Validating the Behavioral Equivalence of TTCN-3 Test Cases

Philip Makedonski, Jens Grabowski
Institute of Computer Science
University of Göttingen
Göttingen, Germany
Email: {makedonski, grabowski}@cs.uni-goettingen.de

Helmut Neukirchen
Faculty of Industrial Engineering,
Mechanical Engineering and Computer Science
University of Iceland
Reykjavík, Iceland
Email: helmut@hi.is

***Abstract*—Refactoring has been proven as useful means to improve the quality of source code. However, when improperly applied, it may introduce undesired changes to the observable behavior of the software. In this paper, an equivalence checking approach is presented to validate the behavior preservation after the application of refactoring in the domain of test cases specified using the *Testing and Test Control Notation Version 3* (TTCN-3). The approach is based on bisimulation and incrementally checks the observable behavior of two test cases at runtime for equivalence. The approach is implemented prototypically and sample experiments are conducted to evaluate the effectiveness of the approach.**

***Keywords*—behavior; equivalence; bisimulation; TTCN-3**

I. Introduction

Refactoring [1] has been established as a common approach for the improvement of the quality of source code. Recently, it has also found its way and is fulfilling its purpose in the domain of test languages. Zeiss et al. [2][3] provided a basis for the application of refactorings to test cases specified in the *Testing and Test Control Notation Version 3* (TTCN-3) by introducing a catalog of refactorings and tool support for their automated application. However, the application of refactorings, as well as the design of new refactorings face a major challenge—refactorings, by definition, must preserve the observable behavior of the refactored entity. Validating that this is indeed the case after a refactoring has been applied is thus essential [4], but has turned out to be a non-trivial task.

Previous work has initially suggested formal approaches based on predicate calculus [5][6], concept analysis [7], graph rewriting [4], and algebraic refinement rules [8]. These have been deemed insufficient and/or not applicable automatically [9]. Other publications [1][6] have proposed the use of specifications to measure the refactored entities against. Test suites can serve this purpose and have thus been suggested to be used as "safety nets" [1]. For the validation of the behavior preservation after refactoring of test suites, however, this will require a perspective shift—the test suites will then have to be validated against the *System Under Test* (SUT) before and after undergoing refactoring, as proposed by [10]. As intuitive as it may sound, such an approach has a limited applicability, due to the fact that generally only a single path of a test suite is covered in such a context. In [2] and [11], it was suggested that bisimulation [12][13][14][15] could be used to check the behavior of the original and the refactored test cases for equivalence.

In this paper, an approach is presented to address the issue of validating the behavior preservation of refactorings in TTCN-3. The approach seeks to show that the observable behaviors of the TTCN-3 test cases before and after refactoring are equivalent. The remainder of this paper is structured as follows: Section II contains a brief overview of relevant key concepts. Section III presents the approach in detail. Its prototypical implementation is outlined in Section IV. Section V presents a summary of the results from the sample experiments conducted to evaluate the applicability of the approach. Finally, Section VI concludes this paper and presents an outlook for future work.

II. Basic Notions

TTCN-3 [16][17] is an internationally standardized test specification and implementation language. It can be used for both functional and non-functional black-box tests on different levels. Various commercial [18][19][20][21][22][23] and in-house tools support editing TTCN-3 source code and compiling it into executable code, as well as management of test suites and campaigns.

Prior to any further discussion on how to approach the problem of validating the behavioral equivalence, notions of behavior and behavioral equivalence need to be established. Refactorings may affect the internal behavior of the entities being refactored, but must not change the observable behavior. The observable behavior of TTCN-3 test cases is defined by the interactions of the test system running the test case with the SUT and the test verdicts, which are the real output to the user. The interactions between the test system and the SUT take the form of messages being exchanged or procedure calls. The approach presented in this paper takes only message-based communication into consideration, however, the methods can be adapted and transferred to procedure-based communication as well.

