ParTeG - A Model-Based Testing Tool

Stephan Weißleder
Humboldt-Universität zu Berlin, Institut für Informatik
Rudower Chaussee 25, 12489 Berlin, Germany
weissled@informatik.hu-berlin.de

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

Abstract

Model-based testing is the comparison of a system under test to reference specifications in form of models. Tool support for model-based testing is of utter importance. In this paper, we present the model-based testing tool ParTeG, compare it with other model-based testing tools in the same area, and show future extensions.

1 Overview

ParTeG [6] is a model-based testing tool that was initially developed to implement new algorithms into a prototype as a proof of concept. By now, ParTeG is available as a free Eclipse plug-in, hosted by Sourceforge. ParTeG is based on the Eclipse UML 2.0 plugins. It automatically generates test cases from UML state machines and class diagrams that are annotated with OCL expressions. ParTeG supports test suite generation for JUnit 3.8, JUnit 4.3, Java Mutation Analysis, and CppUnit 1.12.

The tool has been presented before – the TAV workshop is used for further discussions about the tool's approach. A detailed introduction to ParTeG's features are presented in the following sections.

1.1 Interprete OCL Constraints

ParTeG supports a subset of the Object Constraint Language (OCL), excluding some of the expressions for collections. In contrast to other testing tools, like Smartesting Test Designer, ParTeG avoids assumptions if an OCL expression semantics is unclear. For example, the expression `x = y` as postcondition of an operation `f()` may be interpreted as:

1. After the execution of `f()`, the value of `y` has been assigned to `x`.
2. After the execution of `f()`, the value of `x` has been assigned to `y`.
3. After the execution of `f()`, both values `x` and `y` have changed but now they are equal.

ParTeG supports all three kinds, thus it can handle situations where the concrete value of a variable is unknown. To force the first interpretation, the expression should be transformed into the unambiguous form `x = y@pre`.

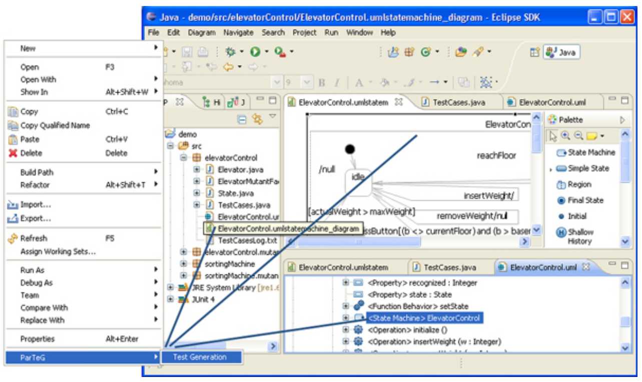
All variables are classified as *fixed* or *changeable* in the current expression: Input values, parameters, constants, and attributes with `@pre` are fixed (they cannot be changed) – variables marked without `@pre` are changeable in postconditions. This general interpretation of variables and the correspondingly general expression transformation rules allow even to deal with inequations in postconditions.

1.2 Combine Coverage Criteria

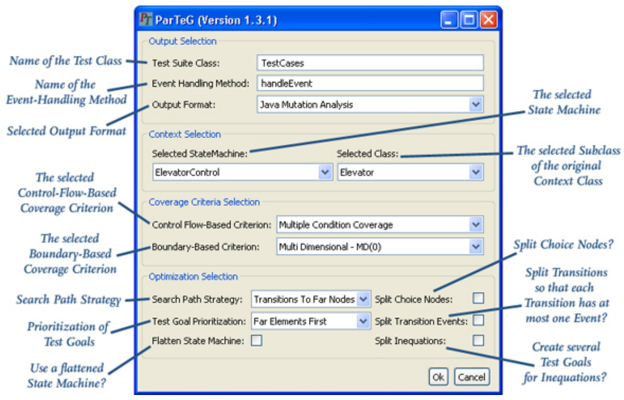
Coverage criteria are a widely accepted means of test suite quality measurement. They can be compared using subsumption relations. The most important feature of ParTeG is the ability to satisfy *combined coverage criteria*: For instance, control-flow-based coverage criteria (e.g. MC/DC) can be combined with boundary-based coverage criteria (e.g. Multi-Dimensional). From each category (control-flow-based or boundary-based), one coverage criterion can be chosen for the combination. At the moment, the following coverage criteria are supported: All-States, All-Transitions, Decision Coverage, masking MC/DC, and Multiple Condition Coverage as well as random input value selection and different variants of Multi-Dimensional: MD(0) with partition boundary values, and MD(1) with values at absolute range boundaries and with random values from input partitions.

