Model-Based Testing in Practice with Smartesting CertifyIt

The goal of this session is:
- to practice the CertifyIt model-based testing tool on an existing example
  - generate tests from the given ATM model
  - concretize tests using a given test harness
  - execute tests on the web application of the system
  - establish the conformance depending on the observations
  - use additional means to drive the test generation (test purposes, TOCL properties)
- add a functionality to the model, and perform the MBT process all over again

We provide for this session 5 laptops that are shipped with:
- a Linux distribution
- a Java Runtime Environment, and a Java Development Kit
- Rational Software Architect (UML modeller)
- the CertifyIt test generator
- a web application that simulates the ATM
- a bench of mutants that present implementation errors

You will work in pairs.

1. Discovery of the System Under Test

The system under test (SUT) is a web application that simulates an ATM.
It is a single web page named "dab.html" that is located on the desktop of the laptops.

You can play with the application to discover how it works (pin code for the valid card is 1234).

2. Discovery of the model

Open Rational Software Architect, select the ATM project and browse the model. You should be able to answer the following questions:
- what is the SUT class?
- what are the model variables?
- how many instances of each class is there?
- what are the requirements that are considered here? (complete the requirements table)
- which part of the specification is not modelled?

3. Generation of functional tests

Open the CertifyIt test generator. Run a test campaign and browse the resulting tests.

Are there any missing requirements that are not covered by these tests?

Publish the test cases with the HTML test publisher. Open the file. What data are exported? How would they be useful?

4. Test concretization and execution

Publish the test cases as a Java/JUnit file, using the appropriate publisher.

Open Eclipse and create a new Java project. Add the files generated by CertifyIt and the concretization layer files to the project, along with the Selenium test driver library.

- how are the abstract and concrete levels mapped together?
- what are the observations that are performed? how are they made?

Compile the project and execute the tests. Compare the implemented conformance relationship to ioco.

5. Test purposes

a) Write a test purpose that ensures the persistence of the retry counter on the card. First, reach a state in which there is only one try left. Then eject the card, take it back, and re-insert it to perform a last failed attempt.

b) Write another test purpose that blocks the card, and then tries to re-use it to perform a withdrawal.

Export the tests, and execute them. Did you reveal a non-conformance? Where does the error come from?

6. TOCL properties

Write the two TOCL properties that correspond to the additional requirements given in annex.

Evaluate the existing test suite w.r.t. these properties.

7. Update of the model

In this part, you will complete the model, and follow the previous steps to re-generate new tests and execute them.

- define a status for the cards (use an enumeration: valid, unreadable, expired, unknown, suspended)
- update the OCL code to verify the card status and perform the appropriate actions
- generate the functional test cases
- update the adaptation layer to deal with these new abstract values and their expected behaviour of their usage.

8. Evaluate the tests' efficiency

The "mutants" directory contains a set of 5 faulty implementations (1 fault per mutant).

Use your new model-based testing skills to test them and find out what the errors are.