A. Behavior

The observable behavior of TTCN-3 test cases can be best monitored at the *Test System Interface* (TSI). The TSI of a TTCN-3 test system consists of communication ports

that are visible to the SUT, and queues associated to these ports. TTCN-3 test cases may (and usually do) utilize multiple parallel test components, performing individual tasks simultaneously and therefore exhibiting individual parallel behaviors. The test components also have communication ports of their own, which can be connected to the ports of other test components for internal communication or mapped to the ports of the TSI for communication with the SUT. The behaviors of the test components are not directly visible to the SUT and thus do not directly constitute a part of the observable behavior of the TTCN-3 test case. It is the collective interaction of all test components with the SUT, through the TSI, as perceived from the SUT's perspective, that defines the observable behavior of a test case. Thus, this notion of behavior will be the basis for comparison and validation of the observable behavior of TTCN-3 test cases.

As it is based on the interaction model of the test case, the behavior can also be divided into atomic constituent parts, which will be referred to as events. An event is any atomic communication operation. A sequence of events constitutes a path, or a possible behavior. The union of all paths defines the complete behavior. An observable event, on the other hand, is an event that is visible to the SUT. The possible sequences of observable events define the observable behavior.

In order to achieve full coverage of the test case behavior, the behaviors of the individual test components, albeit not directly observable, have to be taken into consideration. They will not be used for comparison, but rather for guiding the test case behavior to achieve exhaustive coverage.

Given the above reasoning, there are two levels of behavior that need to be defined: test component behavior for each individual test component, which will serve behavior coverage purposes, and overall (observable) test case behavior, which will be used for comparison. The behavior of a test component can be modeled using a *Labeled Transition System* (LTS) [13], whereas the overall test case behavior needs to reflect the TSI with its ports and port queues. The LTSs used for the representation of the individual test component behaviors consist of a set of states with no specific semantic meaning and a set of labeled transitions that denote the events that can take place at a given state. The labels are direct references to the specific events. The overall observable test case behavior on the other hand is represented by queues of observable events. The queues are associated to ports, one for the in and out directions, for each port, so as to reflect the TSI from SUT's perspective.

Figure 1 outlines the two levels of behavior that need to be considered—Level 0 (L0) denotes the individual test component behaviors and Level 1 (L1) denotes the observable test case behavior at the TSI, with the test system ports and their associated queues.

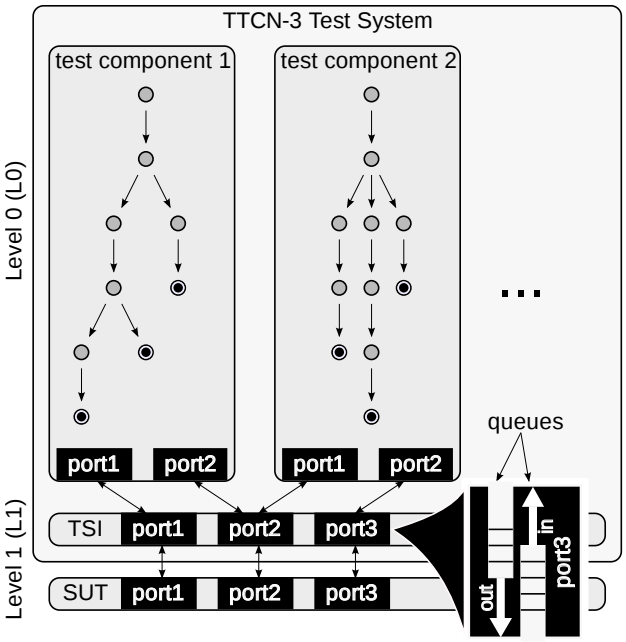

Fig. 1. Notions of behavior in TTCN-3 test systems—Level 0 (L0) denotes the individual test component behaviors and Level 1 (L1) denotes the observable test case behavior at the TSI, with the test system ports and their associated queues.

B. Equivalence

In [24], it was observed that "*the correctness of application of a refactoring rule is in the responsibility of the developer*" and that *"there is (currently) no proof system that allows to formally prove correctness–neither automatic nor interactively"*. The approach presented in this paper seeks to address these issues in the domain of TTCN-3 test cases. Informally, *"the correctness of application of a refactoring rule"* will be defined as *"equivalence in the observable behavior of a test case before and after the application of a refactoring rule"*, and the aim is to provide means to prove that the latter holds under all circumstances that can be externally induced and observed.

The observable behavior of a test case was established to be the interaction model of the test system running the test case, and the SUT, as perceived at the TSI. Equivalence in the observable behavior will then mean equivalent interaction models (at L1). The behavior of the individual test components is not relevant at this point, since it is not visible to the SUT. Moreover, different test component behaviors may still produce the same observable behavior. In fact, there are refactorings that introduce changes to the test component behaviors and the test configuration, while still preserving the overall behavior of the test system.

