Towards Automated Testing of Web Service Choreographies

Felipe Besson, Pedro M.B. Leal, Fabio Kon, Alfredo Goldman, Dejan Milojicic

HAL Id: inria-00585412
https://inria.hal.science/inria-00585412v2

Submitted on 27 Sep 2013

Towards Automated Testing of
Web Service Choreographies [*]

Felipe M. Besson, Pedro M. B. Leal,
Fabio Kon and Alfredo Goldman
Dept. Computer Science, Univ. São Paulo
{besson,pedrombl,kon,gold}@ime.usp.br

Dejan Milojicic
Hewlett Packard Laboratories
Palo Alto, USA
dejan@hpl.hp.com

ABSTRACT

Web service choreographies have been proposed as a decentralized scalable way of composing services in a SOA environment. In spite of all the benefits of choreographies, the decentralized flow of information, the parallelism, and multiple party communication restrict the automated testing of choreographies at design and runtime. The goal of our research is to adapt the automated testing techniques used by the Agile Software Development community to the SOA context. To achieve that, we seek to develop software tools and a methodology to enable test-driven development of Choreographies. In this paper, we present our first step in that direction, a software prototype composed of *ad hoc* automated test case scripts for testing a web service choreography.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Management, Experimentation, Verification

Keywords

Choreography testing, web services, agile methods

1. INTRODUCTION

Service-Oriented Architecture (SOA) aims at the implementation of Service-Oriented Computing by using web services as the building block of applications. Composability of services is one of the SOA principles and few approaches for composing services have been proposed. Orchestration is a centralized approach of service composition. Although

straightforward and simple, its centralized nature has scalability and fault-tolerance problems. To face this problem, Choreographies of web services have been proposed as a decentralized composition solution.

In spite of the importance and the advantages of web service compositions, the automated testing of composed services has not received the needed attention. Some tools, such as SoapUI[1], WebInject[2], and TTR (Test-The-REST) [4] have been proposed for testing atomic services. Since composed services are accessible as atomic services (from an external point of view), these tools can be used for testing compositions. Nevertheless, using such approach, both orchestrations and choreographies are taken as black-boxes, preventing the use of testing strategies such as unit and integration tests. Even though some works have been proposed [1, 3], some inherent characteristics of service compositions such as dynamicity, adaptiveness, and the impossibility to exercise third-party services in testing mode, make such testing strategies difficult to be applied.

Other approaches have been proposed for the validation and simulation of the choreography model [5]. There are still approaches such as MBT (Model-Based Testing) [2] to derive integration test cases from the exploitation of formal models. Differently from them, our approach focuses on supporting the developer in the task of writing a test suite in the spirit of agile software development. Based on customer requirements and knowledge of the internal architecture of the system, agile developers create a test suite composed of hundreds or thousands of automated tests that seek to achieve a large code coverage serving as a safety net for refactoring and code evolution. This paper describes our initial efforts to develop a software framework and supporting tool that applies automated testing techniques and strategies on running choreographies at development time.

2. SOFTWARE PROTOTYPE

Our prototype consists of *ad hoc* bash scripts for a choreography enactment, JUnit test cases for automated testing of the running choreography, and a user interaction prompt for executing the scripts and tests. To validate our prototype, we designed and implemented a simple choreography for booking a trip on OK (OpenKnowledge)[3]. The choreography participants were essentially SOAP/WSDL services and RESTful web services.

Unit tests focus on verifying the behavior of the smallest units of software. In the choreography context, services are considered the units for unit testing [1, 3]. Thus, our unit tests validate the service behavior by verifying each provided functionality. In our current prototype, to test SOAP web services, a Java SOAP client (developed using JAX-WS[4]) needs to be developed for each service endpoint (i.e., the client is specific for each endpoint). Once developed, the tests use this client to invoke the services. Thanks to the inherent flexibility of RESTful services, we developed a generic REST client (i.e., it is not restricted to a specific endpoint). Acceptance tests verify the behavior of the entire system or complete functionality. From the point of view of an end-user, the choreography is available as an atomic service. Thus, the acceptance test validates the choreography as a unit service, testing a complete functionality. In this context, this type of test is similar to the approaches of unit testing using the black-box model, and there is no need to know how the service is implemented.

