Towards Model-Based Testing of Domain-Specific Modelling Languages

J. Merilinna, Olli-Pekka Puolitaival, J. Pärssinen
Index

• Testing Domain-Specific Modelling Languages (DSML)
• Model-Based Testing (MBT)
• Application Testing
• Modelling Language Testing
• Test Suite Generation in Practice
• Case Lego
• Future research
Testing Domain-Specific Modelling Languages (DSML)

- Currently: Manual testing during iterative and incremental DSML development
- Problems:
  - Test applications in synch in metamodel?
  - Test coverage
- How DSML testing can be systematized and automated?
Model-Based Testing (MBT)

- MBT is a black box testing technique
- Phases:
  1. Modelling
  2. Test generation
  3. Test execution
- MBT is decreasing test suite maintenance efforts and improves coverage.
Application Testing

- Tests and the implementation are generated from the same model
- Code generator and metamodel (from the utilized parts) are under testing
- Comprehensive test suite is automatically generated and always in synch with implementation
Modelling Language Testing

- One application can be considered as one test case. Multiple test cases are required!
- Metamodel describes rules of the language
- We argue that applications can be generated from metamodel definitions
Test Suite Generation in Practice

- DSM tool requirements
  - Allows exporting the metamodel and importing the application models
  - MetaEdit+ fulfils the requirements
- MBT tool requirements
  - Supports model importing and exporting the test cases
  - Conformiq Qtronic fulfils the requirements
Case Lego: Generating Application

Application model

Generated code

Compiling

Application Execution
Application Model
Generated Code

```c
#include "robo_framework.nxc"

#define ID3_5257 1
#define ID3_7371 2
#define ID3_5177 3
#define ID3_4386 4
#define ID3_5311 5
#define ID3_4382 6
#define ID3_5751 7

task MyLogic()
{
    int state = ID3_4382 //Starting state
    int run = 1;

    while(run)
    {
        switch(state)
        {
            case ID3_5257: //Beep state
                beep(300, 440, 1);
                until(getNoise() < 40);
                until(getTouch() != 1);
                while(1){
                    if(getNoise() > 40){
```

Compiling

```c
#define ID_5311 5
#define ID_4382 6
#define ID_6731 7

void MyLogic()
{
    int state = ID_4382; // Starting state
    int run = 1;

    while(run)
    {
        switch(state)
        {
            case ID_5257: // Beep state
                beep(300, 440, 7);
                until(getNoise() < 40); // until (getNoise() < 40);
                until(getTouch() != 1);
                while(1){
                    if(getNoise() > 40){
                        state = ID_7371;
                        break;
                    }
                    if(getTouch() == 5){
                        state = ID_4385;
                        break;
                    }
                }
                break;
            case ID_7371: // End state
                run = 0;
        }
    }
}
```
Application Execution
Case Lego: Generating Tests

Application model → Model into QML format → Test generation

Test cases

Test Case Number 1 [MSC]
Test Case Number 2 [MSC]
Test Case Number 3 [MSC]
Test Case Number 4 [MSC]
Test Case Number 5 [MSC]
Test Case Number 6 [MSC]
Test Case Number 7 [MSC]
Test Case Number 8 [MSC]

Qtmeric Configuration [show]
Coverage Information [show]
Requirement Traceability Matrix [hide]
Application Model
Transform the Model into QML Format
Test Generation
Test Execution
Case Lego: One Generated Test Case
Future Research

- More applications model tests
- Metamodel testing and make a demo of that
- Identifying the most potential domains, and possible restrictions.
- Identifying effects in quality, processes…
Questions?