The Journal of Systems and Software 84 (2011) 655–668

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Bringing white-box testing to Service Oriented Architectures through a Service Oriented Approach

Cesare Bartolini[a], Antonia Bertolino[a,*], Sebastian Elbaum[b], Eda Marchetti[a]

[a] ISTI - CNR, Via Moruzzi 1, 56124 Pisa, Italy
[b] Computer Science and Engineering Department, University of Nebraska, Lincoln, USA

ARTICLE INFO

Article history:
Received 20 January 2010
Received in revised form 22 July 2010
Accepted 18 October 2010
Available online 18 November 2010

Keywords:
White-box testing
Coverage adequacy criteria
Testing web services
Service-Oriented Architecture

ABSTRACT

The attractive feature of Service Oriented Architecture (SOA) is that pieces of software conceived and developed by independent organizations can be dynamically composed to provide richer functionality. The same reasons that enable flexible compositions, however, also prevent the application of some traditional testing approaches, making SOA validation challenging and costly. Web services usually expose just an interface, enough to invoke them and develop some general (black-box) tests, but insufficient for a tester to develop an adequate understanding of the integration quality between the application and the independent web services. To address this lack we propose an approach that makes web services more transparent to testers through the addition of an intermediary service that provides coverage information. The approach, named Service Oriented Coverage Testing (SOCT), provides testers with feedback about how much a service is exercised by their tests without revealing the service internals. In SOCT, testing feedback is offered itself as a service, thus preserving SOA founding principles of loose coupling and implementation neutrality. In this paper we motivate and define the SOCT approach, and implement an instance of it. We also perform a study to asses SOCT feasibility and provide a preliminary evaluation of its viability and value.

1. Introduction

Usage of web services has increased dramatically in the last years (Gartner and Forrester, 2003). Different definitions can be found in the literature of what a web service is; the World Wide Web Consortium (W3C) qualifies a web service as (W3C Working Group, 2004) *a software system designed to support interoperable machine-to-machine interaction over a network* and having an interface in a machine-processable format. Such interface descriptions can be published and discovered, thus making it cost-effective for companies to integrate their own services with those developed and managed by third parties (Schroth et al., 2008). Of course web services are not necessarily used across organizations, on the contrary they are also widely used "in-house" and within corporate environments. However, the former is the situation we consider in this paper, because, as we explain below, this exposes the most difficult challenges from the tester's viewpoint.

An emerging paradigm for organizing and utilizing distributed capabilities that may be under the control of different organizations is the Service Oriented Architecture (SOA) (OASIS, 2006), whereas the sequence and conditions in which one web service invokes other web services in order to achieve its goals is referred to as an *orchestration* (W3C Working Group, 2004).

Failures in web service orchestrations, unfortunately, are common and their impact becomes more obvious and detrimental as their popularity and interdependencies increase. For example, a recent failure in Amazon's storage web service affected many companies relying on it (Amazon Discussion Forum).

For a service orchestrator, building effective tests that can detect failures in the interaction among the composed services is challenging for two reasons. First, even if best practices (Torry Harris Business Solutions) are followed by the developer to test an individual service to ensure its quality, nothing guarantees that it will then operate smoothly as part of a dynamic distributed system made of multiple orchestrated but autonomous services. Second, the orchestrator of independently developed services can usually only access their interface to derive test cases and determine the extent of the testing activity. This limited visibility means that the orchestrator has to rely heavily upon an interface whose documentation is often limited and possibly inconsistent with the true system behavior, especially with services that undergo frequent updates (Fisher et al., 2007a).

Researchers have developed several approaches to address these challenges. In particular, much work has focused on test case generation from improved service interfaces (i.e., more precise behavioral specifications) (PLASTIC Validation Framework; Sinha

* Corresponding author.
E-mail address: antonia.bertolino@isti.cnr.it (A. Bertolino).

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.10.024

and Paradkar, 2006; Xu et al., 2005), on the detection of inconsistencies between a service interface description and its behavior (Fisher et al., 2007b), on defining adequacy criteria based on the web services interactions (L. Li et al., 2008), on procedures to build scaffolding for test services in more controlled settings (Sneed and Huang, 2007), and on using the availability of multiple services as oracles (Tsai et al., 2005). One trait shared by existing test approaches is the treatment of web services as *black boxes* (Canfora and Di Penta, 2009), focusing on the external behavior but ignoring the internal structure of the services included in the orchestration. This trait follows the very nature of web services, which are meant to be *implementation neutral*. From a testing perspective, though, this is a pity. White-box approaches are in fact a well-known valuable complement to black-box ones (Pezzè and Young, 2007), as coverage information can provide an indication of the thoroughness of the executed test cases, and can help maintain an effective and efficient test suite.

