Self-adaptive Functional Testing of Services Working in Continuously Changing Contexts

Edith Werner, Helmut Neukirchen, and Jens Grabowski
University of Göttingen, Institute for Informatics
Software Engineering for Distributed Systems Group
Lotzestr. 16-18, D-37083 Göttingen, Germany
Phone: +49 551 39-14414 Fax:+49 551 39-14415
{ewerner,neukirchen,grabowski}@informatik.uni-goettingen.de

Abstract

Web Services and ad-hoc networks are examples for systems that work in continuously changing contexts. The services provided by such systems can be tested in a laboratory environment using the standard conformance testing methodology. Testing the inter-working of services with other services in all contexts is an impossible task, because the possible usages of services may not be known or may increase due to the development of new services. Thus, errors may occur and have to be detected and handled at runtime.

Our approach to this problem is self-adaptive functional testing, which is a combination of monitoring and active testing at runtime. In order to locate an error which has been detected during monitoring, existing test cases are adapted automatically to the specific situation and applied afterwards at runtime.

Motivation

Existing approaches for testing communicating systems, like the international ISO/IEC standard 9646 *Conformance Testing Methodology and Framework* [4], are well studied, mature and have been successfully applied to service testing [2]. Single services as well as composite and distributed services can be tested that way in a laboratory by emulating the service environment.

Since testing cannot be exhaustive, it is impossible to test all possible behaviours of a system under test by conformance testing. As a consequence, interoperability tests are often performed in addition to cover different behaviours than the conformance test did. Interoperability tests are usually also conducted in a laboratory, however the environment is provided by a limited number of other implementations.

For static systems which are only used in a restricted context, these kind of tests may be regarded as adequate. However, modern services are intended to be building blocks, i.e. being used as parts of even larger services. The context in which such a service is deployed is usually not known in advance. Moreover, those services may change their own configuration dynamically at runtime. This may include the distribution of its sub-systems and the bindings of the service to external services and subscribers.

Examples for such systems are *Web Services* [3], *CORBA* systems [8] or ad-hoc networks [9]. This class of systems is hard to test, because it involves dynamically changing configurations.

While the basic functionality of such services can be tested in a laboratory using conformance and interoperability tests, all the different *choreographies*, i.e. the interactions with different users, and different *orchestrations*, i.e. the invocation of further (sub-)service providers, cannot be tested in a laboratory, because the number of adequate tests grows exponentially with the number of components involved. Thus, failures of such services occur usually at runtime in productive use, even though the service has previously passed all laboratory tests. In these cases, the failures are likely triggered by a configuration which is valid, but has not been tested before.

As a solution, we propose self-adaptive functional testing as a new test approach, which allows the test system to adapt existing test cases to the changing contexts of a service under test. The basic idea of self-adaptive functional testing will be described in the next section of this short paper. After that, the mechanisms which might be used for test case adaption are presented. Finally, we point out open issues as an outlook, and in the last section, a conclusion is given.

Self-adaptive Testing

Self-adaptive testing is the automatic adaptation of existing test cases to the current context of a service in which an error occurred. The goal of self-adaptive testing is to enable a test system to reproduce observed errors in a running service. This eases diagnostics and localisation of an error.

An Approach to Self-adaptive Testing

As a prerequisite of our approach for testing services working in changing contexts, we assume that the service under test has been tested in laboratory using standard conformance tests and that these test cases are available. Since the service under test will be running as part of a larger system, the laboratory test can be regarded as a kind of module test.

The first step in our approach is to monitor the system permanently which yields traces of the current behaviour, i.e. passive testing [5, 7]. In the best case, we can use built-in monitors, but if there are none available, separate monitors have to be integrated into the system.

In order to detect errors, the traces observed during monitoring are compared to the service's conformance tests. An error is detected if a test case fails. The trace could also be compared to the service's specification, but often this is not available.

When an error is observed, it may be classified as *internal* or *external* error. An internal error occurs within the service under test, whereas an external error occurs in another part of the monitored system. While external errors are outside the scope of the provider of the service under test, internal errors have to be investigated more detailed. To locate an internal error, it is desirable to be able to reproduce it.

In the case of an internal error, the next step is to replay the monitored trace. If the error can be reproduced that way, no adaptation is necessary and the trace leading to the error is reported to the service's maintainer.

Otherwise, the known test cases could be used to check whether the system still conforms to its specification. If that is not the case, e.g. because the observed error caused inconsistencies in the service under test, the service has to be restarted. The adaptive approach should then be executed under laboratory conditions, because the risk of driving the service under test again in an inconsistent state is high. However, if the system is robust against the error, this can even be done on the deployed system.

If the system still conforms to its specification, i.e. it passes the existing conformance tests, the existing test cases are adapted by the mechanisms described in Section in order to reestablish the context of the error. The obtained new test case is then executed on the system and its results are monitored. If the new test case reproduces the error, the adaptation algorithm terminates and the test system switches back to monitoring. The error and the test case revealing it are then reported to the service's maintainer.

Creating new test cases by adaptation is an incremental approach, we start with the most simple adaptation and gradually progress to more complex adaptations as the simple ones fail to reproduce the observed error. When the adaptation algorithm terminates successfully, the newly obtained test case could be added to the set of known test cases, e.g. if it specifies a hitherto unknown, but nevertheless valid behaviour of the service.

Adaptation Mechanisms

The adaptation of existing test cases to a context where an error has been detected may be related to configuration and behaviour. Thus, we distinguish between *configuration adaptation* and *behaviour adaptation*.

