Model Abstraction versus Model to Text Transformation

Jon Oldevik, Tor Neple, Jan Øyvind Aagedal

SINTEF Information and Communication Technology, Forskningsvn 1, N-0314 Oslo, Norway
{jon.oldevik|tor.neple|jan.aagedal}@sintef.no

Abstract. In this paper we discuss the principal differences between model to model transformation and model to text transformations, and sketch how these are pertinent to different model abstraction levels. We also try to clarify characteristics of model abstraction levels. Finally, we emphasize the role of model to text transformations in model-driven development.

1 Introduction

In system development the main goal of the activity is production of a running system. At this point in time, the most important assets for the running system are the developed or generated code that is compiled and executed. The importance of this has been recognized also by OMG, as signalled by the MOF Model to Text Transformation RFP [1]. This can also be seen in the tools that support model driven engineering today; transformation between model abstraction levels has not been in focus so much as support for generating implementation code. Within the current OMG MDA® regime, this has changed. The main focus has been the QVT standardisation for supporting transformations between models. We now need to address how the model to text transformation should integrate with this and provide a convenient and standard way of generating code from models on different levels.

2 Model Abstraction Levels

When looking at the MDA Guide [2], it is clear that OMG's current visions of model-driven architecture are quite open for interpretation. Its specific focus on different abstraction levels, the Computational Independent Model (CIM), Platform Independent Model (PIM) and the Platform Specific Model (PSM), does not reflect a natural distinction of abstraction levels. In principle, these can be models at any abstraction level, depending on your definition of platform.

A very stringent view of this is to consider platform-independent models to *be any models that still have variability with respect to the target execution platform.* A common interpretation of the platform-specific model is based on a loose definition, in the area of *a model that has implications related to a specific implementation technology, such as EJB.*

So, the separation between PIM and PSM points out some open issues. When does a PIM become a PSM? What is the difference between a PSM and the code representing the system? As long as there is no real way of executing a UML model, the most accurate model of the system will be the implementation code.

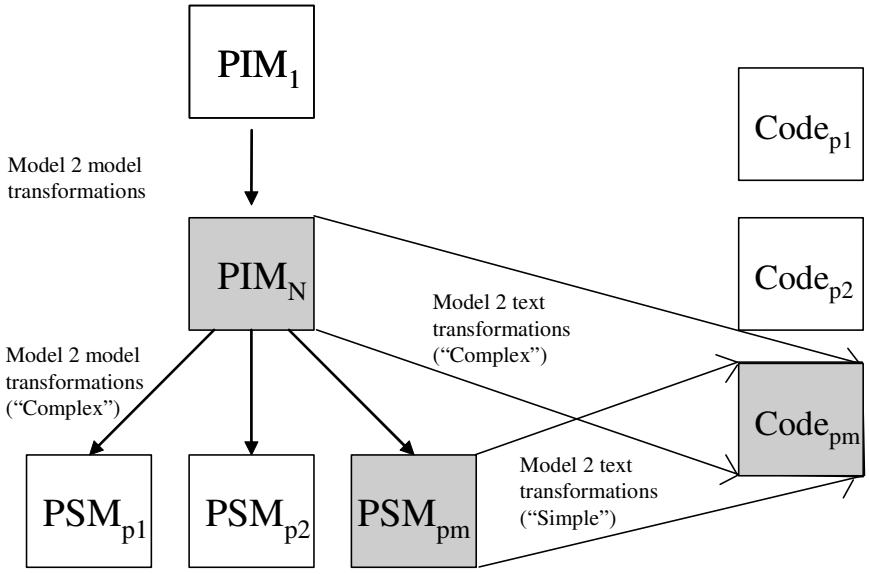

Figure 1 Transformations – From Model to Text

As illustrated by Figure 1 the transformation from model to text can be achieved from different abstraction levels. It is possible to generate code from a quite high-level architecture model, which can be considered a PIM. The transformation logic then will be of high complexity to bridge the detail gap from the PIM to the code. On the other hand, it is possible to create a platform specific model, based on the PIM and then generate textual code from the PSM. The complexity will then be in the model transformation between the PIM and the PSM. The mapping from the PSM to the code should be a simple matter, since the PSM and the code should be closer to semantic isomorphism.

