Composition of Model-based Test Coverage Criteria

Mario Friske, Bernd-Holger Schlingloff, Stephan Weißleder
Fraunhofer FIRST, Kekuléstraße 7, D-12489 Berlin
{mario.friske|holger.schlingloff|stephan.weissleder}@first.fraunhofer.de

Abstract: In this paper, we discuss adjustable coverage criteria and their combinations in model-based testing. We formalize coverage criteria and specify test goals using OCL. Then, we propose a set of functions which describe the test generation process in a generic way. Each coverage criterion is mapped to a set of test goals. Based on this set of functions, we propose a generic framework enabling flexible integration of various test generators and unified treatment of test coverage criteria.

1 Motivation

In the field of software and system testing, the quality of test suites is one of the most important issues. Widely accepted means for assessing the quality of a test suite are coverage criteria. Criteria can be defined on the coverage of certain characteristics of specification, implementation, or even fault detection abilities of the test suite. The result is a large variety of defined criteria for structural, functional, and fault coverage. In the course of model-based engineering, structural coverage criteria have been adopted to models (e. g., UML state machines). Whereas existing norms and standards for safety-critical systems focus mainly on code coverage, this paper deals with model coverage criteria.

An interesting aspect is the relationship between different coverage criteria. Similar criteria are related by subsumption relations (e. g., *All-States* [Bin99] is subsumed by *All-Transitions* [Bin99]). However, relationships between criteria from different groups (e. g., *Multi-Dimensional* [KLPU04] defined on data partitions and *MCDC* [Lig02] defined on control-flow graphs) are not yet thoroughly investigated and need further analysis [WS08]. Nevertheless, testers desire a means for flexibly applying combinations of different coverage criteria, because individual criteria are well investigated and allow dedicated statements on covering certain risks and aspects.

In this paper, we present a generic framework that allows combining test suites at various abstraction levels. We provide functional descriptions for various stages in a coverage-oriented test generation process and discuss possibilities for the composition of coverage criteria and the optimization of test suites at different levels. We formalize coverage criteria and resulting test goals in OCL [Obj06] using an example from the domain of embedded systems. The framework uses functional specifications with OCL as a means for specifying test goals. It features flexible application and combination of coverage criteria and enables to plug-in various test generators. Furthermore, the framework supports precise statements on achieved coverage relying on OCL-based traceability information.

The remainder of this paper is organized as follows: In the next Section, we discuss coverage criteria and their relations and introduce our running example. In Section 3, we give a functional description of a multi-stage test generation process, and formalize test coverage criteria and test goals using OCL. In Section 4, we use functional signatures and specifications of test goals with OCL for sketching a framework to compose test coverage criteria. Finally, we draw some conclusions and give an outlook to future work.

2 Relation between Coverage Criteria

The aim of model-based testing is to validate a *system under test (SUT)* against its specification, the *model*. For that, a *test suite* is generated from the model and executed to examine the SUT. Coverage criteria qualify the relation between test cases and implementation or model. It is common sense to group coverage criteria into sets based on the same foundation, like structural (e. g. [Lig02]), functional (e. g. [Lig02]), and fault coverage (e. g. [FW93, PvBY96]).

Members of one group are related by the subsume relation and visualized as subsumption hierarchy [Nta88, Zhu96, Lig02]. In [Lig02, pages 135 and 145] two subsumption hierarchies for control-flow-oriented and data-flow-oriented criteria are presented. Since both hierarchies contain the criteria *Path Coverage* and *Branch Coverage*, they can be united in a single hierarchy using these criteria as merge points. The resulting hierarchy contains a variety of control-flow-oriented (including path-based and condition-based criteria) and data-flow-based coverage criteria, as depicted in Figure 1.

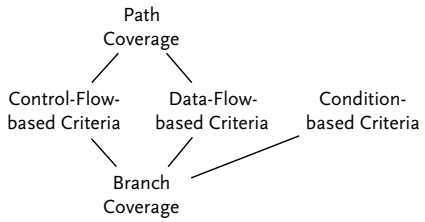

Figure 1: Combination of Different Types of Coverage Criteria

2.1 Example: a Fail-safe Ventricular Assist Device

As an example, we consider a ventricular assist device (VAD) supporting the human heart. It provides an external pulsatile drive for blood circulation, which can help patients with heart diseases to recover. The VAD is designed for stationary use with paediatric and stroke patients. Both pump and control unit are attached outside the body. There are several parameters, which can be set by the doctor via a serial connection such as systolic and diastolic pressure, desired blood flow, and pump frequency. The VAD has a redun-

dant architecture with three pneumatic motors, two of which must be frequently running. The control unit is replicated twice, as depicted in the object model diagram in Figure 2, one processor being operational and the other being in "hot" standby mode. In case of malfunction of one motor, the active controller starts the redundant one (in the appropriate mode). In case of failure of a controller, the redundant one takes over control.

