Concatenating Sequence-Based Requirements in Model-Based Testing with State Machines

Stephan Weißleder
Fraunhofer-Institute FIRST
stephan.weissleder@first.fraunhofer.de

Dehla Sokenou
GEBIT Solutions
dehla.sokenou@gebit.de

Abstract: Model-based testing is a promising technique for quality assurance. In this paper, we present an approach to tackle two challenges in model-based testing: the traceability to functional requirements and the automatic derivation of test cases that represent typical behavior. Our approach is based on the assumption that functional requirements describe typical sequences of behavior. The requirements sequences can be mapped to sequences in a system behavior model and can be used for automatic test generation. In this paper, we choose UML sequence diagrams to describe requirements behavior and UML state machines to describe the complete system behavior. We present the approach, new coverage criteria, the prototype implementation *ConSequence*, and first experiences using an ATM case study.

1 Introduction

Tests compare a system under test (SUT) to a specification. In model-based testing (MBT), this specification is a model. Model-based testing techniques have been investigated for several years [Bin99, BJK05, UL06]. MBT is a promising technique with several advantages over traditional testing such as requirements formalization, which can lead to early detection of inconsistencies, the automation of test design, which usually implies a significant decrease of test design costs, the reduced test maintenance costs, and the objective application of coverage criteria on the model level.

There are, however, still several open issues. One of the most important ones is the traceability of requirements in behavioral test models. An intuitive approach to allow for traceability is to annotate model elements with references to requirements [UL06, page 131]. Putting these information on requirements and system behavior in one model suffers from two draw-backs: The model becomes overloaded and the integration of requirements in the test model has to be done manually. Instead, we propose to use separate models for requirements and system behavior and to link them automatically during test generation. In this paper, we focus on sequence diagrams of the Unified Modeling Language (UML) [Obj09] as representations of functional behavior defined by requirements and on UML state machines as the representation of the system behavior. One sequence diagram can be mapped to several transition sequences of a state machine. These transition sequences have different contexts defined by the current variable assignments, the start state, and the hitherto executed path. The identification and selection of these contexts is a major testing issue.

Mostly, coverage criteria drive the automatic test generation in MBT. We are, however, aware of only one coverage criterion based on requirements, which is called *All-Requirements* [UL06] and requires only one test for each requirement. In this paper, we present an approach to concatenate transition sequences of requirements for automatic test generation. We drive test generation by newly defined requirements-based coverage criteria.

Our paper is structured as follows. In Section 2, we present the related work. We present the approach in Section 3, corresponding coverage criteria in Section 4, a prototype in Section 5, and a case study in Section 6. In Section 7, we conclude the paper, discuss the threats to validity, and present the intended future work.

2 Related Work

Testing is the primary means to detect failures of a SUT. Extensive work has been done in this area [Bin99, BJK05, UL06]. There are several approaches to integrate requirements into test generation from state machines. In [UL06, page 131], the authors propose to reference requirements from single model elements - the requirements are covered if the referencing model elements are. The tool Conformiq Designer [Con] allows to attach the keyword *requirements* to single transitions to reference requirements. In contrast, we assume that requirements are not represented as single model elements, but as interaction sequences. The Microsoft SpecExplorer [Mic10] allows for defining slices of a model for test generation that can also contain references to requirements. In contrast to our approach, however, SpecExplorer does not allow for automatically combining such slices based on corresponding coverage criteria. Furthermore, our approach is based on the definition and satisfaction of test goals [Wei10], which allows for combining test generation with other test generation tools on the test goal level [FSW08].

Many coverage criteria that are usually applied in MBT are focused on the structure of state machines. Coverage criteria for requirements in MBT are very rare. We are only aware of very fundamental criteria like *All-Requirements* [UL06], which is defined to be satisfied if all model elements that reference requirements have been covered. In contrast, we propose coverage criteria that are aimed at the concatenation of manually defined interaction sequences for requirements to drive automatic test generation.