To address this limitation we have conceived an approach through which services can be made more transparent to an external tester while maintaining the flexibility, dynamism and loose coupling of SOAs. Our approach enables a service orchestrator to access test coverage measures (or their byproducts) on the third-party services without gaining access to the code of those services. We refer to this enhancement as "whitening" of SOA testing (Bartolini et al., 2009) to reflect a move towards whitebox testing in terms of providing increased feedback about test executions on a service, yet the service provider remains in control of how much of the internal system structure is revealed.

Whitening is achieved through the use of dedicated services built for collecting coverage data; these services compute the coverage of the services under test, on behalf of the orchestrator. The loose coupling of the web service paradigm is not lost between the orchestrator and the developer of the provided service because the orchestrator is still unable to see anything of the service beyond the interface. In particular, the orchestrator is completely unaware of any implementation detail, and simply obtains some cumulative measures (percentages) which only reveal how much of the service the executed tests are actually using. Loss of loose coupling happens, at most, between the provided service and the coverage collecting service (of which it is reasonable to assume it is a trustworthy third party), but, even so, which and how much information is disclosed is under the control of the provider of the service under test. The approach thus blends naturally into the SOA paradigm and is called *Service Oriented Coverage Testing* (SOCT).

The added transparency from test whitening is clearly far from a "complete" white-box testing; coverage only adds a slight bit of information. However, this improvement in transparency increases testability, letting the service orchestrator gain additional feedback about how a service orchestration is exercised during validation. This feedback can then be used by orchestrators in many ways: to determine whether a coverage adequacy criterion that includes the third-party service structure has been reached; to identify tests that do not contribute as much and can be removed from a suite; or, to drive regression testing to detect possible updates in the implementation of a third-party service that might affect the behavior of their application. On the other end, third-party service providers may be enticed to provide such an extended testing interface as a way to implement continuous quality assurance checks (perhaps in association with the orchestrator), or may be required to do so as part of a service quality agreement.

Whitening SOA testing requires the design of an infrastructure that fits naturally in the service-oriented model by providing test coverage information itself as a service accessible only through a service interface. The infrastructure supporting our approach achieves that goal by requiring:

1. for the developer of a provided service, to instrument the code to enable the monitoring of the execution of target program entities, and make the relative usage information publicly available;
2. for the provider of the coverage collecting service, to track test execution results; and
3. for the service orchestrator testing the integrated application, to request testing information through a standardized published web service testing interface.

From a broader perspective, such infrastructure relies on laying down a *governance* framework to realize inter-organization testing at the orchestration level (Bertolino and Polini, 2009). Such framework encompasses the set of rules, policies, practices and responsibilities by which a complex SOA system is controlled and administered. In this paper we focus on the governance issues more closely associated with the integration testing of the orchestrated application. Of course, governance *per se* does not prevent malicious or irresponsible behavior on the service provider's part. The SOCT approach works as far as all involved stakeholders cooperate diligently (which is not different from any other collaborative engineering endeavor). The service provider, in particular, should ensure that the coverage information sent to the collecting service are precise, complete, and timely.

The very idea of SOCT has been proposed for the first time in Bartolini et al. (2008a), and elaborated in a conceptual approach in Bartolini et al. (2009). This paper extends on the latter work by revising the approach's associated definitions together with its potential applications, providing more detailed explanations of the interactions among the stakeholders, describing a full implemented instance of it, and performing a completely new assessment of its usefulness and performance through a case study. More precisely, in the next section we overview foundational related work and then, in Section 3, we present the problem domain, its motivation and main challenges. In Section 4 we define SOCT concepts and a realization scenario. In particular, the main components of the developed instance are described in Section 4.3. The case study is described in Section 5. Conclusions are drawn in Section 6.

2. Related work

In this section we overview the topic of web service testing, which is currently actively researched, as recently surveyed by Canfora and Di Penta (2009). As mentioned earlier, we focus here on SOA testing at the integration level; in particular we address the need of testing a composition of services that might have been developed by independent organizations. Some of the issues encountered in testing a composition of services are investigated by Bucchiarone et al. (2007), distinguishing between testing of orchestrations and of choreographies.

Today, the standard for service orchestration is the Business Process Execution Language (BPEL) (OASIS WSBPEL Technical Committee, 2007). Several authors have leveraged the BPEL representation for SOA testing. Although different approaches have been devised, the essential common basis in BPEL-based testing is that variants of a control flow diagram are abstracted and paths over this diagram are used to guide test generation or to assess BPEL coverage (see, e.g., Yan et al., 2006; Yuan et al., 2006). Others (including some of the authors of this paper) have also proposed to exploit BPEL data-flow information (Mei et al., 2008; Bartolini et al., 2008b).

So far, all existing approaches to SOA testing validate the services invoked in a composition as black-boxes. Indeed, the shared view is that for SOA integrators "a service is just an interface, and this hinders the use of traditional white-box coverage approaches" (Canfora and Di Penta, 2009). To the best of our knowledge, by