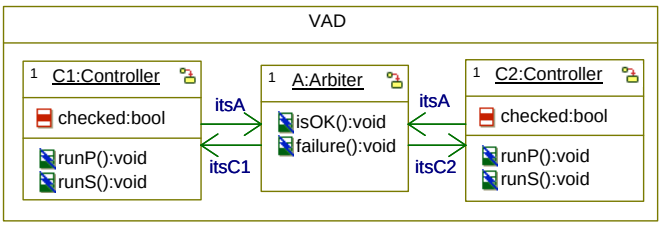

Figure 2: Object Model Diagram for VAD with two redundant controllers

In Figure 3 we give two UML2 state machines of (part of) the startup functionality of the software. Initially, the primary controller performs startup tests, reads the preset configuration, and starts the pumps. It then waits for some time to let the blood flow stabilize and compares the actual and desired pressure and flow values. A discrepancy of these values indicates a problem in one of the pumps, and the pumps are switched. Otherwise, the controller blocks itself, causing a reset, such that the roles of primary and secondary controller are switched, and self-testing is repeated. Subsequently, the VAD goes into main operational mode, where a continuous monitoring of pumps and blood circulation takes place.

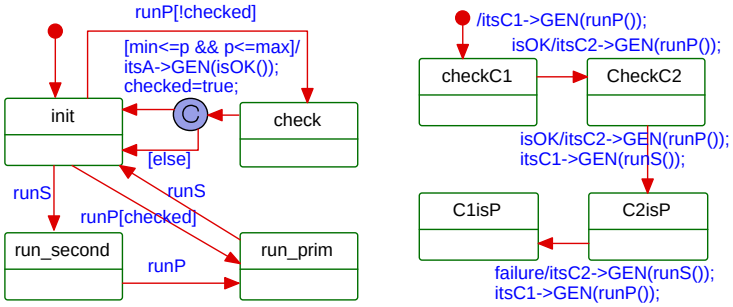

Figure 3: UML State Machines of VAD Controller (left) and VAD Arbiter (right)

2.2 Application of Combined Test Coverage Criteria

Thorough testing of models such as shown above requires application of (a combination of) several coverage criteria each focussing on a different aspect. E. g., for the state machines shown in Figure 3, a possible reasonable strategy comprises the following criteria:

1. (a) *All-Transitions* [Bin99] as lower viable criterion for testing state machine (this subsumes the criteria *All-States* [Bin99] and *All-Events* [Bin99]), and
   (b) *All-n-transitions* [Bin99] as extended state-based criterion targeting typical failures of state machines,
2. *MCDC* [Lig02] covering complex decisions in action code, and
3. *Boundary-Value Analysis* [Mye79] focussing on correct implementation of comparisons.

These criteria relate to the combined subsumption hierarchy in Figure 1 as follows: (1) results from transferring path-based criteria to state machines and (2) is a condition-based criterion. The criterion (3) is orthogonal to the depicted criteria. A data-flow-based criteria is not contained in this list, but still can be used to measure coverage achieved by applying criteria (1)–(3), see [Lig02].

Applying the above coverage criteria to the models of the VAD presented in Section 2.1 in order to generate test cases results in a number of test goals. In the following, we give an example of a test goal for each of the coverage criteria above:

All-2-Transitions: The sequence of the two transition characterized by the following trigger[guards]: (1) runP[!checked], (2) [min$\leq$p && p$\leq$max].

MCDC: Evaluation of the guard [min$\leq$p && p$\leq$max] where both conditions become true (i. e., min$\leq$p=true and p$\leq$max=true).

BVA: The upper bound of the equivalence partition "normal" (i. e., the highest value of $\mathtt{p}$ allowed for normal operation).

Note that it is possible to fulfill all test goals described above with a single test case.

3 Specification of Coverage Criteria and Test Goals using OCL

It is common practice to generate test cases in a multi-step transformation process: First, a generator applies coverage criteria to the specification (e. g., a state machine) and calculates all resulting *test goals* (e. g., paths through a state machine), see [IL04]. Then, the generator creates a *concrete test case* for each test goal (e. g. a sequence of events triggering a certain path through a state machine). We describe the multi-stage test generation process with a set of corresponding functional signatures:

$$gen : S \times CC \Rightarrow TS \times MD$$
$$goalgen : S \times CC \Rightarrow TG$$
$$testgen : TG \Rightarrow TS \times MD$$

where S is a specification, CC a coverage criterion, TG a set of test goals, TS the test suite, and MD meta data (e. g., describing achieved percentage of coverage). The overall

test generation function gen is the functional composition of the generation of test goals $goalgen$ and the generation of logical test cases from test goals $testgen$:

$$gen(S, CC) = testgen(goalgen(S, CC)).$$

In the following, we use OCL as a means to precisely define one exemplary coverage criterion and a corresponding test goal.