State machines and sequence diagrams have been used for automatic test generation: For instance, Abdurazik and Offutt [OA99] generate test cases from state machines. Nebut et al. [NFTJ03] derive test cases from contracts such as use cases and sequence diagrams. In contrast to that, we focus on combining both diagrams to generate test cases. It is straight-forward that sequence diagrams can describe single transition sequences of a state machine. Bertolino et al. [BMM05] combine both diagrams to derive "reasonably" complete test models to achieve early results for partially modeled systems. Sokenou [Sok06] and Nagy [Nag04] showed furthermore that the state machine's start states to execute sequence diagrams can be very important. We extend these approaches by identifying transition sequences instead of start states that are matching to sequence diagrams. This paper is an extension of our work presented in [SW09] and can be understood as a test

counterpart of sequence-based specification as presented in [PP03]. In addition to this work, we defined coverage criteria for these model combinations, present the prototype implementation ConSequence, and report on first experiences in a case study.

There are different approaches to automatic test generation from state machines. These approaches comprise the use of model checkers [GH99], constraint solvers [AS05], or guided path search [Wei]. All of these approaches are able to generate arbitrary transition sequences. The issue with these transitions sequences is that they seldom represent typical user interaction. In contrast to that, our test generation approach is focused on mapping manually defined functional requirements to transition sequences and on combining these transition sequences to longer ones. As a result, we are able to create complex test cases that consist of typical user interaction sequences. In conformance to some of the mentioned approaches, we apply the constraint solver Choco [The09] to check the validity of transition sequences and generate concrete test cases.

3 Approach

As mentioned above, a fundamental issue in MBT is the tracing from model elements to requirements. Interaction sequences to describe requirements can be executed in several contexts. We think that approaches that comprise these information in just one model would lead to an overloaded model. For this reason, we focus on separate models for requirements and system behavior.

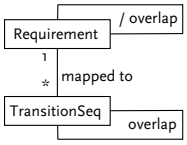

Figure 1: Meta model for mapping relations of requirements and transition sequences.

First, we describe a meta model to express relations of requirements (*Requirement*) and transition sequences (*TransitionSeq*) depicted in Fig. 1: Each requirement can be mapped to transition sequences (in different contexts). If the transition sequences of different requirements overlap each other, then the corresponding requirements are also overlapping. The three tasks of our approach are described in the following. A more detailed description is presented in [WS10a].

Mapping. The task of the mapping is to find the contexts of the requirements in the system model, i.e. to map each requirement to transition sequences in the system model. State machine transitions are triggered by incoming events. Thus, only the incoming messages for the object described by the state machine are extracted in order from the sequence diagram and mapped to transition sequences that are triggered by the same sequence of events (identified by events' names and contexts). A transition sequence ts is a sequence

of state changes, i.e., transitions t_x from a state machine, e.g., for which the target state of t_{x-1} is the source state of t_x.

Overlap Identification. For describing overlapping transition sequences, we define the following terms for all transition sequences *ts1* and *ts2*: The *start state* of any ts1 is the source state of the first transition in ts1. The *end state* of any ts1 is the target state of the last transition in ts1. TS1 is a *prefix*/*postfix* of ts2 iff ts2 starts/ends with the sequence of transitions defined in ts1 and optionally contains subsequent/preceding transitions. The sequence ts1 *is part of* ts2 iff the transitions in ts1 occur in ts2 at any place in the same order, i.e. ts2 consists of the concatenation of a prefix of ts2, ts1, and a postfix of ts2.

Based on these definitions, we define that two transition sequences *overlap* iff a) the start state of one transition sequence is equal to the end state of the other, b) there is a non-empty prefix of one transition sequence that is equal to a non-empty post-fix of the other, or c) one transition sequence is part of the other. The matching transitions that imply the overlap are called the *overlap*. All mapped transition sequences are related to each other if they overlap. The corresponding requirements are also considered overlapping if the mapped transition sequences do. Overlapping transition sequences and requirements are represented in the meta model (see Fig. 1). Transition instances can directly overlap - the corresponding association between requirements is derived from this association.

