**Proxy-Monitor: An Integration of Runtime Verification with Passive Conformance Testing**

Sébastien Salva, Clermont-Ferrand University, Aubière, France

Tien-Dung Cao, School of Engineering, Tan Tao University, Long An, Vietnam

**ABSTRACT**

This paper proposes a conformance testing method combining two well-known testing approaches, runtime verification and passive testing. Runtime verification addresses the monitoring of a system under test to check whether formal properties hold, while passive testing aims at checking the conformance of the system in the long-term. The method, proposed in this paper, checks whether an implementation conforms to its specification with reference to the ioco test relation. While passively checking if ioco holds, it also checks whether the implementation meets safety properties, which informally state that “nothing bad ever happens”. This paper also tackles the trace extraction problem, which is common to both runtime verification and passive testing. The authors define the notion of Proxy-monitors for collecting traces even when the implementation environment access rights are restricted. Then, they apply and specialise this approach on Web service compositions. A Web service composition deployed in different Clouds is experimented to assess the feasibility of the method.

**Keywords:** Clouds, Conformance testing, ioco, Monitoring, Passive Testing, Proxy-Tester, Runtime Verification, Service Composition

**INTRODUCTION**

Model-based Testing is a Software testing approach that is gaining ground in the Industry as an automatic solution to find defects in black-box implementations. Usually, deciding whether an implementation conforms to its specification comes down to checking whether a test relation holds. Such a relation defines the notion of correctness without ambiguity by expressing a comparison of observable functional behaviours. But beyond the use of formal techniques, model-based testing offers the advantage to automate some (and eventually all) steps of the testing process. Generally, the mainstream of testing is constituted by active methods: basically, test cases are constructed from the specification and are experimented on its implementation to check whether the latter meets desirable behaviours. Active testing may give rise to some inconvenient though, e.g., the repeated or abnormal disturbing the implementation.

Passive testing and runtime verification are two complementary approaches, employed to monitor implementations over a longer period of time without disturbing them. The former relies upon a monitor, which passively observes the implementation reactions, without requiring pervasive testing environments. The sequences

DOI: 10.4018/ijisi.2014040102
of observed events, called traces, are analysed to check whether they meet the specification (Miller & Arisha, 2000; Alcalde, Cavalli, Chen, Khuu, & Lee, 2004; Lee, Chen, Hao, Miller, Wu, & Yin, 2006). Runtime verification, originating from the verification area, addresses the monitoring and analysis of system executions to check that strictly specified properties hold in every system states (Leucker & Schallhart, 2009).

Both approaches share some important research directions, such as methodologies for checking test relations and properties, or trace extraction techniques. This paper explores these directions and describes a testing method, which combines the two previous approaches. Our main contributions can be summarized threefold:

1. **Combination of Runtime Verification and Ioco Passive Testing**: We propose to monitor an implementation against a set of safety properties, which express that "nothing bad ever happens". These are known to be monitorable and can be used to express a very large set of properties. We combine this monitoring approach with a previous work dealing with ioco passive testing (IdentificationRemoved, 2012). Ioco (Tretmans, 1996) is a well-known conformance test relation that defines the conforming implementations by means of suspension traces (sequences of actions and quiescence). Starting from an ioSTS (input output Symbolic Transition System) model, our method generates monitors for checking whether an implementation ioco-conforms to its specification and meets safety properties.

2. **Trace Extraction**: The trace recovery requires an open testing environment where tools, workflow engines or frameworks can be installed. Nonetheless, the real implementation environment access is more and more frequently restricted. For instance, Web server accesses are often strictly limited for security reasons. And these restrictions prevent from installing monitors collecting traces. Another example concerns Clouds. Clouds, and typically PaaS (Platform as a service) layers are virtualized environments where Web services and applications are deployed. These virtualizations of resource, whose locations and details are not known, combined with access restrictions, make difficult the trace extraction. We face this issue by using the notion of transparent proxy and by assuming that the implementation can be configured to pass through a proxy (usually the case for Web applications). But, instead of using a classical proxy to collect traces, we propose generating a formal model from a specification, called Proxy-monitor, which reflects a proxy functioning combined with the automatic detection of implementation errors.

3. The proposed algorithms also offer the advantage of performing synchronous (receipt of an event, error detection, forward of the event to its recipient) or asynchronous analysis (receipt and forward of an event, error detection) whereas the use of a basic proxy allows asynchronous analysis only. We compare these two modes and give some experimental measurements.

The paper is structured as follows: we review some related work on passive testing and runtime verification in the next Section. We provide some notations to be used throughout the paper in Section Model definition and Notations. Then, we recall the principles of runtime verification and ioco passive testing. The combination of both which leads to the Proxy-monitor model is defined in Section Combining runtime verification and Proxy-tester. Afterwards, we apply the concept of Proxy-monitor on Web service compositions. Conclusions with directions for further research are drawn in Section Conclusion.
Related Content

Non-Monotonic Modeling for Personalized Services Retrieval and Selection
[www.igi-global.com/article/non-monotonic-modeling-personalized-services/44686?camid=4v1a](www.igi-global.com/article/non-monotonic-modeling-personalized-services/44686?camid=4v1a)

Developing Logistic Regression Models to Identify Salt-Affected Soils Using Optical Remote Sensing
[www.igi-global.com/chapter/developing-logistic-regression-models-to-identify-salt-affected-soils-using-optical-remote-sensing/226402?camid=4v1a](www.igi-global.com/chapter/developing-logistic-regression-models-to-identify-salt-affected-soils-using-optical-remote-sensing/226402?camid=4v1a)
Agile Threat Assessment and Mitigation: An Approach for Method Selection and Tailoring
[www.igi-global.com/article/agile-threat-assessment-and-mitigation/144787?camid=4v1a](www.igi-global.com/article/agile-threat-assessment-and-mitigation/144787?camid=4v1a)

Integrating Compliance Management in Service-Driven Computing: Conceptual Models and Automation Architecture
[www.igi-global.com/chapter/integrating-compliance-management-in-service-driven-computing/115439?camid=4v1a](www.igi-global.com/chapter/integrating-compliance-management-in-service-driven-computing/115439?camid=4v1a)