Model-Based Testing

An overview of existing tools

Pedro Tavares
up201708899@fe.up.pt

Raquel Dias
up201700419@fe.up.pt

Faculty of Engineering of the University of Porto

MESW1718-TVVS
15 December 2017

Model-Based Testing

Definition (Model-Based Testing)

Is a software **testing technique** that uses test generation algorithms and a behavioral model of the system under test to **generate test cases**.

It depends on three key technologies: the **notation** used for the data model, the test-generation **algorithm**, and the **tools** that generate supporting infrastructure for the tests.

Instead of writing hundreds of test cases, the test designer writes an **abstract model** of the system under test, and then, the model-based testing tool **generates a set of test cases** from that same model.

Model-Based Testing

Definition (Model-Based Testing)

Is a software **testing technique** that uses test generation algorithms and a behavioral model of the system under test to **generate test cases**.

It depends on three key technologies: the **notation** used for the data model, the test-generation **algorithm**, and the **tools** that generate supporting infrastructure for the tests.

Instead of writing hundreds of test cases, the test designer writes an **abstract model** of the system under test, and then, the model-based testing tool **generates a set of test cases** from that same model.

Model-Based Testing

Definition (Model-Based Testing)

Is a software **testing technique** that uses test generation algorithms and a behavioral model of the system under test to **generate test cases**.

It depends on three key technologies: the **notation** used for the data model, the test-generation **algorithm**, and the **tools** that generate supporting infrastructure for the tests.

Instead of writing hundreds of test cases, the test designer writes an **abstract model** of the system under test, and then, the model-based testing tool **generates a set of test cases** from that same model.

Tools Overview

We'll take a look on the following tools:

- ModelJUnit
- Spec Explorer
- Graphwalker
- MISTA

ModelJUnit

- Extends from JUnit for model-based testing.
- Analyze a finite state machine written in Java and convert it into a corresponding visual representation.
- After the finite state graph is created, test cases can be automatically generated, customized, and executed based on user preferences.
- It was created by Dr. Mark Utting, but it seems to be abandoned.

ModelJUnit

- Extends from JUnit for model-based testing.
- Analyze a finite state machine written in Java and convert it into a corresponding visual representation.
- After the finite state graph is created, test cases can be automatically generated, customized, and executed based on user preferences.
- It was created by Dr. Mark Utting, but it seems to be abandoned.

ModelJUnit

- Extends from JUnit for model-based testing.
- Analyze a finite state machine written in Java and convert it into a corresponding visual representation.
- After the finite state graph is created, test cases can be automatically generated, customized, and executed based on user preferences.
- It was created by Dr. Mark Utting, but it seems to be abandoned.

ModelJUnit

- Extends from JUnit for model-based testing.
- Analyze a finite state machine written in Java and convert it into a corresponding visual representation.
- After the finite state graph is created, test cases can be automatically generated, customized, and executed based on user preferences.
- It was created by Dr. Mark Utting, but it seems to be abandoned.

ModelJUnit

Finite State Machine

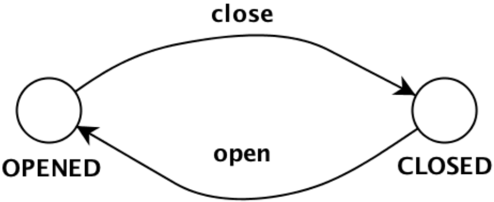

Figure: Simple Finite State Machine with two possible states: `Opened` and `Closed` and two events: `open`, that shifts from state `Closed` to `Open`; and `close`, that shifts from state `Open` to `Closed`.

ModelJUnit

Implementation

```
public boolean Move_To_Closed_Guard() {
  return this.state == State.OPEN;
}

@Action public void Move_To_Closed() {
  this.state = State.CLOSED;
}
```

Listing 1: Snippet that represents a transition/event of the model. It must use the @Action annotation. Guards can also be defined. In this case, it can only move to closed if the current state is open.

Spec Explorer

- Allow to model applications using C# and to generate state machine diagrams and unit tests from those models.
- It automatically generates test cases from very simple code that represents the model of the application.
- The generated test cases can be run against the implementation class, or they can be exported and run against the actual application.
- Documentation is available online on MSDN, as well as a couple of tutorials/examples.

Spec Explorer

- Allow to model applications using C# and to generate state machine diagrams and unit tests from those models.
- It automatically generates test cases from very simple code that represents the model of the application.
- The generated test cases can be run against the implementation class, or they can be exported and run against the actual application.
- Documentation is available online on MSDN, as well as a couple of tutorials/examples.