Concatenation and Test Generation. Based on the overlap relations between pairs of transition sequences, we can create an *overlap graph* covering all mapped transition sequences. Two overlapping transition sequences are concatenated by creating a new sequence that first contains all transitions of the preceding transition sequence and then all transitions of the subsequent transition sequence without using the overlap of both transition sequences twice. Concatenating non-overlapping transition sequences is done via finding a path on the overlap graph from one transition sequence to the other and concatenating all contained overlapping transition sequences. We are aware that the success of the presented approach depends on the quality of the given requirements. For instance, parts of the state machine may be not covered by requirements and, consequently, pairs of non-overlapping transition sequences cannot be concatenated using the overlap graph. In such cases, concatenation has to be done by applying other techniques, e.g. shortest path.

Test generation based on concatenating transition sequences consists of several steps: 1) Concatenating all required transition sequences in the desired order. 2) Concatenating the resulting transition sequence with the outgoing transition of the initial node and a transition sequence that starts at the target state of the initial state's outgoing transition. The resulting transition sequence is a connected path of the state machine that starts at the initial state and describes a possible system behavior that covers the desired transition sequences and requirements, respectively. 3) Validating the resulting path, i.e. it has to be checked that all constraints regarding guard conditions and effects on the contained transitions are valid.

4 Coverage Criteria

In this section, we describe coverage criteria for our approach of driving test generation based on requirements integration and concatenation.

All-Requirements is satisfied if for each requirement, at least one of the mapped transition sequences is covered by a test case.

All-Sequences is satisfied if for each requirement, all mapped transition sequences have been covered by test cases at least once.

All-Requirements-Pairs is satisfied if for each pair of requirements, at least one pair of the mapped transition sequences is covered in the order of the requirements.

All-Sequences-Pairs is satisfied if for each pair of requirements, all corresponding transition sequence pairs are covered in the order of the requirements.

All-Sequences-Twice is satisfied if for each requirement, all mapped transition sequences have been covered by test cases at least twice.

The subsumption hierarchy for the presented coverage criteria is shown in Fig. 2. The coverage criterion All-Sequence-Pairs subsumes All-Sequence-Twice by definition. Likewise, All-Sequence-Twice subsumes All-Sequences and All-Requirements-Pairs subsumes All-Requirements. Since All-Requirements only has to cover one mapped transition sequence but All-Sequences has to cover all of them, All-Sequences subsumes All-Requirements. The same holds for All-Requirements-Pairs and All-Sequence-Pairs.

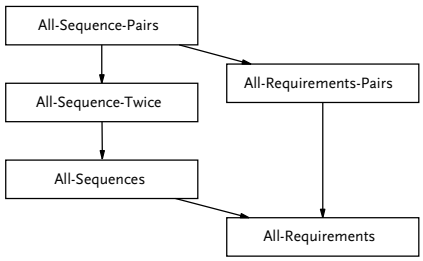

Figure 2: Requirements-based subsumption hierarchy.

5 Prototype Implementation

In this section, we describe the prototype implementation *ConSequence* [WS10b], which derived its name by the approach of concatenating transition sequences instead of single transitions. ConSequence is a Java-based Eclipse-plugin that can read EMF-based model descriptions defined by the Eclipse UML plugins and OCL conditions.

The state machine, the sequence diagrams that should be used for test generation, the coverage criteria to satisfy, the output file, and the test generator to use can be selected. Coverage criteria are transformed into test goals in ConSequence. For each yet unsatisfied test goal, a test case is generated. Unsatisfied test goals are reported. Finally, redundant test cases are removed from the test suite, i.e. test cases for which all test goals are covered by test cases that were created after them. For generating test cases, the tool relies on concatenating transition sequences and computing valid input-output data with the constraint solver Choco [The09]. The supported output formats of ConSequence are JUnit 3.8 and plain text. As a test oracle, we use state invariants that are derived from the state machine.

