(role)CAST: A FRAMEWORK FOR ON-LINE SERVICE TESTING*

Guglielmo De Angelis, Antonia Bertolino
ISTI – CNR, Via Moruzzi 1, 56124 Pisa, Italy

Andrea Polini
Computer Science Division, School of Science and Technologies, University of Camerino, 62032 Camerino, Italy

Keywords: Service oriented architecture, SOA testing, Role based access control, On-line testing.

Abstract: Engineering of service-oriented systems is still an immature discipline. Traditional software engineering approaches do not adequately fit development needs arising in this widely adopted paradigm. In particular, because of dynamic service composition, several engineering activities typically performed off-line (i.e., pre-deployment) have to be carried on also on-line (i.e., during real usage). In this paper, we present a framework called (role)CAST which supports an instantiation of the concept of on-line testing of services, for the purpose of validating their compliance to role-based service access policies.

1 INTRODUCTION

The Service-oriented Architecture (SOA) paradigm has gained significant attention within academia and companies to which it promises effective and convenient new ways of doing business. As a result several quite mature technologies are today available to implement SOA solutions, among which Web Services (WSs) are probably the most widespread. Yet a similar degree of maturity is not evident on the methodological side where several aspects of SOA development need further investigation. In particular it is necessary to rethink software engineering methodologies, so to move from the current largely design-time approach to a mainly run-time model. In this regard, testing is among the software engineering lifecycle phases which are the most affected and would certainly benefit from the possibility of being extended to run-time.

The application of traditional testing techniques typically assumes that the tester has quite detailed information and strong control over the test subject and its environment. In a SOA context these assumptions are hardly acceptable: a service under test will typically interact with other services which are outside of the tester's control. Moreover, the presence of mechanisms for run-time discovery and binding does not even permit to foresee, before run-time, who will be the partner in a service interaction. To overcome the lack of information and control, which makes off-line integration testing activities not effective, we propose to test a service composition within its real execution context: this is referred to in this paper as *on-line testing*. So on-line testing consists of proactive service invocations designed by testers, and foresees the execution of such test invocations on a service while it is engaged in serving real requests. According to this definition we do not consider activities such as run-time monitoring (Ghezzi and Guinea, 2007) or "passive testing" (Lee et al., 1997) as on-line testing, since such approaches limit themselves to passively observe how the system spontaneously behaves, without triggering any proactive action on the system itself.

We are aware though that such an approach poses big challenges, in terms of costs and potential impact, which may undermine its acceptance. Such challenges mainly account for possible testing side effects, in particular when stateful resources are considered. Nevertheless, in some contexts on-line testing can be regarded as a useful technique to increase trust among organizations interacting via deployed services. In this perspective, an encouraging trend is that the "Future Internet" will be increasingly shaped by *service federations*.

*This work has been partially supported by the European Project FP7 IP 216287: TAS^3, and by the European Project FP7 IP 257178: CHOReOS.

De Angelis G., Bertolino A. and Polini A..
(role)CAST: A FRAMEWORK FOR ON-LINE SERVICE TESTING.
DOI: 10.5220/0003340500130018
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (WEBIST-2011), pages 13-18
ISBN: 978-989-8425-51-5

A service federation is a network of services, possibly belonging to different organizations, which abide by defined rules (both technical and organizational) to be followed by all federation members to pursue one or more common goals. In other words, service federations constitute an organized world within which rules and mechanisms are put in place to *govern* the integration and the interactions among participating services. Appropriate rules should be established within a federation to make profit of on-line testing opportunities (Bertolino and Polini, 2009).

Within a federation resources are usually protected by means of a combination of *access control policies*. Such policies can be specified on any role, entity, action, or resource within the federation. In recently developed architectures (Pacyna et al., 2009; Kellomäki, 2009) it is possible to dynamically define and update such access control policies. Moreover, striving for high flexibility, service federations are pushed to develop access control solutions that are decentralized and application-independent (Pacyna et al., 2009). For example, the architectural style of service federations usually keeps the application specific logic and the authorization infrastructure as separated aspects. Thus, the interface of each federated service is dynamically bound to external services (e.g. Policy Decision Point – PDP; Policy Enforcement Point – PEP) in order to grant or to deny service access requests.

In our view, service federations provide a perfect context for applying on-line testing. Specifically, we consider that service federations supporting dynamic composition and run-time modification of the access control policies could exploit an on-line infrastructure to proactively test their evolutions. For example, services within the federation could have to regularly undergo on-line testing to assess that they (continue to) comply with their public and manifested access policies. On the other hand, access control mechanisms have to be considered in any testing strategy also at the application level.