Integration tests intend to solve the problems encountered when unit tested components are integrated. Their goal is to verify the unit interfaces and interactions among system components. After all services have been tested at the unit level, the approach focuses on integrating each service at a time in the choreography. Once a service is integrated, the choreography is enacted by the tester. Then, the framework verifies whether the component (service) newly integrated behaves as expected. This step is achieved by checking the messages sent by that component. For each sent message, its name, destination, and content are compared with the expected values.
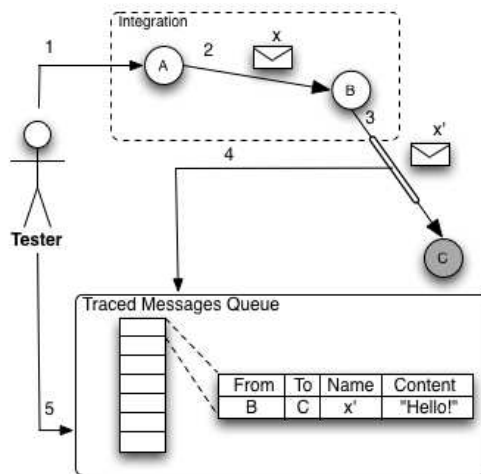

Figure 1: Integration test flow example

We developed and included in our prototype an *ad hoc* framework that implements the steps in this approach. Figure 1 shows an example of how the framework works. As depicted, the tester is integrating the A and B services. In the first step, the tester activates the choreography, invoking service A. During the choreography enactment, messages are exchanged between services A and B (step 2). Our framework collects the output messages from B, storing them in a queue (step 3 and 4). When the interaction is completed,

---

[4] http://jax-ws.java.net

the tool compares what the tester has specified with the collected data (step 5).

The complete source code of our prototype including 16 unit tests, 17 acceptance tests, and 8 integration tests can be downloaded as open source software from http://ccsl.ime.usp.br/baile/VerificationAndValidationOfChoreographies .

3. CONCLUSIONS AND ONGOING WORK

In our ongoing work, one of our goals is to develop a TDD (Test-Driven Development) methodology that will help developers and project leaders to deal with the key issues involved in testing large-scale, distributed Internet systems and will guide them in the production of effective and efficient test suites for web service choreographies. Based on the results of the current work and on the lessons learned from the prototype development, we can derive some requirements and challenges that must be faced to achieve our future goals.

3.1 Requirements and Challenges

The requirements for the proposed environment include (1) the definition or adaptation of a simple language for specifying the deployment of choreographies across the network in a reproducible way, (2) the construction of a tool for parsing specifications written in this language and enacting the choreography, and (3) the development of a framework for writing and executing automated tests for choreographies.

To scale up existing efforts for testing choreographies and to meet these requirements, we must overcome some obstacles. First of all, we have to deal with the lack of observability: since some services export only their interfaces, this prevents white-box testing in some cases. Some inherent characteristics of service compositions such as dynamicity, adaptiveness, and governance issues must also be solved to automate the integration tests. Besides, frameworks for automated testing must be generic, i.e., applicable in any choreography. Finally, some environment issues such as the non consolidation of choreography service standards, the decentralized flow of information, multiple party communication, and parallelism must be adequately treated by our future framework.

4. REFERENCES

[1] H. M. A. Bucchiarone and F. Severoni. Testing service composition. In *8th Argentine Symposium on Software Engineering (ASSE'07)*, Argentina, 2007.

[2] A. Bertolino, E. Marchetti, and H. Muccini. Introducing a reasonably complete and coherent approach for model-based testing. *Electronic Notes in Theoretical Computer Science*, 116:85 – 97, 2005.

[3] G. Canfora and M. Di Penta. Service-oriented architectures testing: A survey. In A. De Lucia and F. Ferrucci, editors, *Software Engineering*, volume 5413 of *LNCS*, pages 78–105. Springer, 2009.

[4] S. Chakrabarti and P. Kumar. Test-the-rest: An approach to testing restful web-services. In *Computation World*, 2009.

[5] L. Zhou, J. Ping, H. Xiao, Z. Wang, G. Pu, and Z. Ding. Automatically testing web services choreography with assertions. In *Proceedings of the 12th international conference on Formal engineering methods and software engineering*. Springer, 2010.