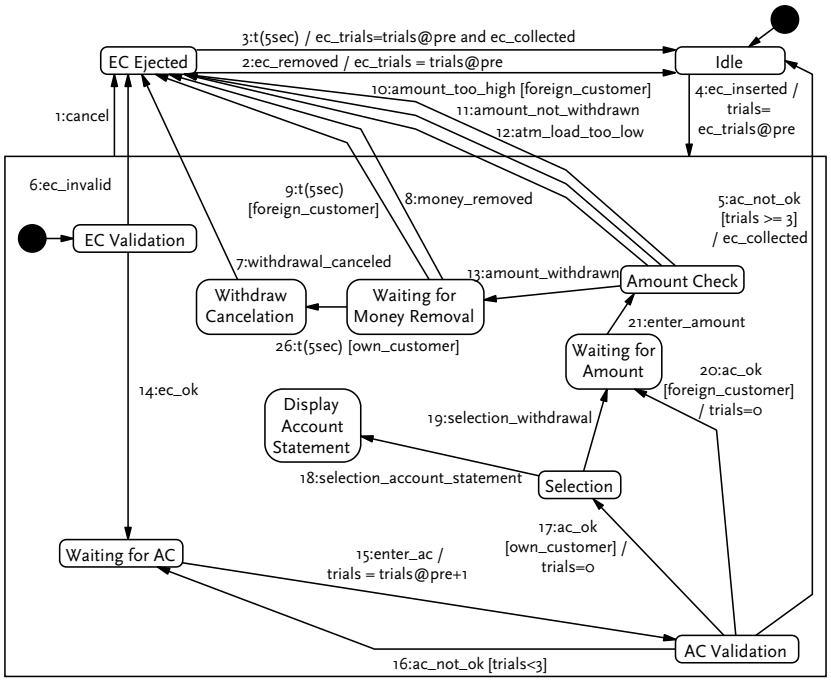

Figure 3: System behavior of the ATM.

6 Case Study

In this section, we describe the case study for ConSequence based on models describing the behavior of an automated teller machine (ATM). For the ATM, one state machine describes the behavior of the system and 25 sequence diagrams describe single interaction sequences. The state machine is shown in Fig. 3.

In the following, we describe our experiments to determine the impact of the defined coverage criteria. For our case study, we manually created a SUT implementation of the ATM. For each test suite generated for a certain coverage criterion, we run mutation analysis with

Jumble [UTC+07] on this SUT implementation. In mutation analysis, faulty versions of the SUT are created by injecting single faults. Each faulty version is called a *mutant*. If this mutant is detected by a test suite, the mutant is said to be *killed*. The ratio of killed to all mutants is the *mutation score*. The *coupling effect* [DLS78] states that most of the more complex faults are also covered by the simple ones [Off89, Off92].

First, we investigate the effects of the test suites generated with ConSequence. Afterwards, we also evaluate the effects of structure-based coverage criteria with corresponding test suites generated by ParTeG [Wei]. Finally, we will compare both kinds and evaluate the effects of combining both test suites.

Requirements-based Coverage. In this section, we analyze the mutation score for the test suites generated for the presented five requirements-based coverage criteria. Table 1 shows the test suite size measured as the number of test cases, the number of function calls and the mutation score of the mutation analysis. As expected, All-Requirements performs worst of all coverage criteria and All-Sequences performs better than All-Requirements without much effort overhead. All-Requirements-Pairs and All-Sequence-Pairs perform even better than All-Sequences. Their execution costs, however, increase dramatically: more than 300 test cases with more than 5000 function calls for All-Requirements-Pairs and All-Sequence-Pairs calls compared to just 23 test cases with only 159 function calls for All-Sequences! This effort increase was expected: The number of test goals for any of these two coverage criteria rises quadratically compared to the number of given requirements. These results were our motivation to consider coverage criteria with lower effort like All-Sequences-Twice that reach a high mutation score at considerable costs.

Coverage Criterion	Number of Test Cases	Number of Function Calls	Mutation Score
All-Requirements	19	165	69/87
All-Sequences	23	159	72/87
All-Requirements-Pairs	317	5411	73/87
All-Sequence-Pairs	369	6277	73/87
All-Sequence-Twice	23	440	73/87

Table 1: Results of mutation analysis for tests generated by ConSequence.

Structure-based Coverage. In this section, we apply ParTeG on the provided state machine to create test suites that satisfy structural coverage criteria [UL06]. We chose the coverage criteria All-States, All-Transitions, Decision Coverage, MC/DC [CM94], and Multiple Condition Coverage (MCC). Since the here presented test model contains no complex guard conditions, the results for Decision Coverage, MC/DC, and MCC are the same for this example (although they differ usually, see e.g. [WS08]). In the presented statistics, we just name MCC. Table 2 shows the results.

