An Approach to use Executable Models for Testing

Michael Soden
Hajo Eichler

Department of Computer Science
Humboldt University Berlin

{soden|eichler@ikv.de}
MDD in Practice

- requirements capturing
- system planning
- system design
- development environment
- test, integration and operation
- medini base technology – modelling infrastructures
Contents of this talk

- Overview and Motivation

- Approach outline
  - Architecture and Framework
  - Execution and test of models

- Current status

- Outlook
MDD in practice: Observations

- **Process related...**
  - Modelling basically for abstraction, overview and documentation
  - No test *integration*, testing means doubled work by separate test-models
    - No practicable solution for *TestData*

- **Language related...**
  - Coding languages provide similar abstraction mechanisms than models
  - Strong execution platform dependencies of developed code, e.g. UI, network, framework or library, etc.

- **Technology related...**
  - Considerable tool support at the code level
  - Certain non-connected tools are used for developing a system (limited or no integration)
  - Metamodelling vs. Profiling... "*all we need is UML*"
  - Metamodel usually does not specify the *platform behaviour*
    - Behaviour concepts facility for MOF missing
Approach: Modelling program and environment

- Specifications are usually not complete
  - main gaps in behaviour modelling

- Executable models provide abstraction
  - Simulation/Test of behaviour
  - Validation against specification model
  - Resolving environment dependencies of production code

- Platform is also modelled (typically goes first)
  - Bi-simulation as assumption for system correctness
Architecture Outline

Metamodel_1 <- Model_1

... <- Model_n

Metamodel_n

Syntax oriented Metamodel <- AST Model

Grammar <- Program

Platform Model_n <- Platform

Specication, Model (including operational semantic) Source code, Program

ANTLR, G<->MM, etc.

QVT
Metamodelling Framework for Model Execution

Model (Abstract Syntax) <<instanceOf>> Metamodel

Runtime Model <<executes>> Platform

Metamodel <<depends>> Metamodel

Behaviour Model <<depends>> Metamodel

MOF + OCL + Actions
Separation of syntax and behaviour

Syntax model

Runtime model (configurations)

Syntax model
Example: operational semantic of C# language

- Query
  - `self.object.theType.oclAsType(CSClass).member->select(method | method.oclIsTypeOf(CSMethod) and method.name = self.methodName)`

- Create
  - `Create Method Parameter`
  - `Create Method Parameter`:
    - `method : CSMethod`
    - `place : Place`

- Invoke
  - `Invoke Method : MethodExecution`

- Assign
  - `Assign Value : Value`

- Evaluate
  - `Evaluate Expression : Expression`

- Main
  - `Main Method : MethodDeclaration`
  - `Main Method : MethodDeclaration`:
    - `methodName:` String
    - `parameters` 0..* {ordered}

- Parameter
  - `Parameter : Parameter`

- Expression
  - `Expression : Expression`

- State
  - `State : State`

- Class
  - `Class : Class`

- Type
  - `Type : Type`
QVT relations map AST to model

- Import transformation provides abstractions
  - Simplification by unification and filtering of concepts
  - Visibility rules (public, private, etc.)
  - ...
- Environment decoupling
  - UI, I/O
  - Library functions, system calls
  - Network/Distribution topologies

- Replace (unimplemented) parts by simulated behaviour of the model

```java
relation While2Loop {
  enforce domain ast while:WhileStmt {
    children = exp : Expression (),
    children = gStatements : Statements (),
    parent = p : TreeNode ()
  };
  enforce domain metamodel loop:CSLoop {
    condition = c : CSExpression (),
    loopContent = mStatements : CSStatement(),
    eContainer = container : CSElement()
  };
  when {
    ContainmentMapping(p, container);
  }
  where {
    Statement2Statement(gStatements, mStatements);
    Expression2Expression(exp, c);
  }
}
```
Simulation and Test Framework Implementation

- Framework implementation based on eclipse EMF
  - Enterprise Architect as editor supporting metamodel and behaviour modelling (M3Actions)
  - OCL support for action queries
  - Extended instantiation concept
  - ANTLR for grammar definitions and ASTs
  - ikv++ QVT engine to transform models ([www.ikv.de](http://www.ikv.de))

- Former implementation
  - AMOF2 Repository (CMOF compliance level)
  - OSLO (Open Source Library for OCL) toolkit
Testing executable models: Anticipated Advantages

- Testing of existing program *against* the model
  - bridge between specification and product

- Reducing overhead for test specifications
  - Early integration testing
  - Capture meaningful sample data during execution of models
  - Reuse simulation scenarios and data for testing
  - Easier setup of “mockup models”
  - Alternative system state control flow

- Cope better with technology integration problems
  - Metamodel independent framework
  - Reuse of existing models across domains possible
Outlook

- Concept for *execution traces* underway
  - Foundation for comparison of executed behaviour

- Ongoing framework extensions
  - Move towards GMF as replacement of EA as modelling editor
  - Recording of execution runs
  - Checking of runtime status
Thank you!

Michael Soden
[soden@ikv.de]