cases unfeasible. Many parts of third parties have to be stubbed or mocked, even if there is a lack in provided information. Different and heterogeneous execution environments and their configuration have to be recreated, whereas a clean initial

system state still does not reflect the production environment. Beside resource limitations, also time restrictions hinder adequate set-ups.

Complete recreation of the execution environments and their configuration for the test environments involves high costs, and even if done carefully, is often not accurate, leading to the risk that even tested software fails during online integration in the production environment.

We want to determine how the state of the services, of the business logic located behind and of the execution environments can influence the system behavior in an unexpected way. Our testing focuses on finding faults that deal with incorrect state transitions of the configuration of the connected services.

To conclude, state-based faults cannot easily be revealed during testing of complex, large, distributed SOA systems in dedicated test environments. A method for identifying state-based faults during online reconfiguration of service-oriented architectures is needed.

On account of this the research hypothesis is: "Online testing is an effective strategy for detecting state-based faults in service-oriented systems."

4 Research Challenges and Proposed Solutions

An efficient online testing method to detect reconfiguration faults has to address following challenges.

What are typical reconfiguration faults in SOAs, and which state and configuration produce those, even if software and services have been offline tested?

The main challenge is to determine which state and configuration data influence program execution in a SOA environment. This has to be narrowed down by the question: "Which information is often left out in the set-up of test environments?".

To define possible faults it has to be clear how reconfigurations of service-centric systems are propagated to local and remote systems. Possible faults have to be partitioned in those that can easily be detected during offline testing, and those that cannot. Latter faults have to be subdivided again, based on the question: "Which errors are already handled by the middleware, and which have to be handled by services and systems themselves?"

Regarding state and configuration, security and authentication policies are often not known or accurately implemented in a test environment. Many errors, like service unavailability and deployment errors, will be detected and partly solved by the middleware. During composition, and service execution, typical integration faults, like interface, data format, and communication protocol mismatches are to be expected. Further, dependency errors caused by evolution of parts of the system can occur, especially if some of the services and systems involved are stubbed or mocked in the test environment. It is important to analyze the impact software evolution has on the existing systems, either local or remote, and which layers of a SOA are affected. This can reach from changes in the business process layer, to the business logic layer and also to changes in the execution environment. Changes in the backend of a service, be it the business

logic or the configuration of the execution environment, can have rippling effects on local or even remote systems. These changes can be for instance, policies, new installed software, changes of database schema or changes of the database management system. Such modifications can cause service level agreement violations, because the network load increases, the new database management system responses slower, or access is now denied.

How should an effective online testing framework for SOA integration and system testing be designed, and what are the requirements for such a framework?

Challenging during design and development of an online testing method, are test isolation, performance, and application enhancement effort. If a system's behavior is assessed online, it has to be guaranteed that testing is isolated from the production system, and no unintended side effects appear. Test execution can only take place if no performance decreases or even unavailability of service are experienced. Enhancement costs for applying online testability mechanisms have to correspond to the positive impact accompanied with them. This implies that the tradeoff of online testing is corporeal. But how can the effectiveness of an online testing method be assessed and measured? Evaluation could be based on the capability of online testing to reveal faults in contrast to offline testing.

Information regarding the state of execution environments and infrastructures (e.g. application server, messaging bus, etc.) should support online testing to reveal faults. "But what is an accessible and uniformed way to provide state information to testing entities?"

Monitoring capabilities can be used to notify about reconfiguration changes in the system. An observer service, installed on each server involved in the SOA environment, could provide access to this information to interested and authenticated testing entities.

How can test cases, oracles, and system models be derived or generated?

In many SOA environments, abstract business models, business process models or UML documents, as well as service descriptions are present. Those represent a good foundation to derive system models, test scenarios and cases.

One technique would consider the running system to be correct and use it as test oracle. The new, adapted version of the system is tested, and the test output is compared with the output of the old, stable system. Capture and replay techniques can help to increase observability, and controllability of the testing process, because old input and output sequences are recorded and reused ad libitum.

5 Research Plan

To address the first research challenge, asking what are typical reconfiguration faults in SOA-based systems, we have implemented an online testing method providing test isolation in a case study. We could gain a first insight in state-based faults, mainly caused by missing required packages, or wrong class bindings. To develop a method for identifying state-based faults in online reconfiguration,

we have to understand better which state and configurations are relevant in SOA systems, and how these influence the testing effectiveness and accuracy. In the coming year, we plan to set up a SOA laboratory in order to examine configuration and state information significant during system reconfiguration. Based on the results we want to develop a fault taxonomy for faults caused by evolutions in distributed, heterogeneous runtime environments.

