Applying Active Continuous Quality Control to Cross-Platfrom Conformance Testing

Johannes Neubauer[1] and Bernhard Steffen[1]

Technische Universität Dortmund, Germany
`johannes.neubauer|steffen@cs.tu-dortmund.de`

Abstract. In this paper we sketch how *Active Continuous Quality Control* (ACQC) can be adapted to *Learning-based Cross-platform Conformance Testing* (LCCT), i.e. to a method which is tailored to validate the preservation of behavioral equivalence after migration. Our method is designed to be applied e.g. after adaptations or technological switches in terms of operation system, programming language, execution environment, third-party components, optimizations, new access methods to the application, or API changes of third-party services. Technically, LCCT is based on a combination of our approach to higher-order integration of user-level test blocks with active automata learning in our learning framework *LearnLib*. Its impact has been shown by revealing migration-specific bugs typically stemming from browser-specific functionality.

Keywords: Active Learning, Model Checking, Testing, Migration, Validation, Model-Based Testing

1 Introduction

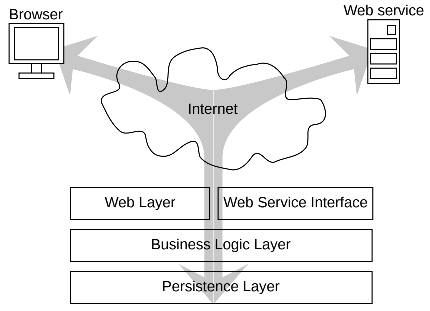


Fig. 1. The layered architecture of the enterprise application OCS

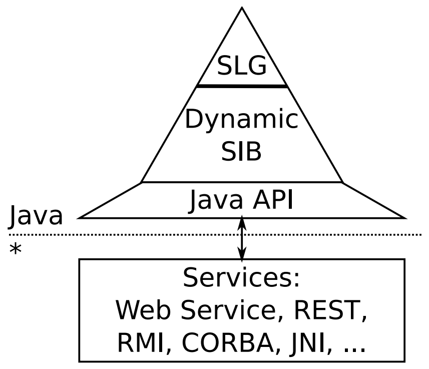


Fig. 2. The new dynamic SIB pattern of the jABC.

Modern software systems, in particular when successful, undergo continuous change and therefore require a continuous accompanying quality control at the system level. This is especially true for multi-layered component-based systems (see Fig. 1) in real-world scenarios, which can easily be changed by updating or exchanging components and their implementations [6]. Today's testing technology does not sufficiently address this problem as it requires enormous ongoing manual effort. E.g., keeping regression test suites or test models up to date is very costly, which is the major hurdle for model-based testing [30] to enter industrial practice. Recently, automata learning technology has been proposed to reduce the manual effort of the required test/model adaptation process. In [32] we presented *Active Continuous Quality Control* (ACQC), an approach that seamlessly integrates learning into the software evolution process in order to arrive at a truly continuous quality control. ACQC uses an incremental variant of active automata learning technology in order to closely monitor and steer the evolution of applications throughout their whole life-cycle with minimum manual effort. Key to this approach is the combination of the common practice of a periodic, e.g. daily, system build with a fully automatic testing process, performed and controlled via incremental active automata learning [24].

In this paper we sketch how ACQC can be adapted to *Learning-based Cross-platform Conformance Testing* [18] (LCCT), i.e. to a method which is tailored to validate the preservation of behavioral equivalence after migration [2]. Our method is designed to be applied e.g. after adaptations or technological switches in terms of operation system, programming language (e.g. reimplementation of legacy software), execution environment (e.g. application server), third-party components (e.g. database vendor, relational versus noSQL databases, or access-layer to the data-base), optimizations (e.g. caching or clustering), new access methods to the application (e.g. a RESTful API [4]), or API changes of third-party services (like Facebook, Twitter, or Google Maps). Technically, we combine our approach to higher-order integration of user-level test blocks with active automata learning in our learning framework *LearnLib* [14, 21]. Its impact has been shown by revealing migration-specific bugs typically stemming from browser-specific functionality.