As expected, All-States performs worst of all coverage criteria. All-Transitions detects a fair amount of mutants with a low number of test cases. The three coverage criteria Decision Coverage, MC/DC, and MCC detected 72 mutants but used only 12 test cases. Compared to All-Sequences, this is the same mutation score for half of the test cases.

Coverage Criterion	Number of Test Cases	Number of Function Calls	Mutation Score
All-States	3	15	39/87
All-Transitions	11	69	72/87
MCC	12	76	72/87

Table 2: Results of mutation analysis for tests generated by ParTeG.

Comparison and Combination. In this section, we compare the results of the test suites generated by ConSequence and ParTeG. We are aware that this comparison strongly depends on the number and quality of the given requirements. Nevertheless, for this case study we can draw first conclusions: The obvious statement is that requirements-based coverage criteria can be able to reach a higher mutation score and structural-based coverage criteria can achieve still good mutation scores with less effort. The next result is that there are benefits of applying both kinds of coverage criteria. The results are shown in Table 3.

Coverage Criterion	Number of Test Cases	Number of Function Calls	Mutation Score
MCC + All-Sequences	35	235	74/87
MCC + All-Sequence-Pairs	381	6353	75/87
MCC + All-Sequence-Twice	35	516	75/87

Table 3: Results of mutation analysis for combined test suites.

The mutants detected by both approaches do not match completely. As a result, the combination of test suite generated by the two mentioned approaches result in even stronger test suites: For instance, combining MCC and All-Sequence-Twice detects 75 mutants with only 35 test cases. Given the mutation scores of the single test suites, we consider this improvement substantial. Even MCC and All-Sequences with the same number of test cases but half the number of function calls of MCC and All-Sequence-Twice performs better than the single test suites. As described in [FSW08], the combination of both test generation approaches at the test goal level can even lead to smaller test suites.

7 Conclusion, Discussion, and Future Work

In this paper, we presented a new approach to traceability in model-based testing. We presented the concatenation of requirements and defined corresponding coverage criteria. We described the developed tool ConSequence and presented the results of a case study. The application of our approach can result in stronger test suites than the ones generated to satisfy structural coverage criteria. As a major result, the combination of requirements-based and structural-based coverage criteria results in even stronger test suites.

There are several points to discuss. For instance, the quality of the presented test generation approach strongly depends on the quality of the given requirements. One way to deal with this problem is conducting more case studies. Furthermore, since our approach is focused on satisfying single test goals, the presented test generation approach can also be combined with other structural coverage criteria at the test goal level. This is conform to

our major result that the combination of structure-based and requirements-based coverage criteria is beneficial. As a result, the quality of our generated tests would be always at least as good as the quality defined by existing structure-based coverage criteria. Furthermore, we only used the message sequences defined in the sequence diagrams to map to transition sequences. Since there may be additional information given in the sequence diagrams, it might be worth considering to include them. Finally, we put the major focus on transition sequences as the context of requirements. There are other context information like, e.g. the attribute values, which should be included into test generation.

This leads to the intended future work. First, we plan to integrate further information about requirements context into test generation. Our second aim is to define and evaluate further coverage criteria. Third, we plan to use the described connection of sequence diagrams and state machines for integration testing: The sequence diagrams already describe integration of several components. They can also be used to combine state machines of the interacting objects. Our fourth aim is to extend the implementation ConSequence so that it can also deal with parts of the state machine that are not covered by requirements, e.g. by combining ConSequence and ParTeG on test goal level. Finally, we aim at extending our approach to also include requirements about forbidden behavior.

References

[AS05] Bernhard K. Aichernig and Percy Antonio Pari Salas. Test Case Generation by OCL Mutation and Constraint Solving. *Int. Conf. on Quality Software*, pages 64–71, 2005.

[Bin99] Robert V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[BJK05] Manfred Broy, Bengt Jonsson, and Joost P. Katoen. *Model-Based Testing of Reactive Systems: Advanced Lectures (LNCS)*. Springer, August 2005.

[BMM05] Antonia Bertolino, Eda Marchetti, and Henry Muccini. Introducing a Reasonably Complete and Coherent Approach for Model-based Testing. *Electronic Notes in Theoretical Computer Science*, 116:85–97, 2005.

