Tutorial for Manual Drive

OSMO Tester

MBT tool

v2.1

Teemu Kanstrén

Table of Contents

[Introduction 3](#_Toc315031952)

[Creating specific tests manually 3](#_Toc315031953)

[Guiding the test generator manually 9](#_Toc315031954)

[Conclusions 11](#_Toc315031955)

[References 11](#_Toc315031956)

# Introduction

This tutorial describes the data modeling concepts of OSMO Tester using simple examples. It skips the background, the theoretical descriptions and just shows some examples. Check the OSMO manual for more lengthy descriptions. The reader should be familiar with the information presented in the OSMO Tester basic tutorial and data modeling tutorial.

The reader is expected to have basic knowledge of Java programming and ability to use their own favourite IDE such as Eclipse, IntelliJ, or Netbeans. The code shown in this tutorial is available in the OSMO Tester examples package.

# Creating specific tests manually

Previously in the data tutorial we created a model that prints “HELLO” and “WORLD” and uses ValueSet and ValueRange data model objects. In this tutorial we use the model program from the data tutorial as a basis and show how to manually create specific test cases from this model. As a reminder, Listing 1 shows the model program that was developed.

public class HelloModel {

private int helloCount = 0;

private int worldCount = 0;

private ValueSet<String> names = new ValueSet<String>("teemu", "bob");

private ValueSet<String> worlds =

new ValueSet<String>("mars", "venus");

private ValueSet<Integer> sizes = new ValueSet<Integer>(1,2,6);

private ValueRange<Double> ranges = new ValueRange<Double>(0.1d, 5.2d);

@BeforeSuite

public void init() {

names.setStrategy(DataGenerationStrategy.BALANCING);

}

@BeforeTest

public void startTest() {

helloCount = 0;

worldCount = 0;

System.out.println("TEST START");

}

@AfterTest

public void endTest() {

System.out.println("TEST END");

}

@Guard("hello")

public boolean thisNameReallyIsIrrelevant() {

return helloCount == worldCount;

}

@TestStep("hello")

public void sayHello() {

System.out.println("HELLO "+names.next()+" ("+sizes.next()+")");

helloCount++;

}

@Guard("world")

public boolean thisNameIsIrrelevant() {

return helloCount > worldCount;

}

@TestStep("world")

public void sayWorld() {

System.out.println("WORLD "+worlds.next()+" ("+ranges.next()+")");

worldCount++;

}

}

Listing 1. The model program from the basic tutorial.

Similarly, Listing 2 shows the configuration we set up to run the model program.

public class Main {

public static void main(String[] args) {

OSMOTester tester = new OSMOTester(new HelloModel());

tester.setSeed(345);

tester.addTestEndCondition(new Length(5));

tester.addSuiteEndCondition(new Length(3));

tester.generate();

}

}

Listing 2. Running the model program.

And as a final reminder, the output from running this model program is shown in Figure 1.

TEST START

HELLO bob (1)

WORLD mars (3.1562892313483015)

HELLO teemu (2)

WORLD mars (1.4289575493440612)

HELLO bob (2)

TEST END

TEST START

HELLO teemu (1)

WORLD venus (3.279034197651822)

HELLO teemu (2)

WORLD mars (2.814722267683214)

HELLO bob (1)

TEST END

TEST START

HELLO teemu (1)

WORLD mars (1.96781339845851)

HELLO bob (6)

WORLD venus (2.7852251942158026)

HELLO bob (1)

TEST END

generated 3 tests.

Figure 1. Example output.

So far the models have been used as a basis by the OSMO Tester for automatically generating test cases based on the defined test algorithm configurations. This is commonly what model-based testing (MBT) is defined as. However, considering overall management of automated test cases, it is useful from the test creation and maintenance viewpoint also to be able to automate more of the test generation in all aspect based on the test model. This way, there is less test script to write manually and the test model is used to cover more testing needs. More cost-effective that is. But let’s get to the point already.

How do we get the manual drive to use? Simply replace the test generation algorithm with the ManualDrive algorithms. This is shown in Listing 3.

public class ManualMain {

public static void main(String[] args) {

OSMOTester tester = new OSMOTester(new HelloModel());

tester.setSeed(345);

tester.addTestEndCondition(new Length(5));

tester.addSuiteEndCondition(new Length(3));

**tester.setAlgorithm(new ManualAlgorithm());**

tester.generate();

}

Listing 3. Running the model program.

Now, when you run this you will see the GUI pictured in Figure 2. Upper left corner shows the log of the test steps and data values you have chosen. In the bottom left corner you see the list of available test steps at this time. The only thing on this list is “hello” since “world” is only allowed after “hello”. Thus the GUI will always reflect what is legal for generation according to your model. Practically, it executes your model program one step at a time as you choose.