3.1 Specification of Coverage Criteria in OCL

In this section, we present an OCL expression that determines model elements for the coverage criterion *All-2-Transitions*. To satisfy this coverage criterion, a test suite has to contain all possible pairs of subsequent transitions t_1 and t_2, where $t_1.target = t_2.source$. All OCL expressions are based on the meta model of UML state machines [Obj07, page 525]. For sake of simplicity, the context of all OCL queries is a flat `StateMachine`. The following expression searches for pairs of subsequent transitions.

```
context StateMachine
def: all2Transitions : Set(Sequence(Transition)) =
  self.region.transition->iterate(
    t1 : Transition;
    resultSet : Set(Sequence(Transition)) =
    Set{Sequence{}} | resultSet->union(
      t1.target.outgoing->iterate(
        t2 : Transition;
        tmpSet : Set(Sequence(Transition)) =
        Set{Sequence{}} | tmpSet->including(
          Sequence{t}->including(t2)))))
```

3.2 Specification of Test Goals in OCL

In the previous section, we gave an OCL expression to define a coverage criterion on a UML model. The application of such a criterion to a specific model instance results in a set of test goals. In the following, we give two alternative OCL expressions for a particular test goal, which results from applying the coverage criterion *All-2-Transitions* on our example of a fail-safe Ventricular Assist Device presented in Section 2.1.

The first alternative depends on the availability of unique identifiers for the corresponding model elements:

```
context StateMachine
def: goalForAll2Transitions : Sequence(Transition) =
  Sequence{self.region.transition->select(name = 't1'),
    self.region.transition->select(name = 't2')}
```

It is also possible to calculate test goals using the OCL definition of the specified coverage criterion. For instance, the following expression selects the first element of the set of elements returned for the criterion *All-2-Transitions*:

```
context StateMachine
def: goalForAll2Transitions : Sequence(Transition) =
  all2Transitions->asSequence()->first()
```

4 A Generic Framework for Composing Coverage Criteria

In this section, we sketch a possible design for a generic framework that allows flexible integration of various test generators and unified treatment of test coverage criteria. The formalization of coverage criteria and individual test goals relies on the previously defined OCL expressions. In order to integrate various test generators, they have to fulfill the following requirements:

Access to test goals: We assume that the generators implement a goal-oriented approach and do not just provide a black-box functionality gen but two separated functions $goalgen$ and $testgen$ with independently accessible outcomes (see Section 3).

Precise signatures: Each test case generator has to provide precise meta-information about it's functionality including it's name, accepted types of specifications (model elements), parameters, achievable coverage, and related coverage criteria.

The implementation of a framework for generator integration requires clarifying the storage of test case and test goals. A unified format for test cases is not necessarily required as long as formal meta-information on the achieved coverage is accessible. We recommend specifying required and achieved test goals using OCL as presented in Section 3.2.

Furthermore, a precise specification of interfaces is required. A platform supporting plug-in-based software development such as *Eclipse* [Ecl] can be used to integrate various generators as plug-ins. Commercial off-the-shelf generators can be integrated using wrappers. Conversion functions for test goals from proprietary formats into OCL are required.

As we have discussed in [FS06], testers ideally want to select coverage using a graphical interface. Various test generators can be integrated in a plug-in-based framework. Each of these generators provides detailed information on achievable coverage, which can be presented to the tester in a structured or a graphical form (e. g., as a subsumption hierarchy). The tester can select a set of coverage criteria (e. g., using sliders on a graphical visualisation of a unified subsumption hierarchy as depicted in Figure 1). The selected coverage information is passed to the generators and used to control the test generation process.

Specifying test goals using OCL allows to attach detailed traceability information [Som07, Chapter 7] to each generated test case. This allows to return information to the tester about unsatisfied test goals as OCL expressions and to realize enhanced dashboard functionalities that overlap diverse generators. The tester can individually process unfulfilled test goals by manually adding test cases or treating test goals shown as being unreachable.

By applying more than one coverage criterion to the multi-stage test generation process, three integration options can be identified: (1) integration at coverage-criteria-layer (i. e., definition of newly combined coverage criteria), (2) integration at test-goal-layer (i. e., merging of test goals), and (3) integration at test-suite-layer (i. e., merging of test suites). The corresponding functional descriptions are the following:

$gen_1 : S \times CC_1 \rightarrow TS_1$
$gen_2 : S \times CC_2 \rightarrow TS_2$

1. $CC_n = CC_1 \oplus CC_2$
2. $TG_n = TG_1 \oplus TG_2$
3. $TS_n = TS_1 \oplus TS_2$

where $\oplus$ is a suitably defined merging operator.