Configuration adaptation includes the activation and deactivation of points for monitoring and for testing. Additional points for testing may be necessary, if the context under investigation requires more peer services than emulated during the conformance test in the laboratory. Monitoring points may be activated, if more detailed trace information is required to locate an error which reoccurs at regular intervals.

For behaviour adaptation, adaptation mechanisms related to control flow and adaptation mechanisms related to data dependencies may be required. Hence, we further distinguish between *control flow adaptation* and *data dependency adaptation*.

Control flow adaptation includes the sequential and parallel composition of test cases. Such a composition may be applicable, if the context under investigation is a multi-context which consists of several single-contexts. The conformance test in the laboratory may have tested single-contexts in isolation by several test cases, but not their independence.

Data dependency adaptation includes the detection of data dependencies in different test cases, which have not been detected during the conformance test. Such an adaptation requires a data flow analysis based on the conformance test cases and the monitor trace leading to the error.

Open Issues

The presented work on self-adaptive functional testing has just started. Thus, we provided in this paper only the motivation and basic idea of our approach. Our future work

will concentrate on further research of adaptive testing. Especially, the following items need further investigation:

Probing effects: By adding a monitor to the deployed system, the system itself is changed. Thus, the errors observed in such an instrumented system may not be typical for the non-instrumented system. Furthermore, it has to be considered that our monitoring and active testing may have impacts on the performance of the system. In case of a severe error, which is triggered by active testing, even a breakdown of the whole system may be caused.

Security: An important aspect of modern services is security. Web Services allow e.g. to encrypt the transmitted data. For analysing the monitored data, an interception of the unencrypted data might be required. On the other hand, it has to be considered that self-adaptive testing is performed on a system which is in productive use, i.e. the processed data is real, sensitive data which may be subject of privacy restrictions.

Economic aspects: Costs and effects of using an external service have to be taken into account when trying to reproduce an error, because calls to external services due to active testing may result in charging of costs, e.g. when using a commercial information service or booking a flight. Furthermore, it has to be considered whether it is at all desirable to reproduce an error at runtime, if the error results e.g. in a denial of service.

Combination operations: The operations for recombining existing test cases into new test cases have just been sketched in this paper. For practical application, further research is necessary, e.g. which combination operators are useful, how to identify stable states in test cases and how to relate them to the logged traces.

Termination criteria: The number of test cases which can be obtained by self-adaptive testing may exceed a feasible extent. If the test system is unsuccessful in reproducing an error, it has to be decided when to stop adapting new test cases. Especially in a running system which may be already in an error condition and thus triggered the active testing, the additional load of active testing has to be taken into account.

Furthermore, we want to implement and assess this approach. As a case study, we want to test Web Services self-adaptively. For this, a suitable test architecture will be realised. In addition to the implementation under test, we intend to implement also the monitoring and injection points of the test system based on Web Services technology. Albeit it is not necessary that implementation under test and test infrastructure are implemented using the same technology, a Web Services-based test architecture yields a highly reusable test infrastructure.

Conclusion

In this work-in-progress paper, we introduced a new approach for self-adaptive functional testing which is suitable for testing services operating in continuously changing contexts. The aim of self-adaptive testing is to reproduce and classify errors which

occur in a running system. This is important to perform the right countermeasures to defeat the error.

Our self-adaptive testing approach is based on monitoring a service during operation and only applying intrusive active tests after an error has been detected. To obtain test cases which are able to reproduce the detected error, existing functional test cases are automatically adapted. Several adaptation mechanisms can be used to recombine existing test cases into new ones.

The proposed approach could also be applied to testing of component-based software, e.g. Java Beans [6]. Even though such software components are usually not used in continuously changing contexts, they are however designed to be used in unknown contexts, which is a special case of dynamically changing contexts.

Related Work

While conformance and interoperability testing are established, self-adaptive functional testing is a new area. During our literature study, we found only one paper which deals with self-adaptive testing: In [1], a so-called *frame* representation of test suites is used. This representation supports to describe initial and final states of test cases which allows to combine them into larger ones.

Future Work

The presented ideas are only a starting point. Our future work will focus on implementing a case study in order to assess our approach and investigate the open issues which have been discussed in the previous section.

References

[1] G. Adamis and K. Tarnay. Frame-Based Self-adaptive Test Case Selection. In R. Laddaga, P. Robertson, and H.E. Shrobe, editors, *Self-Adaptive Software, Second International Workshop, IWSAS 2001, Balatonfüred, Hungary, May 17-19, 2001 Revised Papers*, volume 2614 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2003.

[2] M. Anlauf. Programming service tests with TTCN. In A. Petrenko and N. Yevtuschenko, editors, *Testing of Communicating Systems*, volume 11. Kluwer, 1998.

[3] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. W3C Working Group Note, February 2004. World Wide Web Consortium.

[4] Information technology – Open Systems Interconnection – Conformance testing methodology and framework. ISO/IEC, 1994-1997. International ISO/IEC multipart standard No. 9646.

[5] D. Lee, A. N. Netravali, K. K. Sabnani, B. Sugla, and A. John. Passive Testing and its Applications to Network Management. In *International Conference on Network Protocols (ICNP'97)*. IEEE, October 1997.

[6] SUN Microsystems. Java Beans. http://java.sun.com/products/javabeans.

[7] R.E. Miller and K.A. Arisha:. Fault Identification in Networks by Passive Testing. In *Proceedings 34th Annual Simulation Symposium (SS 2001)*, pages 277–284. IEEE Computer Society, 2001.

[8] Common Object Request Broker Architecture (CORBA/IIOP), formal/2004-03-12. Object Management Group, March 2004.

[9] C. E. Perkins. *Ad Hoc Networking*. Addison-Wesley, 2001.