Textual transformations should be possible from any model abstraction level. For a transformation architect, the challenge is to find the appropriate level, and to design the transformations. The complexity of transformations will increase proportionally to the abstractness of the models.

So, the exact timing, or level of model detail appropriate for transforming from PIM to a PSM rather than PIM to text is not given. Whatever level chosen, the transformation to text needs to be done at some point.

3 From Model to Text

The process of transforming a model to text can be required from any model abstraction level in a system development process; from business models, requirement models, high level architecture models, or detailed design models.

Today, there are numerous approaches for achieving this, implemented in different case tools, MDA tools, etc. In practice, however, the way of doing this is more or less the same:

- There is some kind of implicit or explicit representation of a metamodel (such as the UML metamodel)
- There is some kind of imperative language, such as a scripting or programming language, to write text generators within.

In case tools, there is typically an internal UML metamodel and a specific scripting language for writing text generators. For example, in Poseidon, they use the Velocity Template Language (VTL) to access the internal Java UML API. In IBM Rational Rose, they use Visual Basic extensions to access the internal Rose UML metamodel. Other approaches, such as in UML Model Transformation Tool (UMT), use externalized MOF/UML data on XMI form and some implementation language, such as XSLT, Java, VTL or other for text generation.

Consequently, we can see that there are many scripting languages used, such as XSLT, NiceXSL, Java Server Pages, Jython, VTL, JET, etc. They all target more or less the same problem area, although they have different strengths and weaknesses. The motivation for the MMTT RFP is to try to reach a more standardised way of achieving this, and hopefully leverage tool interoperability. We see today that models are used to generate all kinds of textual output, from different kinds of models. For example, requirements documentation and test cases from requirements models, API and system documentation from design models, as well as implementation code. So, we are looking for a standard that can provide generation of not only source code, but also text for humans.

4 The MMTT Language

A standardised language for model to text transformation needs to have a certain set of characteristics. There will be tradeoffs between expressive power and ease of use. One could argue that writing model to text transformations is not something that will be done by every developer, in fact probably only one or a few persons in a development department will have to deal with this task. Some will probably only use the standard generation scripts for standard platforms that come with the MDA tools. So, the ease of use issue should probably not be considered a limiting factor, but as simple as possible is still the preferred path.

The RFP asks for a language for transforming MOF models to text, which reuses the QVT language specifications. The resulting QVT standard language is therefore the natural extension point.

It is therefore natural to look into what is needed in addition to what is in the QVT proposal (QVT-Merge proposal [3]). QVT will include OCL expressions and the ability to create complex queries for model elements. It lacks, however, the ability to produce text output to files. Creating a specialisation of QVT *TransformationRule*, similar to a *Mapping*, except with the ability to create output, is one possibility.

Creating a "wish-list" for MMTT is only hard in having to try to make some limitations. For instance, there are many features in programming languages such as Java or in the surrounding libraries that would be useful, but we cannot standardize all of these. In the following we have tried to list the features that we mean are needed in model to text transformations:

- The ability to produce output to files from model elements. The language should allow for multiple target files from a model element.
- The ability to iterate model element sets. This is already an integral part of QVT.
- The ability to manipulate text, i.e. text functions such as strcmp, toUpper, toLower, strcat, substring. Some of these are already part of OCL and thus QVT.
- The ability to write and reuse functions. This is directly supported in QVT since *TransformationRule* is a subtype of *Operation*.
- The ability to interact with system services or libraries, for instance getting hold of the current date and time. This can be supported by extending the standard library of QVT with some extended necessary functionality.
- Support for parameterized transformations, where parameters will be provided at transformation time, for instance definition of package names in the Java language.
- The ability to include boilerplate text (such as copyright statements) into the resulting text files

Roundtrip engineering and reverse engineering issues are optional requirements in the MMTT RFP. However, these are indeed essential. These issues are related to MMTT, but are somewhat a different matter. However, the need for tracing what model element has generated what text-artefact is clear in a round-trip engineering context, and is a part that should be handled by the MMTT technology. The QVT standard library operations *markedAs* and *markValue* can be used to support some aspects of traceability.