So click on “hello” in the lower left corner. What you will see is the GUI shown in Figure 3. This is asking you to specify a value for the “names” variable in the model program as the “hello” test step starts by asking a value to be generated for the “names” variable. With manual drive the user becomes the generator and all the OSMO modeling objects will ask the user for the input. Note that this only works with the ValueSet, ValueRange, ValueRangeSet, and ReadableWords objects included with OSMOTester. This should not be a major constraint since most data can be modeled in this way assuming some modeling skills.

Now the GUI for ValueSet shown in Figure 3 contains the options defined for the “names” variable. That would be “teemu” and “bob”. In this case, we choose “teemu” and press “OK”. The other choices are “Skip” and “Auto”. Using these causes the algorithm to generate automatically values in the same way that would be done if not manual drive is used. Skip causes one value to be generated for the variable. Auto causes this and all future instances of the variable to be automatically generated.

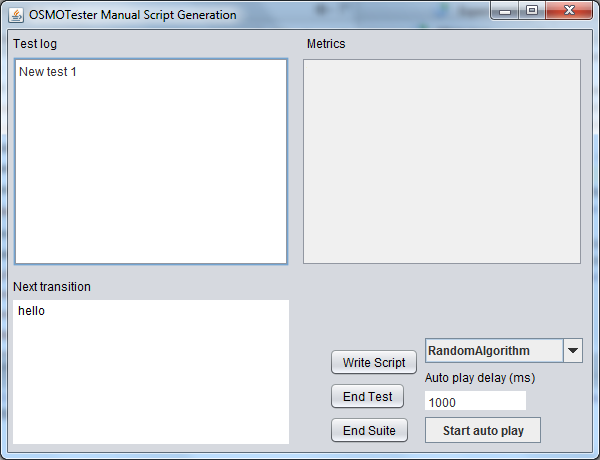


Figure 2. Manual drive GUI.

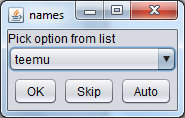


Figure 3. ValueSet GUI.

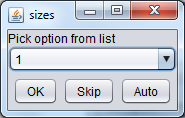


Figure 4. ValueSet GUI for “sizes”.

Now that we chose the value “teemu” and pressed OK, we should see the next value requested. Since the “hello” step also generates a value for this, it is also requested. As it is also a ValueSet, a similar GUI is shown to choose the value. This time we click “Skip”.

Now the manual drive GUI looks like Figure 5.

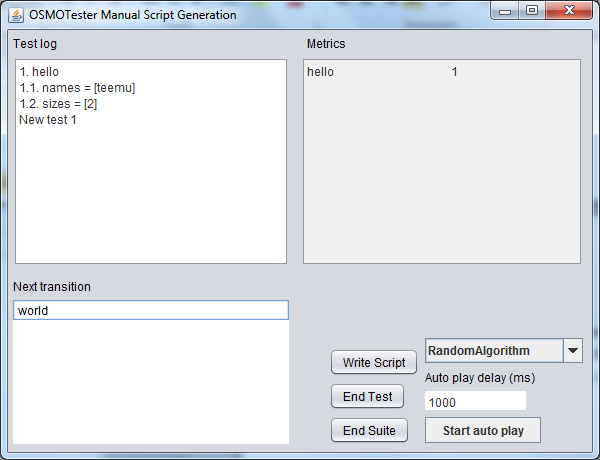


Figure 5. GUI after the first steps.

The log now shows that we have started the first test and chosen the first step, which is “hello”. For this step we have given the variable “names” value “teemu” and the second variable “sizes” got the value “2” from the automated algorithm choice. The metrics show that we have overall one step in our test cases and that is “hello”. The list of possible steps is now updated to show only “world” since that is the only enabled step once “hello” is taken once. Now we click on “world” to execute that step.

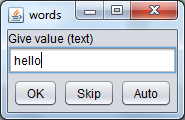


Figure 6. ReadableWords GUI.

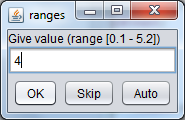


Figure 7. ValueRange GUI.

The “world” step requires values for variables “words” and “ranges”. These are shown in Figure 6 and Figure 7. After these choices, the overall GUI should now look like Figure 8.

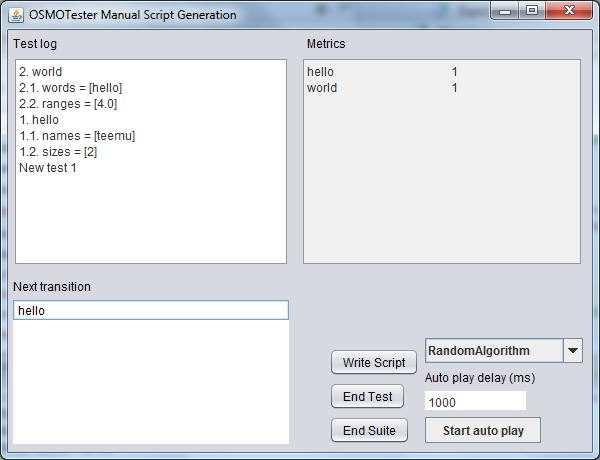


Figure 8. Yet another GUI screenshot.