In contrast to ACQC, where maintaining the stability of a common abstraction layer via a common learning alphabet during the evolution of the *System Under Learning* (SUL) as well as the cross-version reuse of information during the learning process was essential, these issues are not critical for LCCT. Rather, the third issue of ACQC, providing a correct mapping between the abstract level of learning and the concrete implementation, needs to be flexibilized for LCCT to deal with the platform specific versions of implementations. We employ the dynamic service integration feature of the jABC, our graphical application development framework [28] to elegantly solve this problem via higher-order test-block integration [19]. This allows us to realize LCCT elegantly as a learning process accompanied by a dynamic comparision of the platform-specific versions of the learned models. In essence, one could understand LCCT as an iteration between multiple learning phases which terminates when no differences are detected anymore:

– Hypothesis models for the various platform versions are learned independently.
– The differences between these hypothesis models are exploited to construct distinguishing traces which are used to trigger a new learning phase.

In this extended abstract we will focus on the mapping of abstract test symbols to concrete test-block implementations and its realization within the jABC, while we defer the presentation of the iterative LCCT process to the extended version of this paper. For a detailed elaboration of the technical details of dynamic service binding please refer to [18].

In the following, we will therefore first sketch the jABC including its pragmatics, before we address the higher-order test block generation in Sec. 3, followed by a short paragraph about model extrapolation in Sec. 4, a brief discussion of our example scenario in Sec. 5, and our conclusions in Sec. 6.

2 Extreme Model-Driven Design in the jABC

The user-level test blocks are realized in the jABC [13], a framework for service-oriented development that allows users to create services and applications easily by composing reusable building blocks into (flow-) graph structures that are both formally well-defined as well as easy to read and build. These building blocks are called *Service Independent building Blocks* (*SIBs*) in analogy to the telecommunication terminology [25], in the spirit of the service-oriented computing paradigm [11] and of the *one thing approach* [29], an evolution of the model-based lightweight coordination approach of [12] specifically applied to services.

On the basis of a large library of SIBs, the user builds models for the desired system in terms of hierarchical *Service Logic Graphs* [26] (*SLG*). SLGs are semantically interpreted as *Kripke Transition Systems* (*KTS*), a generalization of both *Kripke structures* (*KS*) and *labeled transition systems* [15] (*LTS*) that allows labels both on nodes and edges.

The service integration into the graphical process model design framework jABC is realized via dynamic service binding (see Fig. 2) that supports domain-specific (business) activities [3].

Dynamic service integration is technically achieved by directly binding a service in form of a Java method to an activity, denoted by a *dynamic SIB*. This is realized by considering services as *first-class objects* and therefore introducing a *type-safe second-order context* for exchanging services of a service graph at the parameter level, a step reminiscent of higher-order functions in functional programming languages [23]. The type-safe second-order context is a mapping from a tuple consisting of a name (e.g. ‘userController’) and type information (e.g. `UserController`) to a Java object (at runtime). Each entry in this mapping is called a *context variable*. Activities read and write values from context variables.

3 Higher-Order Test Block Integration

We apply the dynamic service integration [17] approach of the jABC framework to active automata learning. This provides us with the necessary flexibility to infer comparable models via automata learning for validating platform migrations.

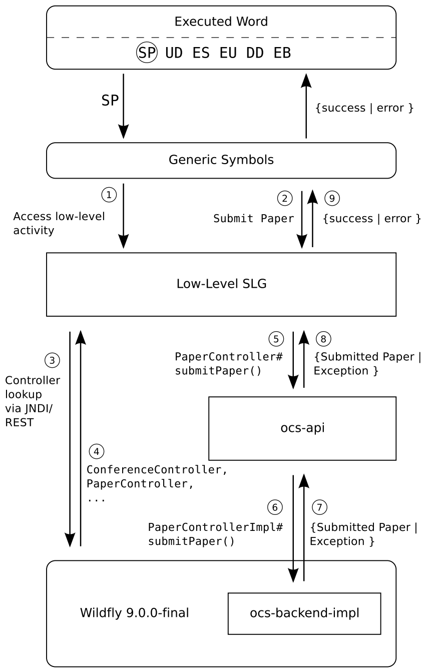
Enabling dynamic service integration does not require any implementation of adapters for mapping of activities to services, as domain-specific activities are instead modeled hierarchically as SLGs themselves on the basis of low-level services:

