ConSequence - Model-Based Testing With State Machines and Concatenated Sequence Diagrams

Stephan Weißleder
Fraunhofer-Institute FIRST
Kekuléstraße 7, 12489 Berlin, Germany
stephan.weissleder@first.fraunhofer.de

Dehla Sokenou
GEBIT Solutions
Koenigsallee 75b, 14193 Berlin, Germany
dehla.sokenou@gebit.de

1 Motivation

Model-based testing (MBT) offers several advantages over traditional testing such as requirements formalization, which can lead to early detection of inconsistencies, automation of test design, which usually implies a significant decrease of test design costs, and reduced test maintenance costs.

In this paper, we deal with two open issues of MBT: First, coverage criteria are often used to measure test quality and to steer automatic test generation. This means that covering all specified structural elements is the main focus of the generated test suite. As a consequence, typical user behavior is seldom reflected in the test suite, and the intention of single test cases is often hard to understand for users. To solve this issue, we propose to define typical user behavior as sequences that are mapped on the behavioral model. The second issue is the poor traceability from behavioral system models to functional requirements. An intuitive approach to allow for traceability is to annotate model elements with references to requirements [16, page 131]. This approach, however, suffers from two draw-backs: The model becomes overloaded and the integration of requirements links in the system 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) [9] as representations of typical single user behavior sequences, e.g., defined by functional requirements, and on UML state machines as the representation of complex system behavior.

Coverage criteria often drive automatic test generation in MBT. We are aware of only one requirements-based coverage criterion called *All-Requirements* [16] that requires to test each requirement. In addition, we present an approach to concatenate transition sequences of requirements that drives test generation by newly defined requirements-based coverage criteria.

In the following, we give an overview of related work in Section 2, present the ConSequence approach in Section 3 and case study results in Section 4, and conclude in Section 5.

2 Related Work

Our work is related to a number of approaches. For instance, in [16, page 131], the authors propose to reference requirements from single model elements - the requirements are covered if the referencing model elements are. Conformiq Designer [3] allows to attach requirements to single model elements. In contrast, we assume that requirements are not represented as single model elements, but as interaction sequences. SpecExplorer [6] 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 the automatic combination of such slices based on corresponding coverage criteria. State machines and sequence diagrams have been used for automatic test generation before: For instance, [12] generates test cases from state machines. In [8] test cases are derived 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. [1] combines both diagrams to derive "reasonably" complete test models to achieve early results for partially modeled systems. The papers [13] and [7] show 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. In contrast to our former work presented in [14], we introduce new coverage criteria, describe a prototype implementation, and show results from a case study.

3 The ConSequence Approach

We focus on concatenating sequences that potentially represent functional requirements. Thus, we use the terms *typical interaction sequence* and *requirements* interchangeably for the rest of the paper.

First, we describe a meta model to express relations of functional requirements (*Requirement*) and transition sequences of state machines (*TransitionSeq*) depicted in Fig. 1: Each behavior specified by a

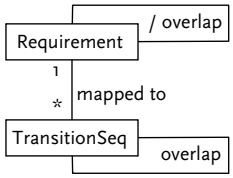

Figure 1: Meta model.

functional requirement can be mapped to several state machine transition sequences depending, e.g., on the start state. If the transition sequences of different requirements overlap each other, then the corresponding requirements are also overlapping.

3.1 Mapping

The task of the mapping is to identify the contexts of the requirements in the behavioral model, i.e., to map each requirement to transition sequences in the state machine. State machine transitions are triggered by incoming events. Thus, only the incoming message events for the object described by the state machine are extracted from the sequence diagram and mapped to state machine transition sequences that are triggered by the same sequence of events.

3.2 Overlap Identification

We define overlapping transition sequences using the following terms for all transition sequences *ts1* and *ts2*: *Start state* of any ts1 is the source state of the first transition in ts1. *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. 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.

3.3 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 sequence diagrams. For instance, parts of the state machine may be not covered by them and, consequently, pairs of non-overlapping transition sequences cannot be concatenated using the overlap graph. In such cases, transition sequence concatenation has to be done by applying other techniques.

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., checking that all constraints regarding guard conditions and effects on the contained transitions are valid.

3.4 Coverage Criteria

In this section, we describe coverage criteria for test generation with the ConSequence approach:

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

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

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

All-Sequences-Pairs is satisfied if for each ordered pair of requirements/sequence diagrams, *all* corresponding transition sequences are covered in the order of the requirements.

All-Sequences-Twice is satisfied if for each requirement/sequence diagram, *all* mapped transition sequences have been covered by test cases *at least twice*, which also comprises transition loops.

The subsumption hierarchy for the presented coverage criteria is shown in Fig. 2.