Spec Explorer

- Allow to model applications using C# and to generate state machine diagrams and unit tests from those models.
- It automatically generates test cases from very simple code that represents the model of the application.
- The generated test cases can be run against the implementation class, or they can be exported and run against the actual application.
- Documentation is available online on MSDN, as well as a couple of tutorials/examples.

Spec Explorer

- Allow to model applications using C# and to generate state machine diagrams and unit tests from those models.
- It automatically generates test cases from very simple code that represents the model of the application.
- The generated test cases can be run against the implementation class, or they can be exported and run against the actual application.
- Documentation is available online on MSDN, as well as a couple of tutorials/examples.

Spec Explorer

Implementation

```
[Rule]
public static void Move_To_Closed() {
  Condition.IsTrue(this.state == State.OPENED);
  this.state = State.CLOSED;
}
```

Listing 2: Snippet that represents a transition/event of the model. It must use the `Rule` attribute. Guards are implemented in the same method.

Graphwalker

- It generates test sequences from state machines modeled in GraphML with an external tool: yEd.
- It is designed to integrate with Java and Maven.
- Generate tests that can be run using a test tool like JUnit or Selenium.
- Each run generates a random run through the program that follow a path until reach all of the edges.
- Simple to acquire, hard to setup and customize.

Graphwalker

- It generates test sequences from state machines modeled in GraphML with an external tool: yEd.
- It is designed to integrate with Java and Maven.
- Generate tests that can be run using a test tool like JUnit or Selenium.
- Each run generates a random run through the program that follow a path until reach all of the edges.
- Simple to acquire, hard to setup and customize.

Graphwalker

- It generates test sequences from state machines modeled in GraphML with an external tool: yEd.
- It is designed to integrate with Java and Maven.
- Generate tests that can be run using a test tool like JUnit or Selenium.
- Each run generates a random run through the program that follow a path until reach all of the edges.
- Simple to acquire, hard to setup and customize.

Graphwalker

- It generates test sequences from state machines modeled in GraphML with an external tool: yEd.
- It is designed to integrate with Java and Maven.
- Generate tests that can be run using a test tool like JUnit or Selenium.
- Each run generates a random run through the program that follow a path until reach all of the edges.
- Simple to acquire, hard to setup and customize.

Graphwalker

- It generates test sequences from state machines modeled in GraphML with an external tool: yEd.
- It is designed to integrate with Java and Maven.
- Generate tests that can be run using a test tool like JUnit or Selenium.
- Each run generates a random run through the program that follow a path until reach all of the edges.
- Simple to acquire, hard to setup and customize.

MISTA

- It uses lightweight high-level Petri Nets as a visual modelling notation.
- It generates executable test code from a test model to several languages and testing frameworks. (Java and JUnit included)
- It make sure that all states and transitions in a model can be reached.
- Models can be built either using a GUI interface or a spreadsheet editor.
- Well suited for test-driven development.
- Simple to acquire. It is just a simple Java JAR application.

MISTA

- It uses lightweight high-level Petri Nets as a visual modelling notation.
- It generates executable test code from a test model to several languages and testing frameworks. (Java and JUnit included)
- It make sure that all states and transitions in a model can be reached.
- Models can be built either using a GUI interface or a spreadsheet editor.
- Well suited for test-driven development.
- Simple to acquire. It is just a simple Java JAR application.

MISTA

- It uses lightweight high-level Petri Nets as a visual modelling notation.
- It generates executable test code from a test model to several languages and testing frameworks. (Java and JUnit included)
- It make sure that all states and transitions in a model can be reached.
- Models can be built either using a GUI interface or a spreadsheet editor.
- Well suited for test-driven development.
- Simple to acquire. It is just a simple Java JAR application.

MISTA

- It uses lightweight high-level Petri Nets as a visual modelling notation.
- It generates executable test code from a test model to several languages and testing frameworks. (Java and JUnit included)
- It make sure that all states and transitions in a model can be reached.
- Models can be built either using a GUI interface or a spreadsheet editor.
- Well suited for test-driven development.
- Simple to acquire. It is just a simple Java JAR application.

MISTA

- It uses lightweight high-level Petri Nets as a visual modelling notation.
- It generates executable test code from a test model to several languages and testing frameworks. (Java and JUnit included)
- It make sure that all states and transitions in a model can be reached.
- Models can be built either using a GUI interface or a spreadsheet editor.
- Well suited for test-driven development.
- Simple to acquire. It is just a simple Java JAR application.

