User Guide

OSMOTester

MBT tool

Table of Contents

[Introduction 3](#_Toc294273485)

[Modeling notation 3](#_Toc294273486)

[Test generation 3](#_Toc294273487)

[Conclusions 3](#_Toc294273488)

[References 3](#_Toc294273489)

# Introduction

OSMOTester is a model-based testing (MBT) tool. It uses a state-machine notation to describe the system under test (SUT) from the testing perspective. A test model of the expected system behavior is provided by the user and used by the tool as a basis for automatically generating test cases for the SUT.

# Modeling notation

This section describes the OSMOTester modeling notation in terms of an example of a vending-machine borrowed from [[1](#Utting2007)]. This vending machine accepts three types of coins (10 cents, 20 cents, 50 cents) and when a total of 100 cents has been inserted the “vend” action can be activated to produce a bottle. Figure X illustrates this as a state-machine.

Figure 1. Vending machine as a state-machine.

The test models for OSMOTester are in practice executable programs written in the Java programming language. The specific model elements for the tool are identified based on a set of specific annotations. Listing 1 illustrates the OSMOTester notation using a vending-machine as an example.

public class VendingExample {

private final Scripter scripter;

private int coins = 0;

private int bottles = 10;

@TestSuiteField

private TestSuite testSuite = null;

public VendingExample() {

scripter = new Scripter(System.out);

}

@Guard

public boolean gotBottles() {

return bottles > 0;

}

@Before

public void start() {

coins = 0;

//uncomment this for failure to continue with 0 available transitions

bottles = 10;

int tests = testSuite.getHistory().size()+1;

}

@AfterSuite

public void done() {

}

@Guard("20cents")

public boolean allow20cents() {

return coins <= 80;

}

@Transition("20cents")

public void insert20cents() {

scripter.step("INSERT 20");

coins += 20;

}

@Guard("10cents")

public boolean allow10cents() {

return coins <= 90;

}

@Transition("10cents")

public void insert10cents() {

scripter.step("INSERT 10");

coins += 10;

}

@Guard("50cents")

public boolean allow50cents() {

return coins <= 50;

}

@Transition("50cents")

public void insert50cents() {

scripter.step("INSERT 50");

coins += 50;

}

@Guard("vend")

public boolean allowVend() {

return coins == 100;

}

@Transition("vend")

public void vend() {

scripter.step("VEND ("+bottles+")");

coins = 0;

bottles--;

}

@EndCondition

public boolean end() {

return bottles <= 0;

}

@Oracle

public void checkState() {

scripter.step("CHECK(bottles == "+bottles+")");

scripter.step("CHECK(coins == "+coins+")");

assertTrue(coins <= 100);

assertTrue(coins >= 0);

assertTrue(bottles >= 0);

}

public static void main(String[] args) {

OSMOTester tester = new OSMOTester(new VendingExample());

// tester.setDebug(true);

tester.generate();

}

}

Listing 1. Example vending machine in OSMO notation.

Here we see a number of the core OSMOTester model annotations being used.

# Test generation

OSMOTester generates test cases from the given test model.



# Conclusions

OSMOTester provides means to create test models and to generate test cases from these models.

# References

OSMOTester home page