The presented approach considers only tests without real-time constraints–while the temporal ordering of interaction events is regarded for comparison of two test cases, the actual timing between the interaction events is disregarded.

C. Bisimulation

Bisimulation [12][13][14][15], on an informal level, is a relation that associates two (or more) systems, that behave in the same way, under all circumstances. It could be thought of as one system simulating the other and vice-versa, that

is, the systems can match each other's actions. The bisimilar systems are thus indistinguishable to an external observer, which is what a test system running a refactored test case and its counterpart running the unrefactored test case should be to an SUT.

The approach proposed in this paper does not directly check for bisimilarity. It is based on the concept of bisimulation, more specifically weak bisimulation, which, in contrast to strong bisimulation, disregards internal actions that occur between observable events, and thus resembles the problem setting more closely than strong bisimulation.

III. Approach

There are numerous challenges that an approach for the validation of refactorings must face. Exhaustive coverage and the resulting state-space explosion are among the most prominent in the general context. Managing the behavior of concurrent communicating systems is also a major challenge, particularly in the present domain. TTCN-3 specific challenges include timers, complex data types and templates, wildcard and matching mechanisms.

The proposed validation approach consists of four major parts: instrumentation, simulation, comparison, and logging. They could be thought of as generally occurring in this sequence, however, simulation, comparison, and logging occur semi-simultaneously. Instrumentation and simulation address the behavior management and exhaustive coverage challenges, as well as the TTCN-3 specific issues. The interleaved simulation and comparison on-the-fly also help addressing the state-space explosion problem, in that only small parts of the behavior representations need to be available at any given point.

Instrumentation is necessary to provide internal information from within the test case at runtime, to facilitate its simulated execution. The information ranges from the message contents and order, that is expected from the SUT, to further control and logging information. The instrumentation is applied on the source code level, by means of parse-tree manipulations, as it is most suitable for the particular context, and utilizes the native communication mechanisms of TTCN-3 to export information and provide external control of the test case execution at runtime. Instrumentation is performed as a preprocessing step, prior to the execution. In cases of data dependencies that cannot be resolved automatically, data may be supplied manually, which is then stored in a data pool for reuse.

Simulation is the core of the validation approach, as it enables the comparison and logging parts and these are built around it. Its purposes are to allow the execution of test cases, managing TTCN-3 behavior and semantics as necessary, systematically cover all behaviors that can be externally observed and induced by producing the necessary SUT responses, provide traces for comparison and logging, and circumvent the state-space explosion problem.

TTCN-3 test cases need an SUT to run against. Executing a test case by itself is therefore not reasonable. When running test cases against the SUT, the complete test case behavior is difficult to cover. Usually, only one behavioral path is covered. To achieve full coverage of a test case, the SUT needs to be steered into sending in a systematic fashion the responses necessary to drive a test case into a desired path. This may be difficult to impossible to achieve depending on the particular SUT. For a generic approach, it would be required that all SUTs obey this principle. Instead, a simulated SUT can be used, which interprets incoming stimuli, and, with the help of the instrumentation, produces the necessary responses to steer the execution of the test case into the desired behavior dynamically, at runtime. During execution, traces are generated from the simulation and fed for comparison and logging purposes to the respective entities.

Achieving exhaustive coverage and managing the state-space explosion are key aspects of the simulation. Exhaustive coverage is achieved by incremental behavior model construction and exploration at runtime. Behavior models are constructed and explored individually, on the test component level (L0). With the help of instrumented events, the reactive behavior of the test case is revealed to the simulated SUT during test case execution, resulting in a simulation configuration. During execution, a path that has not been previously covered is chosen, systematically for each test component. The selection is governed by an exhaustive backtracking algorithm operating on all the behavior models at L0. The test case is then continuously re-executed until all the paths, in all test components' behaviors, in all combinations, are covered.

The state-space necessary for the simulation is reduced to having only the current active path with the branching edges for the alternative paths actively maintained for a test component. During backtracking, the edges and nodes that have been covered are then marked and reduced.

