OMELET : Exploiting Meta-Models as Type Systems

Edward D. Willink

Thales Research and Technology (UK) Ltd
EdWillink@iee.org

Abstract. Meta-modelling is now well established for individual models. The MOF QVT proposal should support meta-model-based transformation between models. However, meta-model compatibility poses a major threat to the successful exploitation of transformation technology. We therefore introduce OMELET, a next generation 'make', that supports integration of diverse transformations and uses meta-models as a type system to ameliorate the threat and pave the way for automated composition of transformations.

1 Introduction

Activities such as the QVT proposal, XSLT schema support and the MDA have provided much needed impetus to model transformation. A model transformation supports the conversion of one (or more) input models into one (or more) output models, and each model is based on an associated meta-model as depicted in Fig. 1.

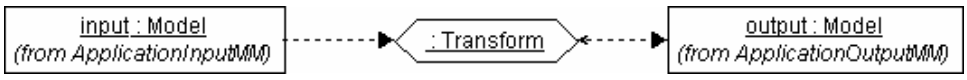

Fig. 1. Typical transformation invocation

In this paper we are interested in the problems that arise with multiple transformations, in particular the problem of meta-model compatibility between two transformations in a chain as depicted in Fig. 2.

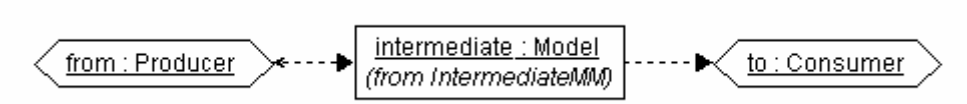

Fig. 2. Typical transformation interconnection

We seek to ensure that the `intermediate` model, produced by an instance of a `Producer` transform and consumed by an instance of a `Consumer` transform, is indeed based on the `IntermediateMM` meta-model.

It is convenient to say that our models are instances of our meta-models. However this is inaccurate; a meta-model is a package containing a variety of useful elements, some of which may be useful in a particular application. Bézivin [1] draws the distinction that a model is based on a meta-model. It is the elements in a model that instantiate elements of its meta-model and also comply with the associated constraints expressed in the meta-model.

We will briefly review the need for and hazards of multiple transformations, discuss some of the limitations of current technologies and suggest how the next generation of tools can address some of the problems.

2 Multiple Transformations

In [2] we introduced the Side Transformation Pattern as a technique to make model transformations modular and re-usable. This was achieved at the expense of changing a typical monolithic transformation involving two meta-models (input and output as in Fig. 1), into a composite transformation with four meta-models and three sub-transformations as shown in Fig. 3. The pattern therefore introduces two intermediate meta-models and four extra opportunities for incompatibility.

