Ideas for a Concrete Visual Syntax for Model-to-Model Transformations

Jorn Bettin
jorn.bettin@softmetaware.com

SoftMetaWare
PO Box 617
Waiheke Island
Auckland, New Zealand

ABSTRACT

The recent OMG work on Queries, Views and Transformations (QVT) has led to five proposals that contain suggestions for notations for model-to-model transformations. This work is a valuable foundation, but the success of the MDA initiative and of QVT in particular will depend on the availability of a concrete syntax for model-to-model transformations that is able to express non-trivial transformations in a clear and compact format. This paper investigates model-to-model transformations from a user's perspective, i.e. it presents ideas for a concrete syntax that would be useful for industrial production of business software.

Keywords

model-to-model transformation, meta modeling, domain-specific modeling, template language.

1. INTRODUCTION

In commercial software development the need for model-to-model transformation arises out of the desire to improve the clarity of transformation specifications, which typically are written in some kind of a textual template language. Anyone who has used template languages extensively to automate the generation of implementation code from highly abstract models will know that currently there is no standard paradigm for template code management. The wider the gap in the level of abstraction between source model and target implementation code, the more template readability and maintainability suffers.

In [9] and [10] we argue that despite these issues, it is still preferable to use template languages to automate model-to-text transformations rather than to leave the translation from model to implementation as a fully manual activity. This does not mean that there is no room for improvement. Hence the focus on model-to-model transformations, which allows to split a complex transformation into (A) a set of model-to-model transformations that map model elements of the source Platform Independent Model to model elements of the target Platform Specific Model and (B) simple model-to-text transformations that map the Platform Dependent Model onto the concrete syntax of implementation languages. In contrast of PIM-to-Text transformations, PSM-to-Text transformations can easily be expressed using a textual template language. In a nutshell the idea is to resolve complex one-to-many mappings in a partially visual language for model-to-model transformations, which significantly simplifies the subsequent PSM-to-Text transformations.

In what way do the QVT transformations represent a step forward?

- The QVT submission by DSTC/IBM/CBOP [2] only defines a non-visual syntax for model-to-model transformations, and therefore is similar to various text-based template languages for model-to-text transformations.
- The submission by Alcatel/Softeam/Thales/TNI-Valiosys/Codagen Technologies Corp [3] provides a visual notation for the assembly of "QVT components" into a chain of transformations. As far as we can see, "QVT components" are intended to be coarse-grained transformations such as a transformation from UML to a relational SQL model. The visual syntax does not cover the detailed specification of how model elements in the source model are mapped onto model elements in the target model.
- The submission by Compuware Corporation/SUN Microsystems [4] sketches a verbose visual notation for transformations that is similar to UMLX.
- The submission by the QVT Partners (Artisan Software/Colorado State University/Kinetium/King's College London/University of York) [5] defines in detail a verbose visual notation for transformations that is similar to UMLX.
- The submission by Interactive Objects Software GmbH/Project Technology, Inc. [6] and [7] provides no visual syntax for transformations.

This paper is not a research paper. Its main intention is to highlight practical issues that are not addressed by the QVT submissions, and to outline ideas for resolving these practical issues. The ideas presented in this paper will be developed further as part of the Generative Model Transformer project [8].

2. MODEL TRANSFORMATIONS AS FIRST-CLASS VISUAL MODELS

Model transformations involve matching patterns from a source model onto corresponding patterns in a target model. In [11] we introduce the term "texture" to denote patterns expressed in a precise UML-based format. Textual template languages allow the expression of patterns in textual source code. If a textual template language is used to map directly from one highly abstract PIM to textual source code, then the structure of the PIM is "drowns" in the textual template code, which is structured along the lines of [textual] implementation code.

Visual notations for transformations have the advantage of being able to represent patterns of the source model and patterns of the target model in a single diagram, and even to represent the mapping between source and target model elements in the same diagram. This is the main motivation behind visual languages such as UMLX [12] and others suggested in the QVT submissions. In practice however, all the currently proposed visual notations have the drawback of being quite verbose, which means that any non-trivial transformation ends up being scattered across a multitude of diagrams, and it becomes questionable whether the result is really better structured than transformations expressed in a textual template language.

