Testing Model Transformations in Model Driven Engineering

Benoit Baudry
INRIA – IRISA
(currently visiting CSU)
Outline

- What about model transformation testing?
- Triskell’s contributions
  - Coverage criteria
  - Model synthesis
- Related work
- Challenges
Model Transformation Testing: Motivation

requirement 1.1 "Registering a book" the "book" becomes "registered" after the "librarian" did "register" the "book". the "book" is "available" after the "librarian" did "register" the "book".
Model Transformation Testing: Motivation

- A transformation is meant to be reused
  - But also has to be adapted from one project to another

- A transformation is meant to hide the complexity
  - We would like to trust the transformation as we trust a compiler
Dynamic testing process

- Test data
- Program
- Execution
- Result
- Specification
- Oracle
- Verdict
- Stopping criterion
- Problems
- Test data generation
- Localization / Debugging
- Diagnosis
- Test data evaluation
Dynamic transformation testing process

- Test data / test model
- Model transformation
- Execution
  - Result / output model
  - Specification
  - Oracle
    - Verdict
      - Stopping criterion
        - true
          - Test data evaluation
        - non vérifié
          - Localization / Debugging
    - false
      - Oracle

Problems

Test model generation
Dynamic transformation testing process

- Specific issues
- Complex data
  - Models are manipulated as sets of objects
- Complex constraints
- Lack of specific tools
Model Transformation Testing

- Currently in Triskell
  - Coverage criteria
  - Automatic synthesis of test models (in coll. With Mc Gill)
  - Specific fault models
Model transformation

Source metamodel structure + constraints

Transformation language

Target metamodel structure + constraints

Input model

Transformation

Output model

pre condition

post condition
Test data generation: criteria

- Several model transformation languages
  - Different features
  - Different paradigms
  - Different domains
- We did not want to choose
- We define black-box criteria
  - Independent of the model transformation language
Test data generation: criteria

● Define test criteria based on the input metamodel
  – Intuition: a set of models is adequate for testing if every class of the input metamodel is instantiated at least once and if the properties have relevant values

● A model for testing is called a test model
Test data generation: Example

What we expect from test models

- Every class to be instantiated
- Properties to take several relevant values
- Combine properties in a meaningful way

Possibly infinite set of models

=> Need stopping criteria
Adapt category partition testing to define ranges of relevant values for properties of the metamodel.
Relevant values for properties

- Define partitions for each property in the input metamodel
- A partition defines a set of ranges on a domain
  - choose one value in each range for the property
- Example
  - partition for AbstractState::label=\{[0],[1],[2..MaxInt]\}
  - A set of test models will need to have, at least three states with three different values for label
Relevant values for properties

Transition::event {"", '{"evt1"}', {'.'+'}}
Transition::source {1}
Transition::target {1}
AbstractState::label {0}, {1}, {2..MaxInt}
AbstractState::container {0}, {1}
AbstractState::incomingTransition {0}, {1}, {2..MaxInt}
AbstractState::outgoingTransition {0}, {1}, {2..MaxInt}
State::isInitial {true}, {false}
State::isFinal {true}, {false}
Composite::ownedState {0}, {1}, {2..MaxInt}
We would like to constrain the models to have a State with one outgoing transition **and** more than one incoming transitions.
Relevant object structures

```
ModelFragment
  `-- ObjectFragment
      `-- PropertyConstraint
          `-- Range
              `-- IntegerInterval
              `-- IntegerRange
              `-- StringRange
                    `-- BooleanRange
                        `-- BooleanValue
                            `-- values: BooleanValue

PropertyConstraint
   `-- constraints
   `-- isComposite: Boolean
   `-- isDerived: Boolean
   `-- default: String

Partition
   `-- range
   `-- isAbstract: Boolean

MultiplicityPartition
   `-- range

ValuePartition
   `-- intervals

org::omg::mof2::emof::Class
   `-- owningClass: 0..1
   `-- isAbstract: Boolean

org::omg::mof2::emof::Property
   `-- range
   `-- values: BooleanValue
   `-- default: String
   `-- isComposite: Boolean
   `-- isDerived: Boolean
```

- **ObjectFragment**
- **ModelFragment**
- **PropertyConstraint**
- **Range**
  - **IntegerInterval**
  - **IntegerRange**
  - **StringRange**
    - **BooleanRange**
      - **BooleanValue**
        - **values: BooleanValue**
- **Partition**
- **MultiplicityPartition**
- **ValuePartition**
- **org::omg::mof2::emof::Class**
- **org::omg::mof2::emof::Property**

Relevant object structures

- Criteria define structures that must be covered by test models
- These criteria combine partitions
- One criterion = set of constraints
  - one criterion declares the set of ranges that should be covered by a set of test models

- Example
  - Range coverage: Each range of each partition for all properties of the meta-model must be used in at least one model.
Test criteria

- Six test criteria (different combinations of ranges)
  - AllRanges
  - AllPartitions
  - + 4 class criteria
    - object fragments constrain each property of the object
- Do not consider constraints on the metamodel
  - Might generate insatisfiable fragments
Evaluating a set of models

- A prototype tool: MMCC
  - Framework for partitions and fragments definitions
- Computes a set of model fragments according to
  - Input metamodel
  - Test criterion
- Checks the coverage of a set of test models
  - With respect to the set of model fragments
Automatic synthesis of test models

● Automatic synthesis useful to
  – Limit the effort for test generation
  – Evaluate the test criteria

● Challenges:
  – Combine different sources of knowledge
  – Expressed in different formalisms
  – Complex constraints
Automatic synthesis of test models