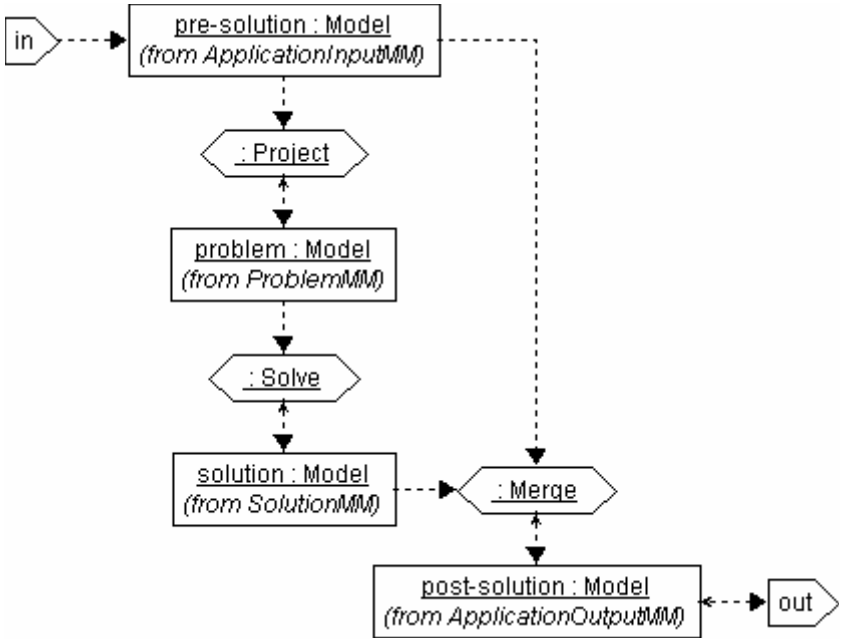

Fig. 3. Side Transformation Pattern

Increasing numbers of stages of transformation will be required as Model-Driven approaches are adopted with greater abstraction in a Platform Independent Model or in some Domain Specific Language in front of a PIM. These transformations will be more manageable if each stage resolves the concerns of a single form of abstraction. We may therefore expect the Model Driven Architecture to involve a chain of transformations to weave the various PIM, Platform Model and Mark Model concerns into a coherent Platform Specific Model. We can also expect the intervening stages in the chains to involve many distinct meta-models, or at least many distinct sub-sets of a smaller number of shared and often standard meta-models.

With many meta-models arising from transformation chains and further meta-models arising from using the Side Transformation Pattern to promote modularity and re-use, the integrity of these meta-models becomes critical to our endeavours. The problems of XMI dialects between early UML tools should act as a salutary warning.

3 Current Technology

Ensuring that models really are accurately based on their meta-models is difficult with current technology, and so there is rather too much reliance on the best endeavours of programmers and their intuition in choosing appropriate sub-sets of inconveniently large meta-models, such as UML. This provides ample opportunity for a joint development of

Producer and Consumer transformations to experience a rather troubled development. Problems are almost guaranteed when a more widespread attempt to re-use these pragmatic transformations is made.

The XML standard provides a good compromise between a human-accessible and a computer-accessible file representation. This makes it very appropriate for interchange between transformations where it is produced and consumed by computers, but needs to be intelligible by humans for at least debugging and sometimes manual interventions.

However, experienced XML users have discovered that XML conformance is a very weak discipline. It is all too easy for the conformant XML dialect of Producer and Consumer to differ, and as a consequence of the eXtensibility of XML, the difference in dialect is only detected after a number of intervening activities have conspired to make diagnosis difficult.

DTDs and now XSDs are therefore increasingly used to validate that the intervening files exhibit both semantic as well as syntactic consistency. This enables detection of errors in the Producer such as generation of spurious constructs and omission of mandatory constructs. However neither DTD nor XSD allow for more subtle validation of constraints on optional constructs. And of course no validation of the input can validate that the Consumer dialect is compatible.

XSLT provides its transformation capability within the XML Technology Space. Unfortunately the absence of comprehensive schema-aware support in current XSLT processors prevents diagnosis of seriously errant XPath expressions. This severely erodes the benefits that XSLT2 (or more readably, NiceXSL[4]) can offer.

Within the Modelling Technology Space, MOF-derived models provide for more accurate modelling in which OCL constraints capture subtle semantics. The lack of a direct model transformation capability should be addressed by the MOF QVT proposal. This should provide inherent rather than accidental compliance with the input and output MOF models and so introduce much needed discipline and efficiency to transformation programming.

When MOF models are converted to Java models to exploit the Program Technology Space, some inaccuracies in a Java-based Producer or Consumer can be avoided at compile time.

Until all transformations are defined in some language such as QVT that enforces model compliance, it is essential to perform as much model validation as possible in order to establish integrity for each intermediate model, and assist in diagnosis of inadequate transformations.

4 Tool Support

`make` and more recently `Ant` have established themselves as important parts of a programmer's tool kit. Both enable a number of programs to contribute to the solution of a larger problem. `make` also allows for some automated discovery of appropriate sequencing and invocation of those programs. However the composition of programs lacks discipline.