– Services are provided as methods of a Java class or interface (i.e. a *remote enterprise bean* (EB)[1], a RESTful- or a Selenium service respectively), abstracting from technical details. These are integrated via dynamic SIBs in low-level graphs, which are absolutely unaware of the process models.
– A fully configured instance of a subclass of the service class or interface is read from the context as input to the corresponding dynamic SIB, and the configured method is executed[2] as the control-flow reaches the activity.
– Available SIB libraries in terms of low-level graphs allow application experts to build high-level, coarse-grained, and domain-specific test blocks.

Being organized in taxonomies, these SIB libraries can easily be discovered and (re)used for building complex process models, the aforementioned service logic graphs (SLG).

4 Test-Based Model Extrapolation

In our setting, active automata learning [24] may be interpreted as a systematic test generation framework that interrogates the SUL and extrapolates an appropriate corresponding (hypothesis) model. For learning reactive systems, like e.g., a web-services or, as in our example scenario, complete web applications, *Mealy machine* models have turned out to be the target model of choice. These can efficiently be learned using the LearnLib [20, 21, 14, 10], our framework for active automata learning. LearnLib[3] provides different active learning algorithms and optimization strategies for handling counterexamples, filter techniques that allow reducing the number of executed tests through domain knowledge [1, 8], further optimizations like parallelization [7], and, since recently [9], a space optimal version, which is also particularly well-suite to support the comparison phase of LCCT.



Executed Word
SP UD ES EU DD EB
SP
{success | error }
Generic Symbols
1
Access low-level activity
2
Submit Paper
9
{success | error }
Low-Level SLG
3
Controller lookup via JNDI/ REST
4
ConferenceController, PaperController, ...
5
PaperController# submitPaper()
8
{Submitted Paper | Exception }
ocs-api
6
PaperControllerImpl# submitPaper()
7
{Submitted Paper | Exception }
Wildfly 9.0.0-final
ocs-backend-impl

Fig. 3. The low-level SLGs dynamically accesses the remote EB or RESTful API of the OCS.

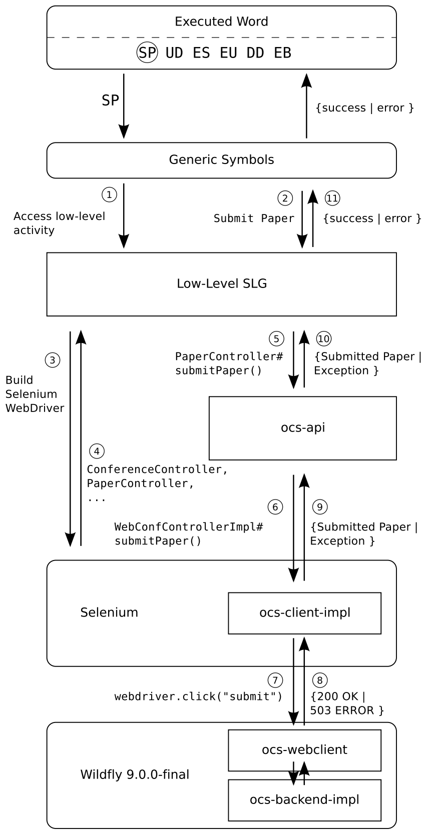


Fig. 4. The low-level SLGs dynamically accesses the web interface of the OCS via Selenium.

5 Example Scenario

The *Online Conference Service* (OCS) [16] is a multi-layered enterprise application (see Fig. 1) where, in particular, the presentation layer (*frontend*) is separated from the business logic (*backend*). The API of the backend is partitioned into controller interfaces for every type of entities modeling the domain of the OCS like a *conference*, a *paper*, or a *report*. A derived controller class implements the different actions that are possible on the respective objects, e.g., 'create a

[1] A Java EE technology to execute enterprise services remotely via *Remote Method Invocation* (RMI).