This paper presents (role)CAST, a framework supporting on-line testing in service federations that includes authentication/authorization/identification mechanisms. The paper is organized as follows. Section 2 presents the (role)CAST framework. Section 3 reports experiences from a demo example. Related works are then summarised in Section 4. Finally, Section 5 draws some conclusions and opportunities for future work.

2 FRAMEWORK

In this section we describe the on-line testing framework we have developed within the European project TAS[3], a FP7 project investigating trustful sharing of personal information[2]. We first outline a high-level architecture (Section 2.1) that has been conceived as generic and can be reproduced in any federation; then we describe its (role)CAST implementation (Section 2.2) meant to be compatible with the TAS[3] reference instantiation.

2.1 High-level Architecture

To implement the idea of on-line testing we need to include in the federation a set of testing services that can pretend being as many service clients with a defined role, thanks to the cooperation of some trusted parties. In fact, some members of the federation should be aware of on-line testing activities and support them through appropriate extensions to the federated infrastructure. So, for example, given a service to test and a published policy, the testing service may interact with the Identity Provider (IdP)[3] asking for being recognized as a service that can play the role defined in the policy.

The high-level architecture we propose is organized in four main components: the Test Driver, the Test Robot, the Tester Backport, and the Oracle (see Figure 1). The **Test Driver** configures and runs instances of the Test Robot component for each on-line testing session. In particular, the Test Driver includes a test scheduler that activates on-line testing sessions either in an event-driven way or periodically.

The **Test Robot** loads the test cases from a repository (see Step ②). Each test case specifies the service under test, the remote operation to invoke, and the role that the Test Robot has to play. As said above, we assume that the IdP collaborates with the Test Robot via the **Tester Backport**. The latter is an abstract component that extends the interface of a generic IdP service. In particular, the IdP delegates the Tester Backport to sign role assertions (i.e. declarations that a given actor can play a given role). The communication between the Tester Robot and the Tester Backport is subject to security issues, as malicious clients may interfere with the Tester Backport trying to get signed credential. In Figure 1 at Step ③, we require that the communication between the Tester Robot and the Tester Backport is established on a secure SOAP channel. For example, we assume that on this channel

[2] http://www.tas3.eu/

[3] Simplifying, an IdP is an asserter that provides attributes representing the identity of an entity.

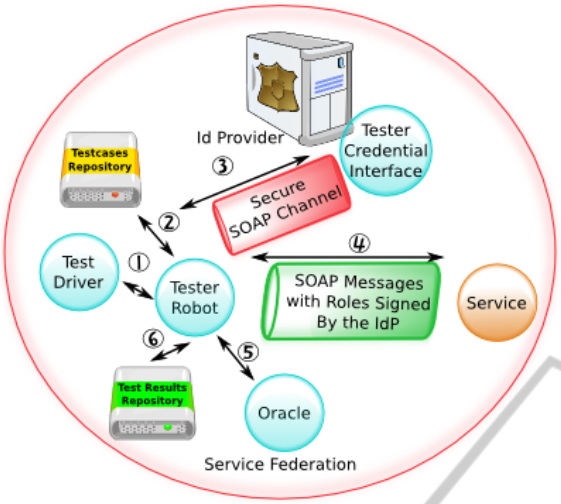

Figure 1: On-line Testing: High-Level Architecture.

the messages are digitally signed, and asymmetrically encrypted. Once the Tester Robot has collected all credentials specified within the test case, it can assemble the SOAP message and send it to the service under test. The SOAP message includes in its header the IdP-signed role that the tester is playing (see Step ④). The response returned by the service under test can be either a "functionally-correct" message or an error on the request for a resource (e.g. "access denied" message). Thus, it is forwarded to an oracle that checks if the reply of the service under test actually conforms to the expected result associated with the test case (see Step ⑤). A difference between the service reply and the expected result reveals a mismatch between how the service access policy is manifested and how this policy is actually implemented. In Figure 1, the **Oracle** is an abstract component that defines the minimal assumptions we made (partial oracle).

Note that, services might masquerade error messages for user-friendliness (e.g., they could produce a "pretty formatted" page). We are assuming that the service federation is able to unambiguously recognize "access denied" error messages without the need to delve into the semantics of the payload of the message.

Finally, the results of the test executions are stored in the Test Results Repository (see Step ⑥) for possible further inspection.