If the syntax for model transformations is too verbose or not intuitive, the wider software development community will not buy into the whole concept, and model transformations will remain on the fringes of software development. We have to learn from what works in practice, such as UML class diagrams. Class diagrams are popular and they "work" because they offer users the right balance between textual and graphical notation, and even give the user a choice between textual and graphical notation - such as containment associations vs. attributes. In a class diagram, within one class "box" the user can capture at least three levels of containment:

1. attributes and operations,
2. operation parameters,
3. parameter details such as name and type.

In a good notation for transformations we need to use similar techniques to keep the number of graphical constructs (boxes and lines) from exploding. In practice, in most cases where there is some form of containment in the source model of the transformation, there will be a corresponding form of containment in the target model.

3. A COMPACT NOTATION FOR META MODELLING

If we want to arrive at a compact and practical visual notation for model transformations, we need to be able to

- represent non-trivial patterns of source and target models in a single diagram
- leverage the compactness of textual template languages for the expression of mappings between source and target model elements.

Normally, if the UML is used for meta modeling, each model element of a meta model is explicitly represented as a class. Why not learn from UML class diagrams and decrease the verbosity of meta models by optionally expressing containment by physically including the contained model elements in the class "box" representing the aggregate? Figures 1 and 2 show the two examples meta models similar to the ones that have been used extensively in the QVT submissions, using a notation that relies on the following concepts:

- *VLIST()* is used to indicate that a set of model elements is contained within an aggregate in the form of a vertical list - such as the attributes or operations section within a standard UML class "box".
- *HLIST()* is used to indicate that a set of model elements is contained in a horizontal list - such as the parameters in the signature of an operation in a standard UML class "box".
- Within a list, elements are comma-separated, and to keep the notation as compact as possible, the type of a modeling element is by default assumed to be String. Contained modeling elements of non-string type can be defined by using *<model element name><space><model element type name>*.
- Brackets are used to indicate nested containment within horizontal lists. How far to make use of this concept should be up to the user, i.e. the notation does not dictate the visual/textual balance.

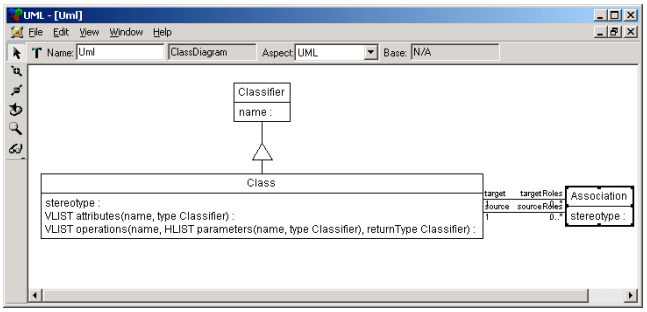

Figure 1 - Simple UML Meta Model

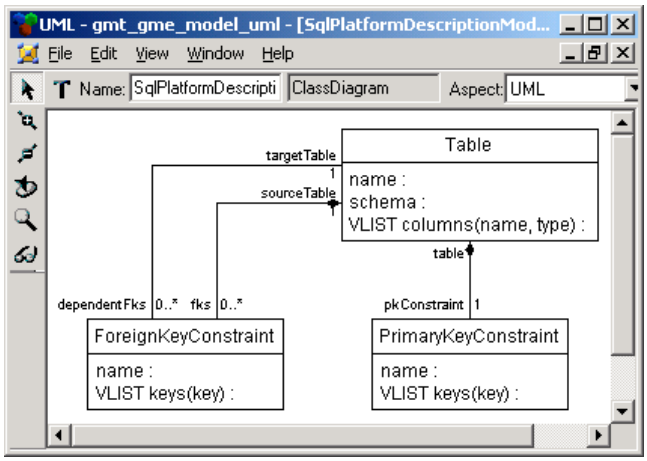

Figure 2 - Simple RDBMS Meta Model

Besides leading to compact meta models, the suggested notation allows the user to influence the physical design of domain specific modeling notations with a very simple and readable syntax. This is a nice side effect, the main advantage of the notation however is the textual notation for containment, which provides an opportunity to leverage textual template language concepts for the expression of the details of model transformations.