[2] We support both interpreted execution using the *Java Reflection API* [5] and full-code generation executing the method directly (i.e. type-safe) in the generated code.

[3] LearnLib is available at `http://www.learnlib.de`

user'. The used input symbols (cf. the edge labels in Fig. 5 in order to identify the abbreviations) are as follows:

`SP` *Submit Paper*: An author submits a paper to the conference but does not provide a document file. Since the number of papers in a conference is negligible for the overall workflow, we allow only one paper submission per conference.

`UD` *Upload Document*: An author uploads a document file for the previously submitted paper.

`DD` *Download Document*: The PC Chair downloads a document file of a paper.

`BD` *Bidding*: A PC Member submits a bidding for a paper.

`SA` *Special Assignment*: The PC Chair assigns a PC Member as reviewer iff the member has bided for the paper.

`SR` *Submit Report*: A reviewer submits a report for a paper.

`ES` *End Submission*: The PC Chair stops the submissions phase. From now on it is not possible to submit any new papers. This will also start the bidding phase and all PC Members will be able to submit biddings for papers.

`EU` *End Upload*: The PC Chair stops the upload phase. It is no longer possible to upload documents to papers.

`EB` *End Bidding*: The PC Chair stops the bidding phase. Members of the program committee are no longer allowed to bid for papers.

`EA` *End Assignment*: The PC Chair stops the assignment phase. From now on it is not possible to assign reviewers to papers. This will also start the review phase during which all reviewers are able to submit a report to an assigned paper.

The respective symbols will be successfully executed if all prerequisites are fulfilled. As in classical testing the membership queries have to be executed independently. In automata learning this is realized via a so-called *reset* that puts the SUL in a predefined state as all queries have to begin in the start state of the hypothesis automaton. In the case study the reset for every membership query creates a new conference and employs exactly one *PC chair*, *PC member*, and *author*. Since the number of papers in a conference is negligible for the overall workflow, we furthermore allow only one paper submission per test run.

In order to faithfully capture true web-based user behavior, we realized an alternative implementation of these controllers, denoted by *web-test controllers*, using the web-test framework Selenium [22]. These controllers, which truly mimic users operating the OCS via a browser, implement the same controller interface than their backend counterpart, and should ideally also have the same impact.

We have used our approach to dynamic service binding (cf. Sec. 3), in order to obtain variants of test blocks for the same API (controller interface) representing different implementations. This allowed us to dynamically exchange the implementation, i.e. remote EB or RESTful API (see Fig. 3) and the browser instrumentation (see Fig. 4) via higher-order service integration, and therefore to efficiently and elegantly coordinate the learning and comparison of the two platform-specific models.

Its impact has been shown by revealing migration-specific bugs typically stemming from browser-specific functionality as shown in Fig. 5. It shows an

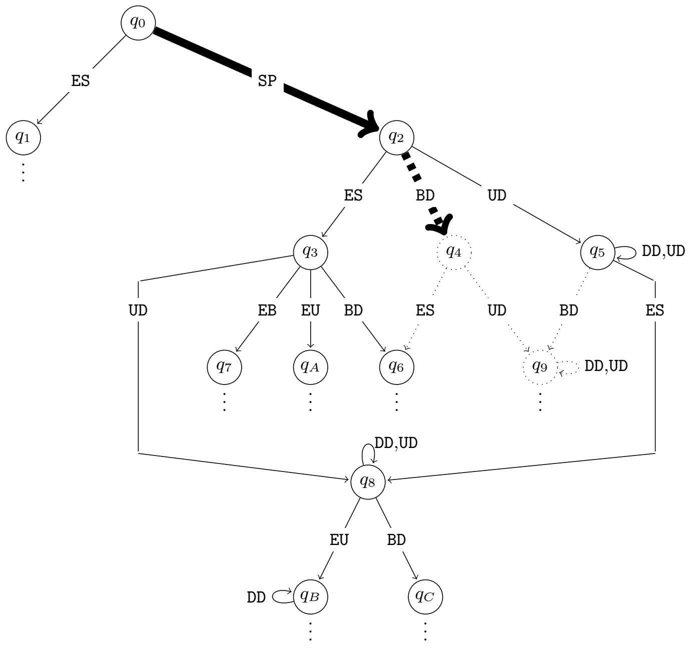


Fig. 5. Differences between learned models regarding the bidding [32].

excerpt of such an automaton of an OCS version with a bug in the security logic in the backend, which has been intercepted in the presentation layer. So it was not apparent by inspecting the presentation layer, only. For better readability we have omitted the output symbols (namely *success* and *error*) of the Mealy machine as well as the error edges (failed execution), since they are all reflexive. This is due to the rollback mechanism of the OCS, so that all erreneous executions have no impact on the state of the system.