[CM94] John Joseph Chilenski and Steven P. Miller. Applicability of Modified Condition/Decision Coverage to Software Testing. In *Software Engineering Journal, Issue*, volume 9, pages 193–200, September 1994.

[Con] Conformiq. Designer 4.2. http://www.conformiq.com/.

[DLS78] Richard A. DeMillo, Richard J. Lipton, and Fred G. Sayward. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer Journal*, 11(4):34–41, 1978.

[FSW08] Mario Friske, Holger Schlingloff, and Stephan Weißleder. Composition of Model-based Test Coverage Criteria. In *MBEES'08: Model-Based Development of Embedded Systems*, 2008.

[GH99] Angelo Gargantini and Constance Heitmeyer. Using Model Checking to Generate Tests from Requirements Specifications. *ACM SIGSOFT Software Engineering Notes*, 24(6):146–162, 1999.

[Mic10] Microsoft. SpecExplorer, 2010.

[Nag04] Roman Nagy. Bedeutung von Ausgangszuständen beim Testen von objektorientierter Software. In *CoMaTech'04*, Trnava, Slowakei, 2004.

[NFTJ03] Clémentine Nebut, Franck Fleurey, Yves Le Traon, and Jean-Marc Jézéquel. Requirements by Contracts allow Automated System Testing. In *ISSRE'03: Proceedings of the 14th. IEEE Int. Symposium on Software Reliability Engineering*, pages 17–21, 2003.

[OA99] Jeff Offutt and Aynur Abdurazik. Generating Tests from UML Specifications. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. 2nd Int. Conf., Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723, pages 416–429. Springer, 1999.

[Obj09] Object Management Group. Unified Modeling Language (UML), version 2.2. http://www.uml.org, 2009.

[Off89] A. Offutt. The Coupling Effect: Fact or Fiction? *SIGSOFT Softw. Eng. Notes*, 14(8):131–140, 1989.

[Off92] A. Jefferson Offutt. Investigations of the Software Testing Coupling Effect. *ACM Transactions on Software Engineering and Methodology*, 1(1):5–20, 1992.

[PP03] Stacy J. Prowell and Jesse H. Poore. Foundations of Sequence-Based Software Specification. *IEEE Trans. Softw. Eng.*, 29(5):417–429, 2003.

[Sok06] Dehla Sokenou. Generating Test Sequences from UML Sequence Diagrams and State Diagrams. In *INFORMATIK 2006: Informatik für Menschen - Band 2, GI-Edition: Lecture Notes in Informatics, P-94*, pages 236–240. Gesellschaft für Informatik, 2006.

[SW09] Dehla Sokenou and Stephan Weißleder. Combining Sequences and State Machines to Build Complex Test Cases. In *MoTiP'09: Workshop on Model-Based Testing in Practice*, June 2009.

[The09] The Choco Team. Choco Solver 2.1.0. http://choco.emn.fr/, 2009.

[UL06] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[UTC+07] Mark Utting, Len Trigg, John G. Cleary, Archmage Irvine, and Tin Pavlinic. Jumble. http://jumble.sourceforge.net/, 2007.

[Wei] Stephan Weißleder. ParTeG (Partition Test Generator). http://parteg.sourceforge.net.

[Wei10] Stephan Weißleder. Simulated Satisfaction of Coverage Criteria on UML State Machines. In *Int. Conf. on Software Testing, Verification, and Validation (ICST)*, April 2010.

[WS08] Stephan Weißleder and Holger Schlingloff. Quality of Automatically Generated Test Cases based on OCL Expressions. In *ICST'08: Model-Based Development of Embedded Systems*, pages 517–520. IEEE Computer Society, 2008.

[WS10a] Stephan Weißleder and Dehla Sokenou. Concatenating Requirements in Model-Based Testing - The ConSequence Approach. Technical report, Fraunhofer FIRST, Department Embedded Systems (EST), Berlin, 2010.

[WS10b] Stephan Weißleder and Dehla Sokenou. ConSequence. http://www.model-based-testing.de/tools/consequence, 2010.