However, to validate the behavior of a refactored test case using equivalence checking, it will still be necessary to have representations of the complete test case behavior at L1 before and after the refactoring. To minimize the amount of information necessary and the size of the representations, the equivalence checking is applied on-the-fly. This is achieved by executing the original and the refactored test case in parallel against two independent simulated SUTs which produce traces for comparison and feed them to the comparison entity. It checks the traces for equivalence at L1 and discards the processed traces. Should any discrepancies in the behavior of the two test cases occur, logging is activated, collecting information from both the comparison and the simulation to provide the necessary information for the identification and localization of the causes.

Concurrent communicating systems present a major challenge for such an on-the-fly approach. With communication flowing over multiple channels concurrently, it is difficult to establish an equivalence relation between two such systems, and even more so to check such a relation on-the-fly. TTCN-3 test systems being inherently concurrent with multiple parallel test components communicating with the SUT over single or multiple communication channels are no exception. Furthermore, the active behavior of a TTCN-3 test system cannot be

directly controlled from the outside, that is, it may send stimuli at will. Its reactive behavior, however, as far as receiving responses sent from the SUT is concerned, is where control can be exercised. In message-based communication in TTCN-3, the reactive behavior is blocking behavior, meaning when a test component is expecting a response from the SUT, its behavior is blocked until the appropriate response is received. This allows partial control over the execution progress of a test case. This also enables synchronization between the two simultaneously running test cases, when they are in stable states, i.e., all the active test components are in a blocked state. Synchronization is necessary for incremental comparison on-the-fly.

Another benefit of dynamic incremental comparison on-the-fly is that it can influence the execution of the test cases to detect and localize deep changes in behavior. Two fully independently running test cases may take different paths at an early point during their execution, generating mismatches in the behavior from that point on, even if only a single new branch was introduced or removed after that point in the behavior representation of the refactored test case at L0, which is referred to as a "deep change" in relation to that early point. Thus, behavior alignment is necessary to detect and localize such problems. Behavior alignment is the process in which one of the simulated SUTs hands over the control of its execution to the other simulated SUT. It is the simulated SUT that is "ahead", that has to surrender the control of its behavior. "Ahead" means that it has covered a sub-tree under a given branch of the behavior representation of a test component's behavior, which is still not fully covered by the other simulated SUT. This is due to the presence of additional branches in that sub-tree. During this process, the simulated SUT that surrendered the control of its execution also suspends its backtracking algorithms. The process continues until the behaviors can be realigned at the point where the control was surrendered.

There are a few peculiarities of TTCN-3 test systems that need to be taken into consideration: Reactive behavior is usually guarded by timers. Should the SUT fail to provide an appropriate response to the test system in a given timeframe, a timeout event occurs and an alternative behavior is triggered. Thus, to allow for a reliable control over the reactive behavior of a test system, it must be possible to trigger timeouts externally, from the simulated SUT. This is achieved by either substituting the timer-related events by discrete events of which the simulated SUT has a full control, or by interfering with the timing mechanisms of the test system through a custom *Platform Adapter* (PA). PAs in TTCN-3 take care of timing representation and other test platform-specific aspects. It generally depends on the particular case which approach is preferable, however, the usage of the more advanced approach that utilizes a custom PA is closer to the native operational context of TTCN-3 test cases, where a real SUT is used.

Finally, logging facilitates the localization and identification of problem areas that introduced changes to the observable behavior of the test case. It is not to be confused with the *Test Logging Interface* (TLI) as defined in the TTCN-3 standards [25]. The standard TLI can be used to provide an additional logging layer, but cannot fulfill all the needs of the presented approach.

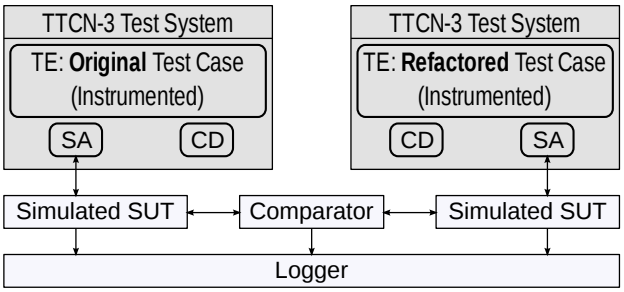

Fig. 2. Architectural overview of the validation approach: two test systems running the original and the refactored test cases (both of which are instrumented) are connected to two independent simulated SUTs; the simulated SUTs are connected to a comparator; both the simulated SUTs and the comparator are connected to a logger.