The thick line followed by a thick dotted line in Fig. 5 shows that a member of the program committee may bid for reviewing a paper right after submission, although this should be possible after ending the submission phase (`ES`), only. This issue has also been found by our *active continuous control* (*ACQC*) approach [32]. However, in contrast to LCCT, ACQC tests the business logic only and therefore does not search for differences between frontend and backend, but it has been able to find the version which introduced the issue in the backend. Thus these are different test approaches that complement each other.

6 Conclusion and Future Work

With ACQC we have presented a novel approach to quality control that employs incremental active automata learning technology in order to periodically infer evolving behavioral automata of complex applications accompanying the development process. In this extended abstract we have presented the adaptation of ACQC to Learning-based Cross-platform Conformance Testing (LCCT), an approach specifically designed to validate successful system migration. Key to our approach is the combination of (1) adequate user-level system abstraction, (2) higher-order test-block integration, and (3) learning-based automatic model inference and comparison. LCCT employs second-order, type-safe execution semantics for (test) process models, which allows one to dynamically exchange the binding of functionality/test blocks at runtime. The impact of our approach has been illustrated along testing the conformance of the presentation layer and the business logic layer of Springer's Online Conference Service OCS in [18].

Complex multi-layered component-based systems can be subjected to rapid evolution, up to the point of exchanging components at runtime [31], a process which is called "online evolution". We are currently investigating how to extend our static approach, which focuses on managing changes that occur between system releases, to an approach capturing the effects of hierarchy [27] and self-adaptation. We think that it will be seen that our higher-order modeling approach is tailor-made for this purpose.

References

1. O. Bauer, J. Neubauer, B. Steffen, and F. Howar. Reusing system states by active learning algorithms. In *Eternal Systems*, volume 255 of *CCSE*, pages 61–78. Springer-Verlag, 2012.
2. A. De Lucia, R. Francese, G. Scanniello, and G. Tortora. Developing legacy system migration methods and tools for technology transfer. *Software: Practice and Experience*, 38(13):1333–1364, 2008.
3. M. Doedt and B. Steffen. An Evaluation of Service Integration Approaches of Business Process Management Systems. In *Software Engineering Workshop (SEW), 2012 35th IEEE*, 2012.
4. R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
5. I. R. Forman and N. Forman. *Java Reflection in Action (In Action series)*. Manning Publications Co., Greenwich, CT, USA, 2004.
6. G. T. Heineman and W. T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
7. F. Howar, O. Bauer, M. Merten, B. Steffen, and T. Margaria. The teachers' crowd: The impact of distributed oracles on active automata learning. In *ISoLA 2012*, Communications in Computer and Information Science, pages 232–247. Springer-Verlag, 2012.
8. H. Hungar, O. Niese, and B. Steffen. Domain-specific optimization in automata learning. In *Computer Aided Verification*, volume 2725 of *LNCS*, pages 315–327. Springer-Verlag, 2003.
9. M. Isberner, F. Howar, and B. Steffen. The ttt algorithm: A redundancy-free approach to active automata learning. In B. Bonakdarpour and S. Smolka, editors, *Runtime Verification*, volume 8734 of *Lecture Notes in Computer Science*, pages 307–322. Springer International Publishing, 2014.