- Meta-model
- Model Transformation Pre-condition
- Test Model Knowledge
  1. Test Model Objectives
  2. Model Fragments

specifies
specifies
specifies

Test Models
The Solution(1):
Combining Knowledge to Common Constraint Language

ECORE Model
ECORE Model
OCL Constraints
OCL Constraints
OCL Constraint
OCL Constraint
Requirements/Natural Language
Requirements/Natural Language
Objects and Property ranges
Objects and Property ranges

expressed as
expressed as
expressed as
expressed as
expressed as

Pre-condition
Pre-condition
Test Model Objectives
Test Model Objectives
Model Fragments
Model Fragments

transformed to
transformed to

Common Constraint Language:
Alloy
Common Constraint Language:
Alloy

Meta-model
Meta-model

INRIA
Model synthesis

The *run command*:

run test_requirement1 for 1 ClassModel, 5 int, exactly 5 Class, exactly 20 Attribute, exactly 4 PrimitiveDataType, exactly 5 Association

1. Specify a scope
2. Specify an exact number of objects

**Output:** Alloy model instance that satisfies meta-model + pre-condition + test_requirement1 and has the specified size
CARTIER: OVERALL FRAMEWORK

Input Meta-model MM_in
ECORE Model
OCL

Test Model Knowledge
Model Fragments
Test Model Objectives

Pre-condition OCL

transformed to
transformed to
transformed to

First-order Relational Logic Statements (in Alloy)
specifies a set of test models

boolean CNF formula

solved by SAT Solver

Alloy XML transformed to XMI

Ecore XMI

input to

Model Transformation MT(I,O)

Output Meta-model MM_out
ECORE OCL

Post-condition OCL

Graphs

specified a set of output models

CARTIER: OVERALL FRAMEWORK

INRIA
Perspectives on model synthesis

- Strengthen the tool
  - Automate what can be

- Experiments

- Design experiments to test model transformations

- We want to numerically estimate via mutation analysis the efficiency of test models
Mutation Analysis
Mutation Analysis

- Evaluate the set of models
  - Producing a Mutation Score

<table>
<thead>
<tr>
<th>Mutant</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Killed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- proportion: 5/8
Mutation Analysis

- Analysis based on fault models
- Faults are based on syntax of programming languages
  - Most common errors
  - For procedural languages, OO languages…
Mutation analysis for model transformation

- What errors occur in a model transformation?
  - Implementation language independency
    - Too many different languages
- Lack data on common errors
Abstract transformation operations

- Navigation, filtering, creation, modification
  - Example of one transformation

(a) Navigation

(b) Filtering

(c) Creation

(d) Navigation

(e) Creation

(f) Filtering

INRIA
Mutation operators

- **Navigation**
  - Relation to the same class
  - Relation to another class
  - Relation sequence modification with deletion
  - Relation sequence modification with addition

- **Filtering**
  - Perturbation in the condition
  - Delete a predicate
  - Add a predicate

- **Creation**
  - Replace an object by a compatible one
  - Miss association creation
  - Add association creation
One specific operator example

- **Navigation**
- Relation to the Same Class Change - RSCC
Mutation Analysis

- The proposed operator have been adapted to the Kermeta language
- Experiments:
  - To compare mutation operators
  - To evaluate the coverage criteria
  - To evaluate different knowledge for test generation
Perspectives in Triskell

- **Experiment!**
  - We have spent a lot of time defining ideas and building the tools
- **White-box techniques for specific languages**
  - Specific adequacy criteria
  - Fault localization
- **Oracle function definition**
- **Application with CNES**
Model Transformation testing in broad
Testing the transformation engine

  - Define fault models for pattern matching

- J. Steel and M. Lawley. *Model-Based Test Driven Development of the Tefkat Model-Transformation Engine*
  - Partition based test generation
Test data generation

- Templates to describe ‘patterns’ for test models
  - J.M. Küster and M. Abd-El-Razik. *Validation of Model Transformations - First Experiences using a White Box Approach*
  - Automatic instantiation of patterns with different combinations of values
  - Related to the notion of test model objective
Test data generation

- K. Ehrig, J.M. Küster, G. Taentzer, and J. Winkelmann. *Generating Instance Models from Meta Models*

- Consider only the structural definition of a metamodel
  - Additional constraints have to be checked a posteriori
Oracle

- Model comparison

- Major issue: producing the expected model
  - Complex, tedious and error-prone
  - Difficult to maintain
  - Except if another version of the transformation or an precise specification is available
Oracle

• **Execute the output model**
  - In case the output model is executable it can be tested

• **Issue for fault localization**
  - If a fault is detected in the output model it has to be located in the model, then back in its source in the transformation
No standard technique to write and evaluate contracts for a model transformation

- is OCL well adapted?

Different levels of complexity

- contracts on the output model
- contracts that relate input and output model elements
- design-by-contract for the transformation (if implemented using an OO language)
Still a lot to do: testing

- Methodology
  - Adapt to specific transformations / domains
- Systematic criteria
- Test environments
- Debugging support
  - Trace the detected error back to its source
- Experiments
Still a lot to do: development methods

- Specification of the model transformation
  - To derive contracts
  - To drive the generation of test models
  - To have accurate oracle functions

- Need two definitions of the transformation
  - Check the conformance of one according to the other = testing
Still a lot to do: tool support

- Tools to support oracle definition and test generation
  - Model comparison
  - Model visualisation

- Model type
  - Save a large number of verifications on input and output
References


