The next step in test automation: computer-generated tests

Antti Kervinen

Intel

antti.kervinen@intel.com

Guest lecture for Software Testing and Quality Assurance
Tampere University of Technology, Finland
Oct 7th 2013
Welcome to the next level in test automation

Typical testing limitations

- Scripts test the same paths over and over again.
- Nobody tests most of the paths a user can take.
- Huge test suites require lots of maintenance.

How model-based testing improves the situation:

- Computer generates tests. In practice, it can cover any number of different paths.
- Tests are generated from relatively small number of models. Remarkably smaller maintenance effort.
Welcome to the next level in test automation

This presentation will give you:

- basic knowledge on model-based testing.
- basic knowledge to try out the fMBT tool.
Contents

Part I  Introduction to model-based testing (MBT)
  - How MBT differs from test cases and test scripts?
  - What are benefits of MBT?
  - When MBT is a good choice?

Part II  Hands-on: test generation from scratch
  - Test generation and execution explained
  - MPlayer example
Traditionally test steps are executed in **fixed order**:  

1. Instantiate Camera preview.  
2. Capture image.  
4. Stop video capturing.  

This does not depend on  
- the **target** of the test: unit, integration, system test... or  
- the **purpose** of the test: smoke, regression, reliability, performance test...  

Now, let’s see how model-based testing differs from this.
There are no predefined test cases in model-based testing. They are generated automatically. This is possible when you have two things:

- **Library** of test steps...
  - preview
  - captureImage
  - startVideoCapt
  - stopVideoCapt

- **Conditions** when they’re enabled
  - always
  - if not capturing
  - always

From these a model-based testing tool can automatically generate many different tests. For instance, ...
Difference between model-based and test case based testing

A reliability test:
- stopVideo capt
- captureImage
- stopVideo capt
- startVideo capt
- stopVideo capt
- preview
- stopVideo capt
- startVideo capt
- preview
- ...

A smoke test:
- startVideo capt
- stopVideo capt
- captureImage
- preview

Test generation parameters define what kind of test is wanted. This will be demonstrated in the next part of the presentation.
Difference between model-based and test case based testing

Now you have learned:

Part I  Introduction to model-based testing (MBT)
- How MBT differs from test cases and test scripts?
- What are benefits of MBT?
- When MBT is a good choice?

Part II  Hands-on: test generation from scratch
- Test generation and execution explained
- MPlayer example

Let’s find out some benefits next.
Benefits of model-based testing

Benefits of model-based testing:

1. Increased test coverage.
2. Easier test case maintenance.

First, an example of increased coverage...
Benefits of model-based testing

A reliability test:
- stopVideo capt
- captureImage
- stopVideo capt
- startVideo capt
- stopVideo capt
- preview
- stopVideo capt
- startVideo capt
- preview
- ...

Consider the reliability test on the left. If you needed to test these functions, would you have created test cases for:
- stopping video capturing without starting it first?
- testing video capturing before and after using preview?
- creating a preview during video capturing?

Most often people do not think of all these cases. Now they were covered automatically.
Benefits of model-based testing

Benefits of model-based testing:

1. Increased test coverage.
2. Easier test case maintenance.

Next, two questions for audience on maintenance...
Benefits of model-based testing

Question 1 (new feature): You are asked to test that preview, captureImage and video capturing work in portrait, landscape and auto orientation modes. How would you handle this, when you have . . .

. . . a dozen test cases for testing them and some of their interactions?

. . . library of test steps and conditions?

- preview
- captureImage
- startVideoCapt
- stopVideoCapt
- nextOrientationMode

- always
- if not capturing
- always
- always

What else these steps can test?
Benefits of model-based testing

Question 2 (disabling tests). Due to changes on a platform, the preview feature will be broken on next week’s builds. You are asked to keep testing video and image capturing, and orientations. How would you handle this, when you have...

...a dozen test cases for testing preview, captureImage and video capturing plus the orientation testing modifications?

...library of test steps and conditions?

- preview
- captureImage
- startVideoCapt
- stopVideoCapt
- nextOrientationMode
- always, never
- if not capturing
- always
Benefits of model-based testing:

1. Increased test coverage.
2. Easier test case maintenance.

We have demonstrated the second benefit. As all test cases are generated, making changes – even big ones – is easy compared to fixing a large number of predefined test cases.
Benefits of model-based testing

Now you have learned:

**Part I**  
*Introduction to model-based testing (MBT)*
- *How MBT differs from test cases and test scripts?*
- *What are benefits of MBT?*
- *When MBT is a good choice?*

**Part II**  
*Hands–on: test generation from scratch*
- *Test generation and execution explained*
- *MPlayer example*

Finally, let’s see where model-based testing is a good choice and where not.
Where model-based testing is a good choice

Model-based testing generates tests for you. **Anything that can be tested with automated test cases, can be tested using model-based testing, too.**

More important than where model-based testing **can** be applied is: Where model-based testing **gives greatest benefits?**

Let’s see how to recognize these cases...
Where model-based testing is a good choice

Model-based testing is beneficial, if

- things need to be tested in many situations / configurations

Does a call come through? How about when playing music? Watching videos? Capturing a video? Transferring a file using 3G, Wifi, Bluetooth and USB? And the same for a Skype call?