The definition of a meaningful set of test cases, and of the corresponding pass/fail decisions, certainly constitutes a complex task which depends on several factors. In scenarios in which access policies are defined at the business level, and possibly expressed in natural language, test cases derivation may be manual and the oracle may refer to an ad-hoc repository that associates test cases with expected results. When policies are expressed using computer readable format (e.g., in XACML (Moses, 2005)), the test cases and expected results can be automatically derived.

Specifically, with respect to this aspect we pursued two main approaches. On the one hand we developed a test designer supporting the tester in test case modeling using UML. The designer uses Model-driven technologies[4] in order to automatically transform such test cases into a Test Driver implementing them. On the other hand, we also support the automated derivation of a test suite starting by a XACML 2.0 service policy (Bertolino et al., 2010).

2.2 (role)CAST

The (role)CAST[5] framework (ROLE CompliAnce Service on-line Testing) is an implementation of the high-level architecture described above. In particular, (role)CAST supports on-line testing of access policies of SOAP services whose roles have been defined by means of SAML assertions.

SAML is an XML-based framework proposed by the OASIS Consortium for communicating user authentication, entitlement, and attribute information. In brief, SAML allows one party to make assertions regarding the identity, attributes, and entitlements about a subject. An assertion contains a subject of the assertion, the conditions used to validate the assertion, and assertion statements (i.e. authentication statements, attribute statements, authorization decision statements). In particular attribute statements usually contain specific identifying attributes about the subject.

The API that (role)CAST provides can be used to program the Tester Robot depicted in Figure 1, with the planned sequences of test invocations.

Each test invocation is configured specifying the URI of the service under test, the payload of the SOAP body message, the role that the Tester Robot has to play, and a key that identifies the test case within the oracle. At each invocation the Tester Robot logs the result of the test in a repository and returns the functional message replied by the service under test.

Service authentication/authorization/identification is essential in many service interaction contexts. In such cases without the possibility of assuming different client identities the tester will not be able to make any invocations to functionality for which specific roles are required. The (role)CAST API was designed to be modular and flexible, so that testers can reuse it as support to authenticating service

[4]http://www.eclipse.org/modeling/

[5]http://labse.isti.cnr.it/tools/rolecast

interactions in other testing approaches (Bertolino et al., 2008) (e.g. functional and non-functional testing). Specifically, the payload of the SOAP body message (i.e. the operation to invoke and its actual parameters) is part of the configuration of the Tester Robot. Testers can pack the SOAP payload focusing on other testing goals and then delegate to (role)CAST the execution of the invocation of the service under test.

Composing (role)CAST with other testing frameworks needs a precise definition of how the Tester Robot handles "access denied" errors. Specifically, when the Tester Robot catches a remote exception thrown by the service under test, it is important to understand if the error was due to a denied access to the resource, or to other issues. In the first case, the Tester Robot first queries the oracle, and then evaluates and logs the test result. In the second case, the Tester Robot logs that an untractable error was raised and forwards the exception to upper layers (e.g. the Tester Robot invoker). The strategy we implemented in (role)CAST is reported in (Bertolino, 2009).

3 AN APPLICATION EXAMPLE

This section describes an application of (role)CAST to a demo service federation. This scenario raises privacy and security concerns, and is one of the demonstration scenarios considered in TAS[3].

3.1 Reference Scenario

The `EmployabilityNetwork` provides on-line access to an internships management and work placement system for the students in a University. Specifically, this scenario foresees that each department of a University has a Placement Service Coordinator (PSC) that forwards applications from students to one or more Placement Provider (PP) looking for a match. `EmployabilityNetwork` functioning relies on an on-line and role-based authorization system. The configurations, the authorization mechanisms, the authorization policies, the members of the federation and the roles that each member plays within the federation can dynamically change during `EmployabilityNetwork` lifetime. For example a new PP can join `EmployabilityNetwork`, or an existing PP can be discovered and linked by a PSC of a department.

In such a scenario, it is evidently untenable to argue that each time the authorization infrastructure is subject to some reconfiguration, the system `EmployabilityNetwork` should suspend its service in order to undergo an irremissible integration testing phase. Instead, on-line testing can proactively stimulate, and dynamically validate the services within `EmployabilityNetwork`, trying to anticipate potential integration errors at run-time. Furthermore, by checking if the services in `EmployabilityNetwork` actually behave in compliance with their expected specifications, on-line testing will increase the trust and the dependability of the whole federation.