3.5 Prototype Implementation

ConSequence [19][1] is implemented as a Java-based Eclipse plugin that can read EMF-based model descriptions defined by the Eclipse UML plugins and OCL conditions. State machine, sequence diagrams

[1] not publicly available, yet - we'll present prototype at TAV

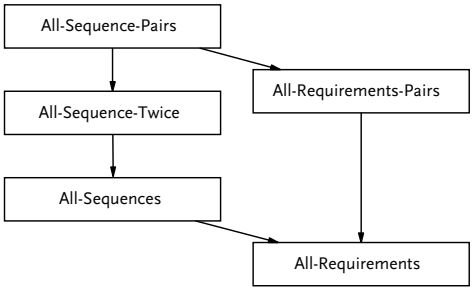

Figure 2: Requirements-based subsumption hierarchy.

that should be used for test generation, coverage criteria to satisfy, output file, and test generator can be selected. Coverage criteria are transformed into test goals. For each yet unsatisfied test goal, a test case is generated. Unsatisfiable test goals are reported. Redundant test cases are removed from the test suite, i.e., test cases for which all test goals are covered by other created test cases. For generating test cases, the tool relies on concatenating transition sequences and computing valid input-output data with the constraint solver Choco [15]. Supported output formats are JUnit 3.8 and plain text. As test oracle, state invariants derived from state machine are used.

4 Case Study

The case study is based on models describing the behavior of an automated teller machine (ATM). One state machine describes the behavior of the system and 25 sequence diagrams describe single interaction sequences. The described process consists of entering the credit card, validating the card, and checking the amount or withdrawing money. The interested reader can find the state machine in [14].

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 [17] 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* [4] states that most of the more complex faults are also covered by the simple ones [10, 11].

We evaluate the effects of 1) the test suites generated with ConSequence, 2) structure-based coverage criteria with corresponding test suites generated by ParTeG [18], and 3) combining both test suites.

4.1 Requirements-Based Coverage

In this section, we analyse the mutation score for the test suites generated for the presented five requirements-based coverage criteria (RBC). Table 1 shows the test suite size measured as the number of test cases, the number of system 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 system calls for All-Requirements-Pairs and All-Sequence-Pairs calls compared to just 23 test cases with only 159 system 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 System Calls	Mutation Score
All-Requirements	19	165	69/87
All-Sequences	23	159	72/87
All-Req.-Pairs	317	5411	73/87
All-Sequence-Pairs	369	6277	73/87
All-Sequence-Twice	23	440	73/87

Table 1: RBC results of mutation analysis.

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

Table 2: SBC results of mutation analysis.

4.2 Structure-Based Coverage

In this section, we describe the application of ParTeG on the provided state machine to create test suites that satisfy structural coverage criteria (SBC) [16]. We chose the coverage criteria All-States, All-Transitions, Decision Coverage, MC/DC [2], and Multiple Condition Coverage (MCC). Since the presented system model contains no complex guard conditions, the results for Decision Coverage, MC/DC, and MCC are the same. 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.

4.3 Comparison and Combination

In this section, we compare the results of the test suites generated by ConSequence and ParTeG. We are

Coverage Criterion	Number of Test Cases	Number of System 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: Combination results of mutation analysis.

aware that this comparison strongly depends on the number and quality of the given models. 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 high 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.

The mutants detected by both approaches do not match completely. As a result, the combination of test suites 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 system calls of MCC and All-Sequence-Twice performs better than the single test suites. As described in [5], the combination of both test generation approaches at the test goal level can even lead to smaller test suites.

5 Conclusion and Outlook

In this paper, we presented a new approach to traceability in model-based testing based on concatenation of requirements and newly defined corresponding coverage criteria. We presented the tool ConSequene and 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 future work, 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 second aim is to extend ConSequence so that it can also deal with parts of the state machine that are not covered by requirements, e.g., by a combination with ParTeG on test goal level.

References

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

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

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

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

[5] M. Friske, H. Schlingloff, and S. Weißleder. Composition of Model-based Test Coverage Criteria. In *MBEES'08: Model-Based Development of Embedded Systems*, 2008.

[6] Microsoft. SpecExplorer. http://research.microsoft.com/en-us/projects/specexplorer/, 2010.

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

[8] C. Nebut, F. Fleurey, Y. L. Traon, and J.-M. Jézéquel. Requirements by Contracts allow Automated System Testing. In *ISSRE'03: Proceedings of the 14th. IEEE International Symposium on Software Reliability Engineering*, pages 17–21, 2003.

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

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

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

[12] J. Offutt and A. Abdurazik. Generating Tests from UML Specifications. In R. France and B. Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723, pages 416–429. Springer, 1999.

[13] D. 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 (LNI), P-94*, pages 236–240. Gesellschaft für Informatik, 2006.

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

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

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

[17] M. Utting, L. Trigg, J. G. Cleary, A. Irvine, and T. Pavlinic. Jumble. http://jumble.sourceforge.net/, 2007.

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

[19] S. Weißleder and D. Sokenou. ConSequence. http://www.model-based-testing.de/tools/consequence, 2010.