Understanding Model Transformation by Classification and Formalization

Shane Sendall, Rainer Hauser, Jana Koehler, Jochen Küster, Michael Wahler

IBM Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{sse, rfh, koe, jku, wah}@zurich.ibm.com

Software modeling techniques offer a means to address the size and complexity of modern day software problems through the use of abstraction, projection, and decomposition. Typically, multiple models are used to describe non-trivial software systems. However, if such models must be related and kept consistent by hand, then the viability of modeling as a means to reduce risks, minimize costs, improve time-to-market, and enhance product quality is nullified (most probably made even worse). As such, model-driven development approaches, in the direction of OMG's Model Driven Architecture initiative [OMG03], must be supported by tools that are at least able to automate the various tasks of keeping models consistent. Furthermore, the more that the various activities of model elaboration, synthesis, and evolution can be automated the more the above stated factors will be better addressed.

Our team at IBM ZRL [BPIA], entitled Business Process Integration and Automation (BPIA), is involved in research and development of model transformation techniques for model-driven development approaches in the domain of ebusiness solutions [GGK+03, HK04, KHK+03]. We are performing work on transforming business-level models to IT-level models and we are involved in the QVT-Merge submission for OMG's Meta Object Facility (MOF) 2.0 Query/View/Transformation Request for Proposal [OMG02]. Our effort in the latter area consists of work on a standard model transformation language for transforming MOF models, where MOF is an OMG standard for defining meta-models, i.e., the abstract syntax of a modeling language.

The problem of model transformation is similar to the one of program transformation and it also makes use of metaprogramming techniques [CH03, SK03]. However, it takes a slightly different direction by working with object-oriented metamodels, which define object graphs rather than syntax trees. We define the term model transformation in the following way: *A model transformation is a mapping of a set of models onto another set of models or onto themselves, where a mapping defines correspondences between elements in the source and target models*.

There are a number of different contexts of use that are applicable to QVT model transformations [Omg02]; these can be broken into two broad categories, inspired by Visser's classification for program transformation [Vis01]: *language translation*, and *language rephrasing*. In the former, a model is transformed into a model of a different language, i.e., a different model, and in the latter, a model is changed in some way, which may involve producing a new target model with the changes (distinct models) or changing the existing source model (single working model).

Like in [Vis01], language translation can be further sub-divided into *migration*: a model is transformed to another one at the same level of abstraction; *synthesis*: a model is transformed to another one at a lower level of abstraction; and *reverse engineering*: a model is transformed to another language at a higher level of abstraction.

Language rephrasing can be sub-divided into *normalization*: a model is transformed by reducing it to a sublanguage; *refactoring*: a model is restructured, improving the design, so that it becomes easier to understand and maintain while still preserving its externally observable behavior; *correction*: a model is changed in order to fix an error; and *adaptation*: a model is changed in order to bring it up to date with new or modified requirements.

The mapping between models established by the transformation may be required to be preserved over time. We call this characteristic of transformation *synchronization*

[GGK+03]. Examples of synchronization include: round-trip engineering and views. As part of synchronization, propagation of changes to a model may be made in one or more directions. Synchronization may be activated in a strict or loose fashion. *Strict synchronization* requires all changes to models to be taken into account immediately or in the next consistent state, e.g., views. *Loose synchronization* makes no statement on when synchronization should occur.

There are many different approaches available for model transformation; some of these include: relational/logic, functional, graph rewriting, generator/template-based, and imperative [CH03].

Our premise is that the different categories of model transformation in the QVT space are suited to different languages and approaches. As such, we believe that we should move towards understanding the requirements of each category and look at which kind of language is suited to which subset of problems. In doing so, we would like to understand the common requirements and also those that differ, and eventually build languages that are specifically address those specific problems.

Some questions that we are interested in addressing/discussing include:

- The field of compilation has a well understood categorization of languages. Building upon this work, how can one effectively formalize the different usage categories in the QVT space and the different model transformation languages so that we can more rigorously understand which ones match which domain? How "declarative" can we make a language targeted for such domains? What further categorization could we do with such formalizations?
- Bi-directional synchronization is a difficult problem in general. What existing approaches offer solutions? Is bi-directional transformation equivalent to uni-directional transformations in either direction? How do you avoid clobbering existing information on the return trip? How can trace information help in the return trip?

References

[BPIA] Business Process Integration and Automation, IBM Zurich Research Labs, Switzerland, 2004. http://www.zurich.ibm.com/csc/ebizz/bpia.html

[CH03] K. Czarnecki, S. Helsen; "Classification of Model Transformation Approaches". Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, USA, 2003.

[GGK+03] T. Gardner, C. Griffin, J. Koehler, R. Hauser; "A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard". MetaModelling for MDA Workshop, England, 2003.

[HK04] R. Hauser, J. Koehler; "Compiling Process Graphs into Executable Code". GPCE-04, 2004.

[KHK+03] J. Koehler, R. Hauser, S. Kapoor, F. Wu, S. Kumaran; "A Model-Driven Transformation Method". EDOC 2003, pages 186-197.

[OMG02] Object Management Group; "Request for Proposal: MOF 2.0 Query / Views / Transformations", 2002. http://www.omg.org/docs/ad/02-04-10.pdf

[OMG03] Object Management Group; "MDA Guide Version 1.0.1". 2003.

[SK03] S. Sendall and W. Kozaczynski; "Model Transformation - the Heart and Soul of Model-Driven Software Development". IEEE Software, vol. 20, no. 5, September/October 2003, pp. 42-45,

[Vis01] E. Visser; "A Survey of Strategies in Program Transformation Systems". Electronic Notes in Theoretical Computer Science, eds. Gramlich and Lucas, vol. 57, Elsevier, 2001.