Option 1 can be reasonable if the coverage criteria are not connected by the subsumption relation. Option 2 can be realized via function $goalunion : TG_1 \times TG_2 \rightarrow TG_n$. Option 3 can be realized via function $testunion : TS_1 \times TS_2 \rightarrow TS_n$. Note that in general, the test suite resulting from the union of two test goals is not the same as the union of the test suites for the components. Usually, it is desired that the resulting test suite is optimized with respect to implicit properties like the number or the average length of test cases. One solution is a dedicated optimization operation $opt : TS_n \rightarrow TS_{opt}$ that optimizes a test suite according to given properties while preserving coverage. The optimization requires a weighting function. As an example, an implementation of opt could remove duplicates and inclusions from a test suite.

5 Conclusions and Outlook

In this paper, we addressed specification and combination of coverage criteria. We sketched the relations between coverage criteria and provided a consistent set of functional signatures to define coverage criteria, test goals, test suites, and the corresponding generation functions. This is complemented by a formalization of coverage criteria and test goals. We substantiated our explanations by the example of a fail-safe ventricular assist device.

The following benefits result from our approach: Different test case generators can be integrated and combined using test goals represented as OCL expressions. Due to the use of OCL for formalization, coverage criteria, test goals, and test suites can be compared, merged, and optimized. Integration is possible at coverage-criteria-layer, test-goal-layer, and test-suite-layer. The use of OCL enables testers to define arbitrary coverage criteria. Furthermore, using OCL allows to evaluate achieved coverage and supports analysis and completion of missing test goals. The formalization of the test suite generation process allows to trace test cases to test goals, coverage criteria, and specification elements.

In the future, we plan to extend the presented approach. This includes the definition of coverage criteria on abstractions (including model-based tool support) and the prioritization of test goals. This paper did not deal with the fault detection capabilities of various (combined) coverage criteria; this topic needs to be investigated further. First steps towards this question have been done (e. g., in [PPW$^+$05, WS08]). Currently we have no tool support for the presented method. We want to implement the sketched framework and a corresponding OCL-based test generator.

References

[Bin99] R. V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Object Technology Series. Addison-Wesley, 1999.

[Ecl] Eclipse Foundation. Eclipse. http://www.eclipse.org/.

[FS06] M. Friske and B.-H. Schlingloff. Abdeckungskriterien in der modellbasierten Testfallgenerierung: Stand der Technik und Perspektiven. In Holger Giese, Bernhard Rumpe, and Bernhard Schätz, editors, *"Modellbasierte Entwicklung eingebetteter Systeme II" (MBEES)*, pages 27–33. Technische Universität Braunschweig, 2006.

[FW93] P. G. Frankl and E. J. Weyuker. A Formal Analysis of the Fault-Detecting Ability of Testing Methods. *IEEE Transactions on Software Engineering*, 19(3):202–213, 1993.

[IL04] I-Logix. *Rhapsody Automatic Test Generator, Release 2.3, User Guide*, 2004.

[KLPU04] N. Kosmatov, B. Legeard, F. Peureux, and M. Utting. Boundary Coverage Criteria for Test Generation from Formal Models. In *ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*, pages 139–150, Washington, DC, USA, 2004. IEEE Computer Society.

[Lig02] P. Liggesmeyer. *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. Spektrum Akadamischer Verlag, 2002.

[Mye79] G. J. Myers. *The Art of Software Testing*. John Wiley & Sons, Inc., 1979.

[Nta88] S. C. Ntafos. A Comparison of Some Structural Testing Strategies. *IEEE Transactions on Software Engineering*, 14(6):868–874, 1988.

[Obj06] Object Management Group. OCL 2.0 Specification, version 2.0 (formal/06-05-01), 2006.

[Obj07] Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2 (formal/07-11-02), 2007.

[PPW+05] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner. One evaluation of model-based testing and its automation. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 392–401, New York, NY, USA, 2005. ACM.

[PvBY96] A. Petrenko, G. v. Bochmann, and M. Yao. On fault coverage of tests for finite state specifications. *Computer Networks and ISDN Systems*, 29(1):81–106, 1996.

[Som07] I. Sommerville. *Software Engineering*. Addison-Wesley, 7th edition, 2007.

[WS08] S. Weißleder and B.-H. Schlingloff. Quality of Automatically Generated Test Cases based on OCL Expressions. http://www.cs.colostate.edu/icst2008/, April 2008.

[Zhu96] H. Zhu. A Formal Analysis of the Subsume Relation Between Software Test Adequacy Criteria. *IEEE Transactions on Software Engineering*, 22(4):248–255, 1996.