Chapter 4

Trends and Research Issues in SOA Validation

Antonia Bertolino
Consiglio Nazionale delle Ricerche, Italy

Guglielmo De Angelis
Consiglio Nazionale delle Ricerche, Italy

Antonino Sabetta
Consiglio Nazionale delle Ricerche, Italy

Andrea Polini
University of Camerino, Italy

ABSTRACT

Service Oriented Architecture (SOA) is changing the way in which software applications are designed, deployed and maintained. A service-oriented application consists of the runtime composition of autonomous services that are typically owned and controlled by different organizations. This decentralization impacts on the dependability of applications that consist of dynamic services agglomerates, and challenges their validation. Different techniques can be used or combined for the verification of dependability aspects, spanning over traditional off-line testing approaches, monitoring, and on-line testing. In this chapter we discuss issues and opportunities of SOA validation, we identify three different stages for validation along the service life-cycle model, and we overview some proposed research approaches and tools. The emphasis is on on-line testing, which to us is the most peculiar stage in the SOA validation process. Finally, we claim that on-line testing is only possible within an agreed governance framework.

SOA TESTING: ISSUES AND OPPORTUNITIES

In this chapter we discuss issues that testers have to face in the validation of service oriented architecture (SOA). Due to the high dynamism and the multi-organizational characteristics of SOA, the traditional testing process is not adequate anymore to guarantee an acceptable level of reliability and trust. At the same time, new opportunities arise for SOA validation. In particular, we argue in favor
of moving testing activities from the laboratory to the field, i.e., towards run-time.

This is aligned with the emerging trend of a surrounding open world (Baresi et al., 2006), in which software-intensive systems are more and more pervasive and evolve at an unprecedented pace, the boundaries between software and the external world become fuzzy, and no single organization is anymore in control of a whole system. In the future world, SOA development is going to be fully decentralized and multi-supplier, and services are going to be pervasive, dynamic, autonomic, multi-tenant. Quoting (Baresi et al., 2006), developers recognize that they must shift from the prevailing synchronous approach to distributed programming to a fundamentally more delay-tolerant and failure resilient asynchronous programming approach. Global behaviors emerge by asynchronous combinations of individual behaviors, and bindings and compositions change dynamically.

In such a context, where it will be the norm that services that are “stranger” to each other will dynamically connect and collaborate (Bertolino, 2010), how can the behavior of a SOA system be validated?

In the remainder of this section, to introduce SOA validation, we revisit some typical assumptions behind traditional testing and discuss why these are not anymore obvious in a service-based context. Then in the second section we outline a three-stage SOA testing process, including offline, admission and on-line testing, and for each stage we briefly overview issues and trends. In this overview we mention some approaches. However we warn that the references provided must be considered as an illustrative sample, somewhat biased towards the authors’ own research, and in no way the overview has to be taken as an attempt of an exhaustive literature survey. As the most novel of the three stages is on-line, in the third section we depict its potential application scenarios. Finally, in the fourth section we introduce the framework of SOA test governance that we foresee to facilitate collaborative SOA validation. The last section draws conclusions.

Software Testing Basic Assumptions

Testing of software systems is typically structured in three phases, namely unit, integration and system testing. The objective of unit testing is to check that the behavior of each single module composing the system corresponds to the intended one. Typically, the module undergoing unit testing is executed in a test environment that mimics the behavior of the other connected modules. On the other hand, the objective of integration testing is to check the correctness of module agglomerates. In this case the focus shifts to checking that some interconnected modules, that have been possibly already verified in isolation, are actually able to interoperate and provide correct results. The final step is usually system testing, which aims at checking that the system as a whole correctly provides the functionality for which it has been conceived and built.

In each of the three phases, testing activities can be carried out based on some basic assumptions that in the development of “traditional” software are so obvious as to even remain implicit. We identify three main basic assumptions:

1. Software access
2. Pre-run-time model/specification availability
3. Off-line reproducibility (or no side effects)

The software access assumption foresees that in order to check the various modules, either in isolation or in agglomerates, the tester has full access to the functionality provided by the various elements composing the system. Depending on the applied testing strategy this assumption may go even further, requiring the possibility of accessing the source code (white-box testing).

The second assumption concerns the availability, before run-time, of appropriate reference information such as a model or the specification,