In `Ant`, the control flow (`depends`) defining the program sequencing is independent of the data flow (the task-specific input and output commands), so there is ample opportunity for typographic mistake and no inherent reason why the output of one program should be suitable as the input of another.

In `make`, the control flow is deduced from the file dependencies, so the control and data flow are consistent although sometimes surprising. The typical use of file name extensions to identify the data content of intermediate files encourages consistent usage, but there is still no inherent guarantee that the file extension correctly describes the content.

For transformations, we require the same ability to exploit a mix of custom and standard contributions, and we need to ensure that the usage of the transformations is valid. Meta-modelling provides the solution to these problems, since the appropriate meta-model

provides a strict definition of the permissible type of each intermediate 'file' in the composition.

We may therefore look towards a next-generation `make` in which rules are defined by registering the capabilities of particular model transformations in terms of the acceptable input and satisfied output meta-models. Using a very simple `make`-like example; given a pair of transformation signatures `(name = input-model-name : input-meta-model -> output-model-name : output-meta-model)`

```
compile = c_file : c_MM -> o_file : o_MM
link = o_file : o_MM -> exe_file : exe_MM
```

and a request to produce a model based on the `exe_MM` from a model based on the `c_MM`, we can deduce a suitable transformation chain to comprise `compile` followed by `link`. We can augment the chain with validation of input, intermediate and output meta-model compliance.

Many practical transformations are only appropriate for a sub-set of the syntax or semantics of particular meta-models. For instance simplified support for UML state charts might exclude History States, and an executable profile must exclude facilities with ill-defined semantics. This inhibits arbitrary model-independent chaining of transformations, but if the transformation chain is deduced within the context of the models to be transformed, the actual meta-model sub-sets are known and sub-set transformations can be exploited reliably.

We therefore require transformations to accurately define the sub-set meta-models that the transformation supports. Since this information will not be automatically available for many transformation technologies, we must be able to assert this as part of a transformation declaration.

Determination of the sub-sets in use by particular models should be a relatively straightforward model analysis to be performed by the transformation tool.

We must allow the user to specify a transformation chain, explicitly when they need complete control, implicitly when automation is acceptable, partially when they need to exert some influence, and historically when they need to repeat a previous sequence.

The non-implicit specifications provide intermediate way-points in the transformation chain, between which a transformation chain must be established. The tool must enable the user to view the actual chain, understand why certain transformations are necessary, and more importantly understand why certain transformations are unsafe.

This is the goal of the Eclipse/OMELET project [3]. Upgrading the capabilities of `make` to adopt meta-models provides the opportunity to deduce powerful transformation compositions. Adopting the Java extensibility approaches underlying `Ant` provides the opportunity to integrate transformations arising from a wide range of differing technologies. Using meta-models allows the transformation intermediates to be validated and transformation chains deduced.

At the time of writing a preliminary OMELET release is available that demonstrates the ability to register and invoke a diversity of transformations and meta-models. A rather more useful release should be available by the time this paper is presented.

5 Acknowledgements

The author is grateful to Thales Research and Technology for permission to publish this paper, which is influenced by work done on the SPEAR and GSVF-2 projects.

6 Conclusions

We have shown how meta-models can introduce discipline to transformation chains and motivated the development of OMELET, a next generation `make`-style program that uses meta-models to impose a type system on transformations that are implemented in a diverse range of technologies.

7 References

1 Jean Bézivin, MDA : From Hype to Hope and Reality, Guest talk at UML'2003.

http://www.sciences.univ-nantes.fr/info/perso/permanents/bezivin/UML.2003/UML.SF.JB.GT.ppt

2 Edward D. Willink and Philip J. Harris, The Side Transformation Pattern - making transforms modular and re-usable, submitted to SETra-2004, October 2004.

http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/omelet-home/doc/SETra2004/SETra2004-Pattern.pdf

3 The Eclipse OMELET Project.

http://dev.eclipse.org/omelet

4 The SourceForge NiceXSL Project.

http://nicexsl.sourceforge.net