MISTA

- It uses lightweight high-level Petri Nets as a visual modelling notation.
- It generates executable test code from a test model to several languages and testing frameworks. (Java and JUnit included)
- It make sure that all states and transitions in a model can be reached.
- Models can be built either using a GUI interface or a spreadsheet editor.
- Well suited for test-driven development.
- Simple to acquire. It is just a simple Java JAR application.

MISTA

Petri Nets

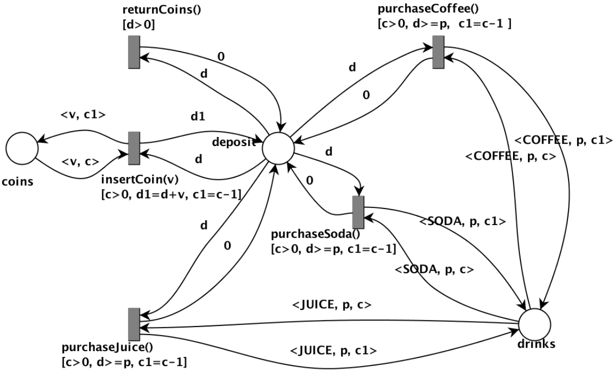

Figure: Petri Net that represents a vending machine system.

MISTA

Petri Nets

Definition (Petri Net)

A Petri net consists of **places**, **transitions**, and **arcs**. Arcs run from a place to a transition or vice versa, never between places or between transitions.

The places from which an **arc runs to a transition** are called the **input** places of the transition.

The places to which **arcs run from a transition** are called the **output** places of the transition.

MISTA

Petri Nets

Definition (Petri Net)

A Petri net consists of **places**, **transitions**, and **arcs**. Arcs run from a place to a transition or vice versa, never between places or between transitions.

The places from which an **arc runs to a transition** are called the **input** places of the transition.

The places to which **arcs run from a transition** are called the **output** places of the transition.

MISTA

Petri Nets

Definition (Petri Net)

A Petri net consists of **places**, **transitions**, and **arcs**. Arcs run from a place to a transition or vice versa, never between places or between transitions.

The places from which an **arc runs to a transition** are called the **input** places of the transition.

The places to which **arcs run from a transition** are called the **output** places of the transition.

MISTA

Petri Nets

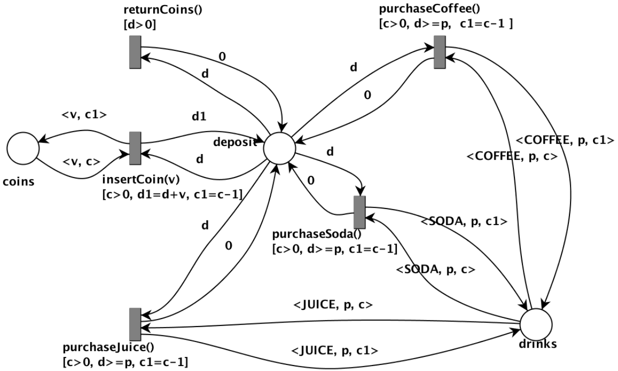

Figure: Petri Net that represents a vending machine system.

MISTA

Petri Nets

The Vending System Petri Net has the following places:

- `coins`
- `deposit`
- `drinks`

The Vending System Petri Net has the following transitions:

- `insertCoin(v)`
- `returnCoins()`
- `purchaseJuice()`
- `purchaseSoda()`
- `purchaseCoffee()`

MISTA

Some of the test coverage criteria available

- **Reachability tree coverage**: generates the reachability graph with respect to all given initial states and, for each leaf node, creates a test from the corresponding initial state node to the leaf.
- **Transition coverage**: tests are generated to cover each transition.
- **State coverage**: tests are generated to cover each state that is reachable from any given initial state.
- **Depth coverage**: only the tests whose lengths are no greater than the given depth are generated.

MISTA

Example of a generated test

```java
public void test116() throws Exception {
  System.out.println("Test␣case␣116");
  vm.insertCoin(Coin.DOLLAR);
  vm.insertCoin(Coin.NICKEL);
  vm.purchase(COFFEE);
  vm.insertCoin(Coin.QUARTER);
  vm.returnCoins();
  assertTrue("1_1_1", vm.getDeposit() == 0);
}
```

Listing 3: Example of a generated test based on Reachability tree coverage. Tries to check if the vending machine deposit is empty after returning the coins.

Model-Based Testing

An overview of existing tools

Pedro Tavares
up201708899@fe.up.pt

Raquel Dias
up201700419@fe.up.pt

Faculty of Engineering of the University of Porto

MESW1718-TVVS
15 December 2017