From these, “end test” and “end suite” do what the title says, mainly noting that they cause associated model elements to be executed (after, aftersuite, before). It is also possible to start autoplay, which will choose steps according to the algorithm chosen in the algorithm box. The delay can be modified while running the autoplay to change how fast the steps are taken.

Typically the script is generated (offline) or test is executed (online) while the model program is executed. Additionally, a script in OSMO Tester format can also be written to disk using “write script”. The result is written to a text file called “osmo-tests.txt” in your working directory. In this case the output should look like that in Figure 9. Later, this file can be used to re-execute the manually created test script from the test model.

action, name, value

new test,,

step,hello,

variable,names,teemu

variable,sizes,2

step,world,

variable,words,hello

variable,ranges,4.0

Figure 9. Written script.

A similar script can also be written manually as a text file or the GUI generated file can be modified manually. To execute the text file, we need some special OSMO magic as shown in Listing 3.

public class ManualMain {

public static void main(String[] args) throws Exception {

AsciiParser parser = new AsciiParser();

List<TestScript> scripts = parser.loadAndParse("osmo-tests.txt");

ScripterMain main = new ScripterMain();

main.setSeed(345);

Collection<Object> models = new ArrayList<Object>();

models.add(new HelloModel());

main.run(models, scripts);

}

}

Listing 4. Running the manual script.

Here, we cannot define the end conditions or the algorithms since everything will simply be executed according to the script, which should contain all the required elements. Of course, if something is not there the results are unpredictable.. In this case, executing this with the previously generated script gives the output shown in Figure 10.

TEST START

HELLO teemu (2)

WORLD hello (4.0)

TEST END

generated 1 tests.

Figure 10. Example output.

# Guiding the test generator manually

In the above, we manually crafted some very specific scripts. Besides this, it is commonly interesting to be able to generate several tests that “slice” the general model according to some more specific rule without going to the level of detail required for specific single test cases.

Using the same model programs as before as input, we can initiate a GUI with the code shown in Listing 5.

public class DSMMain {

public static void main(String[] args) {

OSMOTester tester = new OSMOTester(new HelloModel());

tester.setSeed(345);

tester.addTestEndCondition(new Length(5));

tester.addSuiteEndCondition(new Length(3));

FSM fsm = tester.getFsm();

DSMGUI g = new DSMGUI(fsm);

g.setVisible(true);

}

}

Listing 5. Initializing the DSM GUI.

This GUI is shown in Figure 11.

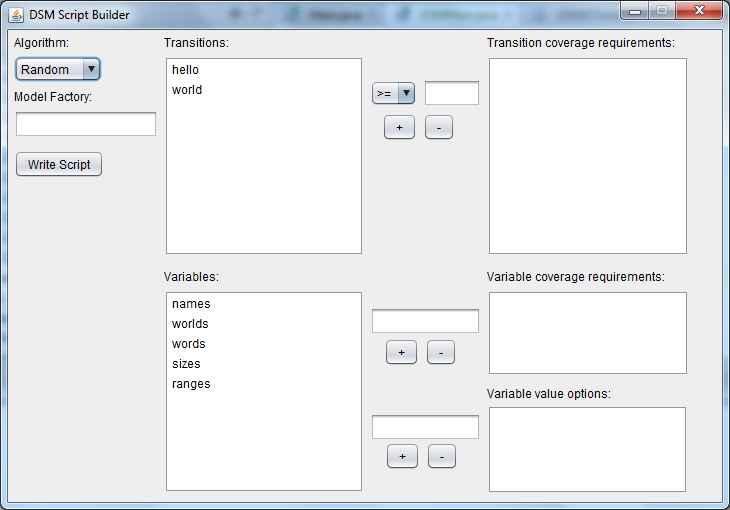


Figure 11. DSM GUI with the hello world example.

The rest of this tutorial needs to be written to describe the options..

To run the DSM script from a file we use Listing 6.

public class DSMLoaderMain {

public static void main(String[] args) throws Exception {

AsciiParser parser = new AsciiParser();

DSMConfiguration config = parser.loadAndParse("osmo-dsm.txt");

osmo.tester.scripting.dsm.DSMMain.execute(config);

}

}

Listing 6. Running the DSM script.

# Conclusions

This tutorial showed the how to use the manual modeling mechanisms to guide test generation with OSMO Tester. It still needs finishing for the DSM part and all that…

# References

OSMOTester home page, discussion forums & source code: <http://code.google.com/p/osmo/>