10. M. Isberner, F. Howar, and B. Steffen. The open-source learnlib. In D. Kroening and C. S. Păsăreanu, editors, *Computer Aided Verification*, volume 9206 of *Lecture Notes in Computer Science*, pages 487–495. Springer International Publishing, 2015.
11. T. Margaria. Service is in the eyes of the beholder. *IEEE Computer*, 40:33–37, 2007.
12. T. Margaria and B. Steffen. Lightweight coarse-grained coordination: a scalable system-level approach. *STTT*, 5(2-3):107–123, 2004.
13. T. Margaria and B. Steffen. Agile it: Thinking in user-centric models. In *Leveraging Applications of Formal Methods, Verification and Validation, Proc. ISoLA 2008*, volume 17 of *Communications in Computer and Information Science*, pages 490–502. Springer Verlag, 2009.
14. M. Merten, B. Steffen, F. Howar, and T. Margaria. Next generation learnlib. In *TACAS 2011*, volume 6605 of *LNCS*, pages 220–223. Springer-Verlag, 2011.
15. M. Müller-Olm, D. Schmidt, and B. Steffen. Model-checking: A tutorial introduction. *SAS*, pages 330–354, 1999.
16. J. Neubauer, T. Margaria, and B. Steffen. Design for Verifiability: The OCS Case Study. In *Formal Methods for Industrial Critical Systems: A Survey of Applications*. John Wiley & Sons, 2011. In print.
17. J. Neubauer and B. Steffen. Second-order servification. In *ICSOB*, pages 13–25, 2013.
18. J. Neubauer and B. Steffen. Learning-based cross-platform conformance testing. 2015. In submission.
19. J. Neubauer, B. Steffen, and T. Margaria. Higher-order process modeling: Product-lining, variability modeling and beyond. *arXiv preprint arXiv:1309.5143*, 2013.
20. H. Raffelt, B. Steffen, and T. Berg. Learnlib: a library for automata learning and experimentation. In *FMICS '05*, pages 62–71. ACM, 2005.
21. H. Raffelt, B. Steffen, T. Berg, and T. Margaria. LearnLib: a framework for extrapolating behavioral models. *STTT*, 11(5):393–407, 2009.
22. Selenium. SeleniumHQ Web application testing system, visited july 2015. `http://seleniumhq.org/`.
23. P. Sestoft. Higher-order functions. In *Programming Language Concepts*, volume 50 of *Undergraduate Topics in Computer Science*, pages 77–91. Springer London, 2012.
24. B. Steffen, F. Howar, and M. Merten. Introduction to active automata learning from a practical perspective. In *Formal Methods for Eternal Networked Software Systems*, volume 6659 of *LNCS*, pages 256–296. Springer-Verlag, 2011.
25. B. Steffen and T. Margaria. Metaframe in practice: Design of intelligent network services. In *Correct System Design, Recent Insight and Advances*, volume 1710 of *LNCS*, pages 390–415. Springer, 1999.
26. B. Steffen, T. Margaria, V. Braun, and N. Kalt. Hierarchical service definition. In *Annual Review of Communication*, pages 847–856. Int. Engineering Consortium Chicago (USA), IEC, 1997.
27. B. Steffen, T. Margaria, V. Braun, and N. Kalt. Hierarchical Service Definition. *Annual Review of Communications of the ACM*, 51:847–856, 1997.
28. B. Steffen, T. Margaria, R. Nagel, S. Jörges, and C. Kubczak. *Model-Driven Development with the jABC*, volume 4383 of *LNCS*, pages 92–108. Springer Berlin/Heidelberg, 2006.
29. B. S. T. Margaria. Business process modeling in the jabc: The one-thing approach. In *Handbook of Research on Business Process Modeling*, pages 1–26. IGI Global, 2009.

30. J. Tretmans. Model-Based Testing and Some Steps towards Test-Based Modelling. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems*, volume 6659 of *Lecture Notes in Computer Science*, pages 297–326. Springer-Verlag, 2011.
31. Q. Wang, J. Shen, X. Wang, and H. Mei. A component-based approach to online software evolution: Research articles. *J. Softw. Maint. Evol.*, 18(3):181–205, May 2006.
32. S. Windmüller, J. Neubauer, B. Steffen, F. Howar, and O. Bauer. Active continuous quality control. In *Proceedings of the 16th International ACM Sigsoft symposium on Component-based software engineering*, pages 111–120. ACM, 2013.