Figure 2 depicts an architectural overview of the approach. The approach builds around the standard TTCN-3 test system architecture. Two test systems running the original and the refactored test cases (both of which are instrumented) are taken and connected to two independent simulated SUTs. The illustration of the test systems features only the relevant functional entities: the *Test Executable* (TE) which runs the TTCN-3 test cases, the *Codec* (CD) which provides encoding and decoding functionalities for the transformation of TTCN-3 data to and from SUT compatible data, and the *System Adapter* (SA) which implements the base means of communication between the TE and the SUT. The simulated SUTs are then connected to a comparison entity, or a comparator. It takes care of comparison and synchronization of the SUTs. In case behavior alignment is necessary, the comparator also mediates the directed control between the simulated SUTs. Finally, both the simulated SUTs and the comparator are connected to the logging entity, or the logger, and provide logging information for post-validation analysis over their logging interfaces.

IV. Implementation

Other tools already provide bisimulation implementations, for example the *Construction and Analysis of Distributed Processes* (CADP) [26][27] toolbox. However, no tool currently supports TTCN-3 directly, meaning that the relevant behavior representations still need to be extracted and transformed into suitable input for these tools. Thus, complete behavior representations are still necessary. Furthermore, the tools also impose certain limitations on their input and lack specific functionalities that are, needed, for example, in order to make behavior alignment possible. This makes them less suitable for the needs of this project.

The approach has been prototypically implemented to test and validate its applicability. Its implementation builds upon existing tools for TTCN-3. The conceptual entities for instrumentation, simulation, comparison and logging have been

implemented as standalone distributed tools for flexibility. This design enables the distributed application of the approach for improved efficiency. The instrumentation tool builds upon the infrastructure provided by the *TTCN-3 Refactoring and Metrics Tool* (TRex) [2][3]. The simulation tool consists of a test system adapter as a bridge between the simulated SUT and the test system running the test case, a simulation client and a simulation server. While the approach covers most of the relevant features of TTCN-3, its implementation of the instrumentation and the simulation currently covers a subset of the language and currently implements only a part of the approach, as described in detail in [28][29]. It aims to cover the basic TTCN-3 features necessary to run a simple test case. Although sufficient for the validation of most refactorings applied to simpler test cases, support for the advanced features of TTCN-3 needs to be realized to enable the simulation of real TTCN-3 test cases that utilize most of these advanced features.

V. Results

Several case studies were conducted to evaluate the applicability of the approach. The case studies concentrated on validating the behavior equivalence (or lack thereof) of individual refactorings applied to custom designed test cases, rather than large scale application of refactorings to existing publicly available test suites. There are several reasons for his:

- At this stage, it has been more interesting to study the application of different types of refactorings in detail, rather than obtaining statistical knowledge over applying the same refactoring multiple times.
- As mentioned in Section IV, the prototypical implementation supports only the basic features of TTCN-3 to enable the validation of simple test cases. Industrial-size test suites utilize many of the more advanced features, which are not immediately relevant for refactorings or the validation approach as it is, however, they are necessary for the proper execution and simulation of the test cases. This will be subject to change in the implementation.

The case studies were based on detecting the correct and incorrect application of six common TTCN-3 refactorings: *Extract Altstep*, *Replace Altstep with Default*, *Replace Template with Modified Template*, *Parametrize Template*, *Prefix Imported Declarations*, and *Extract Function*, all taken from [2]. The case studies have been systematically analyzed and documented in [28], in terms of possible issues that may occur if the particular refactoring is applied incorrectly, how would these issues manifest themselves, and how would the implementation of the proposed approach handle these issues.

The prototypical implementation of the approach was able to successfully identify direct changes to the observable behavior resulting from various errors that can be made when a refactoring is applied. General causes for unintentional changes in the behavior include failure to pass the correct parameters and/or set the proper parameter passing modes when extracting code segments, or failure to notice differences in code parts that are to be extracted. More specific causes are related to the usage context of certain refactorings, as well as to the domain of TTCN-3. Incorrectly applied refactorings can cause changes ranging in severity from hard to notice data-integrity corruption to major control flow changes. Changes to the observable behavior resulting from these causes include the introduction of new events (and the corresponding branches), removed events (missing branches), changed message contents and changed test verdicts. In cases where a new event occurred deep in the behavior representation tree structure, behavior alignment was successfully applied to identify and localize the cause.