Just write a test step for each activity, and you will get the tests.
Where model-based testing is a good choice

Model-based testing is beneficial, if

- things need to be tested in many situations / configurations
- you need long tests with lots of variation

Take the previous test steps and make the preconditions liberal: music is played during file transfer, video capturing is started during music is played, etc. And calls are received in different combinations of activity.

Now you can generate tests where the device is used in very interesting ways. If wanted, tools can generate and run a single test for hours or days and keep varying different activity combinations all the time.
Where model-based testing is a good choice

Model-based testing is beneficial, if

- things need to be tested in many situations / configurations
- you need long tests with lots of variation
- many combinations or interleavings to be tested

Test a service with \( n \) concurrent users. Is there an interleaving of user actions that renders the system unresponsive for any of the users?
Where model-based testing is a good choice

Model-based testing is beneficial, if

- things need to be tested in many situations / configurations
- you need long tests with lots of variation
- many combinations or interleavings to be tested
- you do monkey testing, fuzzing, . . .

Some model-based testing tools (like fMBT) allow inspecting the state of the system under test in preconditions of test steps. This enables, for instance, generating tests that look which buttons are on the display, choose a test step that clicks one of them, and then look again what could be the next test step. A sophisticated test generator allows testing different combinations more efficiently than a pure random “monkey”.
Where model-based testing is a good choice

The other way around, model-based testing is not beneficial, if...

- the system under test is stateless, and
- there are no different combinations (such as parameter values or environment configurations) that should be tested.

For a stateless systems it’s enough to test every input only once. The order in which test steps are executed and inputs sent does not matter. There is no reason to use model-based testing tools for generating many test step sequences in this case.
Now you have learned:

**Part I** *Introduction to model-based testing (MBT)*
- *How MBT differs from test cases and test scripts?*
- *What are benefits of MBT?*
- *When MBT is a good choice?*

**Part II** *Hands-on: test generation from scratch*
- *Test generation and execution explained*
- *MPlayer example*

Next we’ll take a look at the fMBT tool.
Hands-on: test generation from scratch

In this part of the presentation:
- Test generation and execution explained
- MPlayer example
Test generation and execution explained

fMBT test generation and execution:

1. Load test configuration, most importantly model, adapter and end conditions.
Test generation and execution explained

fMBT test generation and execution:

1. Load test configuration, most importantly model, adapter and end conditions.

2. Loop...
   - choose one of possible test steps

- choose one of possible test steps
Test generation and execution explained

fMBT test generation and execution:

1. Load test configuration, most importantly model, adapter and end conditions.

2. Loop...
   - choose one of possible test steps
   - try executing it in an adapter

Antti Kervinen (Intel)
Test generation and execution explained

fMBT test generation and execution:

1. Load test configuration, most importantly model, adapter and end conditions.
2. Loop...
   - choose one of possible test steps
   - try executing it in an adapter
   - validate executed test step reported by the adapter
Test generation and execution explained

fMBT test generation and execution:

1. Load test configuration, most importantly model, adapter and end conditions.

2. Loop...
   - choose one of possible test steps
   - try executing it in an adapter
   - validate executed test step reported by the adapter
   - execute the test step in test model

Antti Kervinen (Intel) The next step in test automation / Part II: Hands-on with fMBT
fMBT test generation and execution:

1. Load test configuration, most importantly model, adapter and end conditions.

2. Loop...
   - choose one of possible test steps
   - try executing it in an adapter
   - validate executed test step reported by the adapter
   - execute the test step in test model

3. ... until any of end conditions is met.
Test generation and execution explained

fMBT editor supports two modelling languages: GT (graph transformations) and AAL (a pre/postcondition language with adapter blocks). In this example we’ll use AAL/Python, that is, AAL where the code is written in Python.

AAL syntax for test step definitions:

```plaintext
action "name of this test step" {
    guard() { code: return true iff test step can be executed }
    body() { code: update variables after successful execution }
    adapter() { code: interact with the system under test }
}
```
We will generate tests for MPlayer’s (http://www.mplayerhq.hu) slave mode. In the slave mode MPlayer can be controlled in many ways via standard input. The mode enables using MPlayer as a backend behind a GUI. We will test that pause, continue, step to next / previous file, and adding files to the play list works.
Testing MPlayer’s slave mode

Install git for next steps:

```bash
sudo apt-get install git
yum install git
```

Install fMBT from sources:

Install dependencies listed in README: https://github.com/01org/fMBT, then
```bash
git clone https://github.com/01org/fMBT.git
cd fMBT; ./autogen.sh; ./configure; make; sudo make install
```

(Optional) Install MPlayer to run tests:

```bash
apt-get install mplayer (Ubuntu, universe or multiverse)
yum install mplayer (Fedora, rpmfusion)
```

Download the MPlayer test:

```bash
git clone https://github.com/askervin/fmbt-mplayertest.git
cd fmbt-mplayertest
```

Launch fMBT editor with mplayertest.aal and a test configuration:

```bash
fmbt-editor mplayertest.aal smoke.conf
```