Test cases are generated starting with a control-flow-based respectively transition-based coverage criterion. This coverage criterion is converted into a set of model-specific test goals. For instance, All-States is converted into test goals, each of which is referencing one state of the state machine. The test goals are used to generate transition sequences from the initial state to the referenced state. For each transition sequence, all influencing guard conditions and operations' pre-/postconditions are logged – they are transformed and used for concrete input parameter selection corresponding to the selected boundary-based coverage criterion. The transition sequences and the corresponding concrete input parameters are used to create test cases.

During automatic test generation, a test case for a test goal can cover other test goals. ParTeG supports monitoring of test goals, i.e. all already covered test goals are excluded from further test case generations.



(a) Modeling


Name of the Test Class
Name of the Event-Handling Method
Selected Output Format
The selected Control-Flow-Based Coverage Criterion
The selected Boundary-Based Coverage Criterion
Search Path Strategy
Prioritization of Test Goals
Use a flattened State Machine?
The selected State Machine
The selected Subclass of the original Context Class
Split Choice Nodes?
Split Transitions so that each Transition has at most one Event?
Create several Test Goals for Inequations?
Ok
Cancel

ParTeG (Version 1.3.1)

Output Selection

Test Suite Class: TestCases
Event Handling Method: handleEvent
Output Format: Java Mutation Analysis

Context Selection

Selected StateMachine: ElevatorControl
Selected Class: Elevator

Coverage Criteria Selection

Control Flow-Based Criterion: Multiple Condition Coverage
Boundary-Based Criterion: Multi Dimensional - MD(0)

Optimization Selection

Search Path Strategy: Transitions To Far Nodes
Split Choice Nodes: ☐
Test Goal Prioritization: Far Elements First
Split Transition Events: ☐
Flatten State Machine: ☐
Split Inequations: ☐

(b) Preferences

Figure 1: ParTeG User Interface

1.3 Mutation Analysis

Mutation analysis is a wide-spread approach to measure the fault detection capability of a test suite. This approach is based on fault injection in a correct implementation. All faulty implementations are called mutants. Mutants that are semantically different from the original implementation can be detected (*killed*) by a test suite. The assumption is that the more mutants are detected by a test suite, the better is this test suite's general fault detection capability [1].

ParTeG supports mutation analysis in two ways. First, it can use a mutation factory provided by the tester that delivers a new mutant for each test execution. These mutants can be fed into the factory manually, automatically, or by any other arbitrary means. Second, ParTeG can generate JUnit 3.8 code which can be used by the mutation tool Jumble [5].

2 Comparison with Commercial Tools

Tool	Test Cases	Detected Mutants
Rhapsody ATG	4	10/24
LTD	4	10/24
ParTeG	5	24/24

Table 1: Comparison with Commercial Tools

In previous case studies, we compared ParTeG to Rhapsody ATG [2] and Leirios (now: Smartesting) Test Designer (LTD) [4]. As quality criterion for test suites, we used mutation analysis. For one of our examples, a freight elevator, we generated 24 mutants. Table 1 shows the results for this relative simple example where only ParTeG is able to detect all mutants. We found that the support of combined coverage criteria has a wide influence on the quality of the generated test suite. For instance, LTD supports only the satisfaction of All-Transitions where ParTeG is able to satisfy a set of combined coverage criteria. The comparison with other tools is based on past versions of these tools. To validate the former results, a new comparison with current versions is necessary. For this purpose, we collected additional case studies for test generation with ParTeG, among others two industrial case studies.

3 Conclusion and Outlook

ParTeG offers a complete test cycle from the creation of test models (using the UML plugins or the editor TopCased [3]) to test suite generation for combined coverage criteria and subsequent test execution. At the current state, ParTeG supports automatic creation of input partitions even for multiple interacting input parameters, several different search strategies, test goal prioritization, test model adaptations, and derivation of state machines along inheritance relations between classes.

We plan to extend the implemented approach of ParTeG in the future. For example test generation can be supported for other models like a combination of the already integrated models (class diagrams, state machines) with interaction diagrams. A short comparison of ParTeG to commercial tools showed that ParTeG is able to detect more mutants than the others for our examples. We created a set of further test models from academia and industry and plan further comparisons to commercial tools.

References

[1] James H. Andrews, Lionel C. Briand, and Yvan Labiche. Is mutation an appropriate tool for testing experiments? In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 402–411, New York, NY, USA, 2005. ACM.

[2] IBM (Telelogic). Rhapsody Automated Test Generation. http://www.telelogic.com/products/rhapsody.

[3] Open Source. TopCased UML Editor 3.0. http://www.topcased.org/, 2009.

[4] SmarTesting. LTD/UML. http://www.smartesting.com.

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

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