The case studies demonstrated that the approach and its implementation were able to deal with the state-space explosion problem for the sample test cases used. To achieve complete coverage, the implementation executed the test cases over and over, until all the behaviors that can be externally induced and observed were covered. However, only the data necessary for the validation of the current execution is necessary at any point during the validation process. This helps to manage the state-space explosion in that no complete representation of the observable behavior is necessary for the validation of behavior preservation. The demands are then arguably shifted from the space domain to the execution time domain. Given that execution of test cases is necessary for the collection of the data necessary to produce the behavior representations in the given context anyway, the required time for the validation of a refactored test case was approximately the same as the time required to cover all the behaviors that can be externally induced and observed, with only marginal increase due to the comparison steps.

A major observation is that the approach presented is useful for the detection of immediate changes to the observable behavior, that can be directly observed in a test case at its development state during the validation.

VI. Conclusion and Future Work

In this paper, an approach for the validation of refactored TTCN-3 test cases is presented. The approach uses equivalence checking and is loosely based on bisimulation. To circumvent the state-space explosion problem, it is applied on-the-fly to two simultaneously executing test cases. The approach has been proven useful for the detection of immediate changes to the observable behavior of test cases by means of a sample case study.

The application scope of the approach can also expand to more generic scenarios, such as detecting whether a given change was just a behavior preserving optimization or refactoring or rather a bugfix or a new feature. Another alternative application scenario might involve the comparison of different TTCN-3 execution environments which are running the same test case, to validate that the execution environments are interchangeable in the context of the given test case. The latter is an issue for tools implementing the TTCN-3 standard.

While the methodology supports most of the relevant TTCN-3 features, the implementation of the approach is still at the prototype level. Thus, the effectiveness of the approach

on larger test cases, e.g. huge standardized TTCN-3 test suites, remains to be studied, once support for advanced TTCN-3 features used by these test suites is provided in the implementation. Therefore, the extension of the implementation to support these advanced TTCN-3 constructs and features is of particular importance. Also, only a subset of the available refactorings has been studied. The analysis of other refactorings, as well as larger data sets, may provide further insights.

Other future considerations include partial and in-place validation, capture and replay features, and partial ordering of events, to name a few. Partial and in-place validation could be used to isolate the refactored areas and validate only the relevant contexts. Capture and replay features may prove very useful for quick re-validation of particular scenarios of interest, exact reproduction of a particular execution, or offline validation. Partial ordering of events is a minor enhancement that may be of interest in some application contexts, such as testing shared media. Such an enhancement, however, may also require a revision of the currently utilized notion of equivalence.

Another point of interest may be providing performance and overhead measurements. However, such measurements will hardly be generally applicable and will only provide relative data. The expectation is that they are strongly dependent upon the complexity of the data and the test configuration, as well as on the behavior of the individual test components and on the operational environment.

REFERENCES

[1] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[2] B. Zeiss, "A Refactoring Tool for TTCN-3," Master's thesis, Institute for Informatics, ZFI-BM-2006-05, ISSN 1612-6793, Center for Informatics, University of Göttingen, Mar. 2006.

[3] H. Neukirchen, B. Zeiss, J. Grabowski, P. Baker, and D. Evans, "Quality assurance for TTCN-3 test specifications," *Software Testing, Verification and Reliability (STVR)*, vol. 18, no. 2, pp. 71–97, Jun. 2008.

[4] T. Mens, S. Demeyer, B. D. Bois, H. Stenten, and P. V. Gorp, "Refactoring: Current research and future trends," *Electr. Notes Theor. Comput. Sci*, vol. 82, no. 3, 2003.

[5] W. F. Opdyke, "Refactoring Object-Oriented Frameworks," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL, USA, 1992. [Online]. Available: citeseer.ist.psu.edu/article/opdyke92refactoring.html

[6] D. B. Roberts, "Practical Analysis for Refactoring," Ph.D. dissertation, University of Illinois, 1999.