Let us consider the WSs `ComputerSciencePSC`, `BiologyPSC`, `BitIdeaPP`, and the `JobCenterPP` belonging to `EmployabilityNetwork`. Where `ComputerSciencePSC`, and `BiologyPSC` are respectively the PSC of the Computer Science and of the Biology department; `JobCenterPP` is a service interface of a general purpose job-center; while `BitIdeaPP` is the interface to the recruiting system of a company acting in ICT. Each PP accepts requests from a PSC according with the status of their subscription to the federation. For example, the PSC of the department of Computer Science can forward queries to both the PP, while the subscription of the PSC from the Biology department is only valid for querying the generic `JobCenterPP`.

When accessing a WS, each service within `EmployabilityNetwork` must provide the credentials testifying the roles it is playing. Technically, their SOAP requests will include within the SOAP header a SAML assertion in form of WS-Security Token (Monzillo et al., 2006).

3.2 Implementation and Usage

We implemented a running demo that simulates the scenario described above. We also implemented a simple module that emulates the functionality of an IdP service. We do not pretend that such a module will be used as a real IdP, nevertheless it is able to sign SAML assertions with the RSA public-key cryptography algorithm, and this is enough for our purpose.

We start by devising a set of test cases to be (periodically or following an event-driven strategy) executed within `EmployabilityNetwork` to validate role-based authorization. Figure 2 depicts the UML-based design of a test plan for `EmployabilityNetwork`. In this case, the abstraction level of the test context is intentionally high, giving emphasis to both the role of the services, and the target of the test.

The test plan schedules invocations between the four WSs. With reference to Figure 2, a UML relation tagged by the term ≪testCase≫ (i.e. a UML stereotype) specifies the configuration of a test case.

For example, Figure 3 depicts the instantiation of the tagged values for the test case of a client acting as

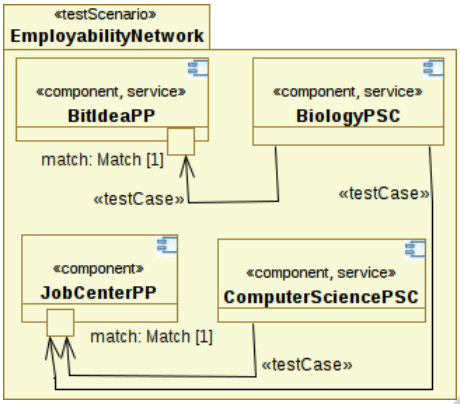

Figure 2: Test Plan for EmployabilityNetwork.

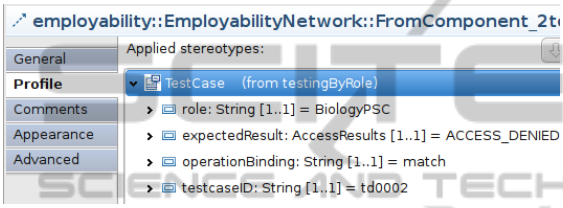

Figure 3: Test Case Configuration for BiologyPSC.

BiologyPSC, that invokes the match operation exported by the service BitIdeaPP. This test succeeds if access to the resource is denied. Furthermore, the tagged values associated with the test scenario schedules that the test plan is periodically executed (e.g. every 2 hours, or weekly, or other).

A code generator embedded in the UML test cases designer automatically compiles the test context depicted in Figure 2 into an implementation of the Test Driver component built on (role)CAST. An on-line testing session over EmployabilityNetwork can be started executing such generated component. Thus, the Test Driver acting as a service within EmployabilityNetwork, and precisely playing the role of a BiologyPSC, periodically invokes the other PPs with requests to access their functionality.

To demonstrate the usefulness of on-line testing, we played with the EmployabilityNetwork testbed by injecting malfunctions and verifying if the on-line test cases could detect these errors. For instance, we injected a bug in the service module of JobCenterPP that decides whether access requests should be granted or not (i.e. the PDP). Specifically, we have modified such PDP to provide a random response between *permit* or *access-denied*, to incoming authorization requests. Thus, for example, the PDP of JobCenterPP may erroneously recognize the ComputerSciencePSC tokens as associated to an unauthorized account. In this case, the service module of JobCenterPP that implements the decisions of the PDP (i.e. the PEP) may deny legitimate service requests.

The rationale behind this or similar bugs is twofold: on the one hand the decision process associated with management of service accesses (i.e. the JobCenterPP service) usually relies on a complex SOA that deals with roles and identities. Thus, this architecture style may suffer from integration testing issues related to run-time discovery and binding mechanisms.