In the next year, our online testing method has to be enhanced to identify these faults, and to be applicable in industrial SOA environments (e.g., IBM product suite).

Subsequently, we will explore capabilities to reduce the application enhancement effort. This will include automatic generation of test cases, and of testability artifacts allowing runtime service assessment. With this outcome, the online testing method will be extended to an open source online testing framework, providing support for test generation and execution.

Throughout the research, the outcomes will be evaluated based on industrial case studies.

6 Evaluation Plan

The SOA laboratory is a network of distributed servers, on which services, installed on different execution environments, form actual applications, inspired by industry. Communication is handled by a centralized service registry and bus.

This laboratory will be used to conduct representative case studies, following Yin [13], to evaluate our online testing method. Update scenarios of different parts of the system will function to determine the accuracy of the test environment to prevent failures, and the fault finding capabilities of our method.

In collaboration with our industrial partner, we also want to assess the effort to prepare an application, with support of our framework, for online testing. For that, we will measure, for example, the required time and the number of necessary changes in the existing system. A comparison of effort and fault finding capabilities should indicate if online testing has a positive tradeoff.

7 Expected Outcomes

The resulting contributions of the doctoral studies include:

1. A new approach for online testing to detect state-based faults.
2. A series of fault models classifying faults that appear especially in the production environments.
3. Empirical evidence concerning the effectiveness of the proposed approach.
4. An open source tool for online test support, including generation and execution of test cases.

Our results should corroborate the hypothesis by indicating the effectiveness of online testing to detect state-based faults during system reconfiguration.

References

1. A. Bertolino, L. Frantzen, A. Polini, and J. Tretmans. Audition of web services for testing conformance to open specified protocols. In *Architecting Systems with Trustworthy Components*, pages 1–25. Springer, 2006.
2. Antonia Bertolino, Guglielmo Angelis, Lars Frantzen, and Andrea Polini. The plastic framework and tools for testing service-oriented applications. pages 106 – 139, Berlin, Heidelberg, 2009. Springer-Verlag.
3. Daniel Brenner, Colin Atkinson, Oliver Hummel, and Dietmar Stoll. Strategies for the run-time testing of third party web services. In *SOCA '07: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, pages 114–121, Washington, DC, USA, 2007. IEEE Computer Society.
4. A. Bucchiarone, H. Melgratti, S. Gnesi, and R. Bruni. Testing service composition. In *Proceedings of the 8th Argentine Symposium on Software Engineering (ASSE'07) Mar del Plata, Argentina*, pages 29–31, August 2007.
5. Gerardo Canfora and Massimiliano Di Penta. Testing services and service-centric systems: Challenges and opportunities. *IT Professional*, 8(2):10–17, 2006.
6. Gerardo Canfora and Massimiliano Penta. Service-oriented architectures testing: A survey. pages 78–105, 2009.
7. Schahram Dustdar and Stephan Haslinger. Testing of service-oriented architectures a practical approach. In *Object-Oriented and Internet-Based Technologies*, 2004.
8. Alberto González, Éric Piel, and Hans-Gerhard Gross. Architecture support for runtime integration and verification of component-based systems of systems. In *1st International Workshop on Automated Engineering of Autonomous and run-time evolving Systems (ARAMIS 2008)*, pages 41–48, L'Aquila, Italy, September 2008. IEEE Computer Society.
9. Michaela Greiler, Hans-Gerhard Gross, and Khalid Adam Nasr. Runtime integration and testing for highly dynamic service oriented ICT solutions – an industry challenges report. In *TAIC-PART '09: Proceedings of the Testing: Academic & Industrial Conference on Practice And Research Techniques*, pages 51–55, Windsor, UK, 2009. IEEE.
10. Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.
11. W. T. Tsai, Jerry Gao, Xiao Wei, and Yinong Chen. Testability of software in service-oriented architecture. In *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference*, pages 163–170, Washington, DC, USA, 2006. IEEE Computer Society.
12. W. T. Tsai, Ray Paul, Weiwei Song, and Zhibin Cao. Coyote: An xml-based framework for web services testing. In *HASE '02: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering*, page 173, Washington, DC, USA, 2002. IEEE Computer Society.
13. R. K. Yin. *Case Study Research, Design and Methods*. Sage Publications, Beverly Hills, CA, second edition, 1994.