[7] G. Snelting and F. Tip, "Reengineering Class Hierarchies Using Concept Analysis," IBM T.J. Watson Research Center, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA, Tech. Rep. RC 21164(94592)24APR97, 1997. [Online]. Available: citeseer.ist.psu.edu/snelting98reengineering.html

[8] M. L. Cornélio, "Refactorings as Formal Refinements," Ph.D. dissertation, Universidade Federal de Pernambuco, Brasil, Mar. 2004.

[9] L. A. Tokuda, "Evolving Object-Oriented Designs with Refactorings," Ph.D. dissertation, University of Texas at Austin, 1999.

[10] A. van Deursen, L. Moonen, A. van den Bergh, and G. Kok, "Refactoring Test Code," in *Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001)*, M. Marchesi and G. Succi, Eds., May 2001.

[11] H. Neukirchen, "Re-Usability in Testing," Presentation, TAROT Summer School 2005 at the Institut National des Télécommunications, Paris, Jun. 2005.

[12] L. Aceto, A. Ingólfsdóttir, K. G. Larsen, and J. Srba, *Reactive Systems: Modelling, Specification and Verification*. New York, NY, USA: Cambridge University Press, 2007.

[13] R. Milner, *A Calculus of Communicating Systems*, ser. Lecture Notes in Computer Science (LNCS). Springer, 1980, vol. 92.

[14] ——, *Communicating and mobile systems: the π-calculus*. New York, NY, USA: Cambridge University Press, 1999.

[15] D. Park, "Concurrency and Automata on Infinite Sequences," in *Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings*, ser. Lecture Notes in Computer Science (LNCS), P. Deussen, Ed., vol. 104. Springer, 1981, pp. 167–183.

[16] ETSI, "European Standard (ES) 201 873-1 V3.4.1 (2005-08): The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language," European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, also published as ITU-T Recommendation Z.140, 2008.

[17] C. Willcock, T. Deiß, S. Tobies, S. Keil, F. Engler, and S. Schulz, *An Introduction to TTCN-3*. John Wiley & Sons, Ltd, 2005.

[18] "Danet TTCN-3 Toolbox," http://www.danet.com/en/it-services/testing/, [18 April 2009].

[19] "Elvior MessageMagic," http://www.elvior.ee/messagemagic, [18 April 2009].

[20] "Métodos y Tecnología Exhaustif/TTCN," http://www.mtp.es/content/view/114/146/lang,en/, [18 April 2009].

[21] "OpenTTCN Oy OpenTTCN Tester for TTCN-3," http://www.openttcn.com/products/tester, [18 April 2009].

[22] "Telelogic Tester," http://www.telelogic.com/products/tester/index.cfm, [18 April 2009].

[23] "Testing Technologies TTworkbench," http://www.testingtech.com/products/ttworkbench.php, [18 April 2009].

[24] J. Philipps and B. Rumpe, "Roots of Refactoring," in *Tenth OOPSLA Workshop on Behavioral Semantics. Tampa Bay, Florida, USA, October 15, 2001.*, K. Baclavski and H. Kilov, Eds. Northeastern University, 2001.

[25] ETSI, "European Standard (ES) 201 873-6 V3.4.1 (2005-08): The Testing and Test Control Notation version 3; Part 6: TTCN-3 Test Control Interface," European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, also published as ITU-T Recommendation Z.145, 2008.

[26] H. Garavel, R. Mateescu, F. Lang, and W. Serwe, "CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes," in *Computer Aided Verification, Berlin, Germany, July 3-7, 2007, Proceedings*, ser. Lecture Notes in Computer Science (LNCS), W. Damm and H. Hermanns, Eds., vol. 4590. Springer, 2007, pp. 158–163.

[27] INRIA/VASY, "Construction and Analysis of Distributed Processes (CADP) Home Page," http://www.inrialpes.fr/vasy/cadp/.

[28] P. Makedonski, "Equivalence Checking of TTCN-3 Test Case Behavior," Master's thesis, Institute for Informatics, ZFI-MSC-2008-16, ISSN 1612-6793, Center for Informatics, University of Göttingen, Nov. 2008.

[29] ——, "Code Instrumentation for the Equivalence Checking of TTCN-3 Test Case Behavior," Institute for Computer Science, Georg-August-University of Göttingen, Tech. Rep., 2008.