4. VISUAL REPRESENTATION OF MODEL-TO-MODEL TRANSFORMATIONS

A visual representation of model-to-model transformations can be achieved by depicting the meta models of source and target using the notation described in the previous section and then

- Using links between source and target meta model elements to indicate the creation of target model element instances as required by the transformation
- In the target model elements, adding transformation specifications by using OCL and an appropriate textual template language to map source model element instances onto target model element instances.

Figure 3 shows the example of transforming a simple UML model into a simple RDBMS model.

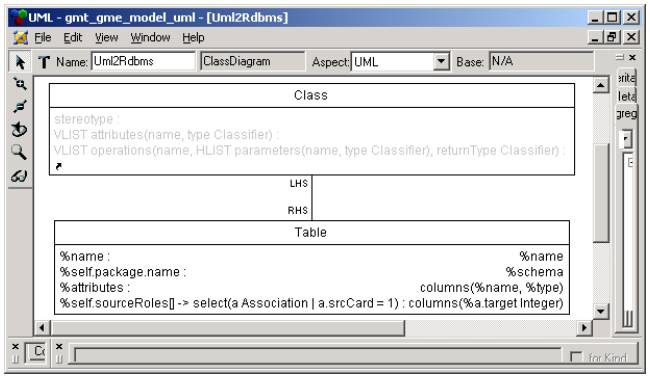

Figure 3 - UML to RDBMS Transformation

5. OUTLOOK

The QVT submissions have resulted in several proposed textual notations for model transformations. We believe that the most promising path to a practically useful notation is

- to use a visual notation to specify the creation of elements in the target model;
- to use a combined visual/text-based notation for modeling containment in meta models as suggested in this paper, such that the number of "boxes" does not explode;
- to leverage the proposed meta modeling notation, the OCL standard, and template language concepts to minimize the verbosity of model-to-model transformations.

…

6. REFERENCES

[1] Model Driven Architecture. www.omg.org/mda/

[2] DSTC/IBM http://www.omg.org/docs/ad/03-08-03.pdf

[3] OpenQVT http://www.omg.org/docs/ad/03-08-05.pdf

[4] XMOF http://www.omg.org/docs/ad/03-08-07.pdf

[5] QVT Partners http://www.omg.org/docs/ad/03-08-08.pdf

[6] IO/PT http://www.omg.org/docs/ad/03-08-11.pdf

[7] IO/PT examples http://www.omg.org/cgi-bin/apps/doc?ad/03-08-13.zip

[8] Generative Model Transformer http://www.eclipse.org/gmt/

[9] J. Bettin. Raising the level of abstraction of design models. OOPSLA 2001 Companion, (October 2001).

[10] Bettin, J.: Measuring the Potential of Domain-Specific Modeling Techniques. Proceedings of the Second Domain-Specific Modeling Languages Workshop, OOPSLA, Working Papers W-334. Helsinki School of Economics, (2002), 39-44, http://www.cis.uab.edu/info/OOPSLA-DSVL2/Papers/Bettin.pdf

[11] 4. Bettin, J.: A Language to Describe Software Texture in Abstract Design Models and Imple-mentation. Computer Science And Information Systems Reports TR-26, OOPSLA Workshop on Domain-Specific Visual Languages. University of Jyväskylä, (2001), 1-9, http://www.isis.vanderbilt.edu/OOPSLA2K1/Papers/Bettin.pdf

[12] Edward Willink, “The UMLX Language Definition”, http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/doc/umlx/umlx.pdf.