Correctness of Constraint-Aware Model Transformations

Xiaoliang Wang, Yngve Lamo

Department of Computer Engineering, Bergen University College, Norway

27 October 2011
Nordic Workshop on Programming Theory, Västerås, Sweden
Outline

Introduction

Diagram Predicate Framework

Correctness of Model Transformation
Model-Driven Engineering (MDE)

- In model-driven engineering, models are
  - Primary artefacts
  - Used to specify, generate and maintain code
  - Manipulated by model transformations
- Advantage
  - Productivity is greatly improved
  - Consistence between models is assured
Model Transformation

- Model transformation is automatic
  - Platform Independent Model (PIM) $\rightarrow$ Platform Specific Model (PSM)
  - Model $\rightarrow$ executable code
  - Model refactoring

- Improves the software development productivity and quality
Model Transformation

- Model transformations:
  - Source models $\rightarrow$ Target models
- Model transformation rules:
  - Source metamodel $\leftrightarrow$ Target metamodel
- Given a source model and a set of model transformation rules, we use the following transformation process:
  - Find a suitable rule
  - Change on the source model according to the rule
  - Generate a new model which satisfies the target metamodel
  - Repeat the process until there is no suitable rule found
Correctness of Model Transformation

- Software programs need validation before deployment
Correctness of Model Transformation

- Model transformations must also be reliable
Correctness of Model Transformation

- Model transformations must also be reliable
Correctness of Model Transformation

- Model transformation rules are designed manually
- In order to ensure reliability, it is necessary to check the correctness of the model transformation
Outline

Introduction

Diagram Predicate Framework

Correctness of Model Transformation
Diagram Predicate Framework (DPF)

- A fully diagrammatic specification framework for MDE
- Aims to be a diagrammatic formalism to define and reason about models and model transformations
Diagram Predicate Framework (DPF)

- Models are formalized as diagrammatic specifications which consist of an underlying graph structure together with a set of atomic constraints
A modelling language is formalized as a modelling formalism \((\Sigma_2 \triangleright S_2, S_2, \Sigma_3)\)

- Specification \(S_2\) represents the metamodel of the language
- Signature \(\Sigma_3\) contains predicates which are used to add constraints to the metamodel \(S_2\)
- Typed signature \(\Sigma_2 \triangleright S_2\) contains predicates which are used to add constraints to the specification \(S_1\) that are specified by the modelling formalism
Constraint-Aware Model Transformation

Joined Modelling Formalism

\[ \Sigma_3 \]

\[ \Gamma_3 := \Sigma_3 \cup \Xi_3 \cup \Theta_3 \]

\[ \Theta_3 \]

\[ \Sigma_2 \triangleright S_2 \]

\[ \Gamma_2 := \Sigma_2 \triangleright S_2 \cup \Theta_2 \triangleright T_2 \]

\[ \Theta_2 \triangleright T_2 \]
Diagram Predicate Framework (DPF)

Constraint-Aware Model Transformation

Model transformation rules

<table>
<thead>
<tr>
<th>$\mathcal{L} \triangleright \hat{J}_2$</th>
<th>$\mathcal{R} \triangleright \hat{J}_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rule 1 Class to table</strong></td>
<td><strong>Rule 2 Attribute to column</strong></td>
</tr>
<tr>
<td>$\mathbf{1:Class}$</td>
<td>$\mathbf{1:Class} \rightarrow \mathbf{1:Table}$</td>
</tr>
<tr>
<td>$\text{Int:DT}^t$</td>
<td>$\mathbf{1:Table} \leftarrow \mathbf{1:Class}$</td>
</tr>
<tr>
<td>$\mathbf{1:Col}$</td>
<td>$\mathbf{1:Col}$</td>
</tr>
<tr>
<td>$\text{Attr}$</td>
<td>$\mathbf{2:Col}$</td>
</tr>
<tr>
<td>$\mathbf{1:DT}^s$</td>
<td>$\mathbf{1:DT}^s$</td>
</tr>
</tbody>
</table>

$\mathcal{L} \triangleright \hat{J}_2$ represents the transformation of a class to a table, while $\mathcal{R} \triangleright \hat{J}_2$ represents the transformation of an attribute to a column. The rules are expressed using diagrammatic notation, where classes are represented by rectangles, attributes by diamonds, and tables by circles. The transformation rules are visualized with arrows indicating the direction of the transformation.
Outline

Introduction

Diagram Predicate Framework

Correctness of Model Transformation
Correctness of Model Transformation

A match of a rule:

- It exists a graph homomorphism from the left hand side of the rule to the model

If a match of a rule is found in a model, we say that the rule is applicable to the model.

A model transformation is correct if:

- For any valid source model, a sequence of applicable rules which constructs a valid target model can be found
Correctness of Model Transformation

Rule application strategy

- When several rules are applicable at the same time
- When several matches of a rule are found in the model
Which method to use

For correctness of program:

- Testing: Never completely identify all the defects
- Theorem provers: Need a mathematical formalization of the program and involves human activities
- Model checkers: State explosion problem
Which method to use

Model transformations are automatic

- Run automatic tests of model transformations
- A sequence of applicable rules to constructs a desired target model
- Feedbacks assisting the designers to correct the rules
Which method to use

- For any deterministic program, each input only have one execution path
- For a model transformation, several different sequences of applicable rules may exist
- Model checker can check all the possible sequences
Model checking is an automatic way to verify that a model satisfies a given specification

- Model is represented as a Kripke structure

- Specification is formalized in temporal logic, CTL or LTL

\[ E[\neg selection \ U brew] \]
Verification Process

Given

- Joint modelling formalism (JMF), including the source metamodel (SMM) and the target metamodel (TMM)
- Transformation rules (MTRs)
- Source model (SM)

A kripke structure can be constructed through this procedure
Verification Process

- We define an initial state $i$ representing SM
- For each state $s \in S$ and for every MTR $r : \mathfrak{L} \triangleright S_2 \leftrightarrow \mathfrak{R} \triangleright S_2$ we check $\text{IsMatch}(\text{Model}, \mathfrak{L} \triangleright S_2)$. If it is true, the rule is applicable.
- For each state $s \in S$ and for every applicable MTR $r : \mathfrak{L} \triangleright S_2 \leftrightarrow \mathfrak{R} \triangleright S_2$, we define a new state $r(s) \in S$ and a transition $t : s \rightarrow r(s)$.
Verification Process

Correctness property:
In the future there is a state where no more rule is applicable and from this state a valid target model can be derived. In CTL, it is formalized as

\[ EF! \text{AnyRuleApplicable(Model, MTRs)} \land \text{IsInstanceof(getTargetModel(Model), TMM)} \]
Future Work

- Find a suitable way to make rule application terminate
- Find way to implement the approach
- Find way to evaluate the approach
  - Efficiency of checking
  - Number of states handled by the model checker
Thank you!

Questions?

For more information visit: http://dpf.hib.no/