On the other hand, the credentials themselves may depend on an external set of configuration data (i.e. expiration, validity, trustworthiness with respect to the party that assert the role, etc). Such set may change at runtime and unexpectedly invalidate pre-existing configurations. For example, let us assume that the access to the services is specified and managed according to policies, e.g. written in XACML (Moses, 2005). A policy conflict could occur whenever an access policy to a service changes in a way that is not anymore compatible with other authentication or privacy policies. As expected, the on-line testing session on EmployabilityNetwork revealed the inconsistency due to the injected bug. It is important to remark that, as discussed in Section 4, all the inconsistencies were revealed launching designed test cases in order to validate specified behaviours rather than by observing the system behaviour (i.e. "monitoring", or "passive testing"), thus anticipating the detection of potential errors during real service usage.

4 RELATED WORK

The very idea of continuing testing after a system has been released is not new, however while previous approaches referred to re-test after delivery for evolving software, for us on-line testing is essential to carry out useful service integration testing. As systems become more and more distributed and pervasive, several frameworks are being developed for observing system behaviour during real usage, serving different purposes, e.g., profiling (Elbaum and Diep, 2005; Orso et al., 2002) or assessing Quality-of-Service (Raimondi et al., 2008). Nevertheless such kind of frameworks limit themselves to observe how the system "spontaneously" behaves, and do not proactively validate any selected behaviour. This paper is also related to the quite active research thread on SOA testing. Due to space limitation, for existing approaches to SOA testing we refer the reader to a recent extensive survey by Canfora and Di Penta (Canfora and Di Penta, 2008).

5 CONCLUSIONS AND FUTURE WORK

SOA is characterized by specifications and technologies focusing on run-time interaction among software services. In this paper we argued in favor of on-line testing. Specifically, we considered a context in which on-line testing can more usefully prove its usefulness, namely role-based authorization in federated network services. Along this direction, we proposed a generic on-line testing framework in which services are tested with respect to their public access policy.

With respect to our next-future work, we are experimenting the (role)CAST library within the TAS^3 project, which targets full-project scale demonstrators in two domains: *e-health*, and *e-employability*. In the longer term, we intend to explore the usage of (role)CAST as an enabling tool to possibly apply other on-line testing strategies.

REFERENCES

Bertolino, A., editor (2009). *D10.2 : Trustworthiness Architecture and Proof of Concept*. The TAS^3 Consortium.

Bertolino, A., De Angelis, G., Frantzen, L., and Polini, A. (2008). The PLASTIC Framework and Tools for Testing Service-Oriented Applications. In *Proc. of ISSSE*, volume 5413 of *LNCS*, pages 106–139. Springer.

Bertolino, A., Lonetti, F., and Marchetti, E. (2010). Systematic XACML request generation for testing purposes. *Software Engineering and Advanced Applications, Euromicro Conference*, 0:3–11.

Bertolino, A. and Polini, A. (2009). SOA test governance: Enabling service integration testing across organization and technology borders. In *Proc. of WebTest*, pages 277–286. IEEE CS.

Canfora, G. and Di Penta, M. (2008). Service-Oriented Architectures Testing: A Survey. In *Proc. of ISSSE*, volume 5413 of *LNCS*, pages 78–105. Springer.

Elbaum, S. and Diep, M. (2005). Profiling deployed software: Assessing strategies and testing opportunities. *TSE*, 31(4):312–327.

Ghezzi, C. and Guinea, S. (2007). Run-time monitoring in service-oriented architectures. In Baresi, L. and Di Nitto, E., editors, *Test and Analysis of Web Services*, pages 237–264. Springer.

Kellomäki, S., editor (2009). *D2.1 : TAS^3 Architecture*. The TAS^3 Consortium.

Lee, D., Netravali, A., Sabnani, K., Sugla, B., and John, A. (1997). Passive Testing and Applications to Network Management. In *Proc of ICNP*, page 113. IEEE CS.

Monzillo, R., Kaler, C., Nadalin, A., and Hallem-Baker, P. (2006). *Web Services Security : SAML Token Profile 1.1*. The OASIS Consortium.

Moses, T. (2005). *eXtensible Access Control Markup Language Version 2.0*. The OASIS Consortium.

Orso, A., Liang, D., Harrold, M., and Lipton, R. (2002). Gamma system: Continuous evolution of software after deployment. In *Proc. of ISSTA*, pages 65–69.

Pacyna, P., Rutkowski, A., Sarma, A., and Takahashi, K. (2009). Trusted identity for all: Toward interoperable trusted identity management systems. *Computer*, 42(5):30–32.

Raimondi, F., Skene, J., and Emmerich, W. (2008). Efficient online monitoring of web-service SLAs. In *Proc. of SIGSOFT FSE*, pages 170–180. ACM.