A Taxonomy of Model Transformations

Tom Mens, Krzysztof Czarnecki, Pieter Van Gorp

Presented by Wentao He
Dagstuhl Seminar

- Dagstuhl is a computer science research center in Germany, located in the scenic countryside of Saarland.

- Unlike most conferences, the focus is not solely on the presentation of established results but to equal parts on results, ideas, sketches, and open problems.

- Frequently praised by participants as the most productive academic events they have ever experienced!
About This Paper

- Summarized the results of the discussions of the 2\textsuperscript{nd} working group of the Dagstuhl Seminar on Language Engineering for Model-Driven Software Development.

- Proposed a taxonomy of model transformation.

- Aimed to help developers in deciding which model transformation approach to choose.

The new extension at Dagstuhl
Developers: “Which model transformation approach is best suited to deal with my problem?”

In order to decide which model transformation approach is most appropriate for addressing a particular problem, a number of crucial questions need to be answered.
What needs to be transformed into what?
Program and Model Transformation

- **Program transformation**
  - Artifacts are programs (i.e. source code, byte code, machine code).

- **Model transformation**
  - Software artifacts are models.
  - Encompasses program transformation.
“A transformation is the automatic generation of a target model from a source model.”

— A. Kleppe, J. Warmer, W. Bast


The above definition should be generalised!
Endogenous vs. Exogenous

- Endogenous transformation
  - Transformation between models expressed in the same language.
  - Example?
    - Optimization, refactoring, simplification and normalization

- Exogenous transformation
  - Transformation between models expressed in different languages.
  - Example?
    - Code generation, reverse engineering, migration
Horizontal vs. Vertical

- **Horizontal transformation**
  - Transformation where the source and target models reside at the same abstraction level.
  - Example?
    - Refactoring

- **Vertical transformation**
  - Transformation where the source and target models reside at different abstraction levels.
  - Example?
    - Refinement
A concrete example of all possible combinations:

<table>
<thead>
<tr>
<th></th>
<th>Horizontal</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endogenous</td>
<td>Refactoring</td>
<td>Formal Refinement</td>
</tr>
<tr>
<td>Exogenous</td>
<td>Language migration</td>
<td>Code generation</td>
</tr>
</tbody>
</table>
Differentiate whether the source and target models belong to one and the same or different technological spaces.

In the case of different technological spaces:
- transformation tools need to bridge the technological spaces
- actual transformation is executed in the technological space of either source or target model.
What are the important characteristics of a model transformation?
Level of Automation

- A distinction made between model transformations that can be automated and that need to be performed manually (or at least a certain amount of manual intervention).

- Example:
  - Transformation from a requirement document to an analysis model - manual intervention is needed
Complexity

- Light-duty versus Heavy-duty

- The difference in complexity is so big that they probably may require entirely different set of techniques and tools.

- Example:
  - Parsers, compilers - usually heavy-duty
Preservation

- Each transformation preserves certain aspects of the source model in the target model.

- The properties that preserved can differ significantly depending on the type of the transformation.

- Example:
  - Refactoring – behavior preserved, structure modified
What are the success criteria for a transformation language or tool?
Functional Requirements

- Ability to create/read/update/delete transformations (CRUD)
- Ability to suggest when to apply transformations
- Ability to customize or reuse transformations
- Ability to guarantee correctness of the transformations
- Ability to deal with incomplete or inconsistent models
Functional Requirements … 2

- Ability to group, compose and decompose transformations
- Ability to test, validate and verify transformations
- Ability to specify generic and higher-order transformations
- Ability to specify bidirectional transformations
- Support for traceability and change propagation
- More…?
What are the quality requirements for a transformation language or tool?
Non-functional Requirements

- **Usability and usefulness**
  - has to serve a practical purpose
  - should be intuitive and efficient to use

- **Verbosity versus conciseness**
  - *conciseness* means as few syntactic constructs as possible, but this often requires more work to specify complex transformations
  - need to find the balance between these two conflicting goals

- **Scalability**
  - should be able to cope with large and complex transformations
Non-functional Requirements ...

- **Mathematical properties**
  - prove theoretical properties of the transformation such as termination, soundness, completeness, correctness

- **Acceptability by user community**
  - the best transformation language from a theoretical point of view may not necessarily be the best from a pragmatic point of view

- **Standardization**
  - the transformation tool should be compliant to all relevant standards
What mechanisms can be used for model transformation?
The major distinction between transformation mechanism is whether they rely on declarative or an imperative (or operational) approach.

- **Declarative approach**
  - focus on *what* aspect
- **Imperative approach**
  - focus on *how* aspect
Some Declarative Approaches

- **Functional programming**
  - any transformation can be regarded as a function that transforms some input (the source model) into some output (the target model)

- **Logic programming**
  - features: backtracking, constraint propagation, bidirectionality of rules, and unification
Graph programming

**Advantage:**
- visual notation
- formally founded
- can compose smaller transformations into complex ones

**Disadvantage:**
- current techniques not compatible with each other
- current tool support not mature for industrial use
Conclusion

- Provided a taxonomy of model transformation.
- Should help developer choose a particular transformation language, tool or technology for his specific needs.
Discussion

- Do you find this *taxonomy* helpful?
- Any more thoughts or comments?