The optional requirement of detecting and handling hand made changes in target files is more closely related to the subject. One may argue that this is outside the scope of a language for model to text transformation, but it is an essential requirement to a tool providing transformation capabilities.

Figure 2 shows a possible extension to the QVT-Merge submission. Here, we have added a *TextTransformation*, which specializes the *Mapping* class from the metamodel. In addition, it overrides the *body* AssociationEnd, which can be a *CompoundExp*. We have also introduced a *FileExpression* class, which provides a context for output to a file.

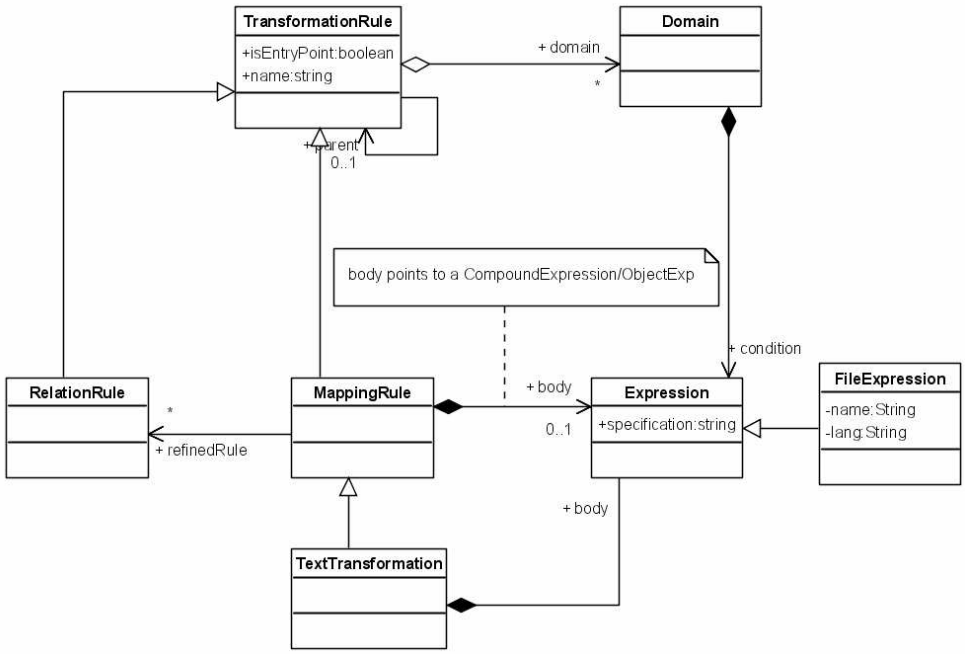

Figure 2 QVT-Merge metamodel specialisation

In the example below, a possible concrete syntax for the *TextTransformation* is shown. In this example, there are two file contexts within the TextMapping, providing two Java file outputs for each class in the source model. An important issue is how to standardise this part of the language. A possible approach is to allow hooks for embedding different languages within the MMTT. This is similar to how scripting is supported within HTML. As the simple example below shows, text mappings easily become complex and hard to read. In order to get the expressiveness needed, a language based on OCL-like constraints will not be sufficient. Additional imperative language constructs are needed.

```
Textmapping Simple_Class_to_Java
{
  domain { (SM.Class)[name = n, attributes = A] }
  body {
     let package_postfix = "qvt.org";
     let output_dir = "c:\test"
     file f [dir=output_dir, fname=n, ext="java", lang="MOFScript"] {
        print ("public class" + fname + "{");
        A->iterate(a | Simple_Attribute_To_Java (a, this));
        print ("}");
     }
}
TextMapping Simple_Attribute_To_Java
{
  domain {(SM.Attribute)[name = n, type = t]}
  domain {(File f)}
```

```
  body {
    f.print ("private " + Simple_Type_to_Java_Type(t.getName()) +
             " " + n.toLower() + ";")
  }
}
String Simple_Type_To_Java_Type (String typename){
   // Need logic to check different values and return different
   // type names based on this.
}
```

If MMTT standardises one language, lets say MOF Scripting Language (MOFScript), and provides extension points for other languages, this would lead to a flexible model.

5 Conclusion

Since it is really not defined how platform-specific a PIM needs to be to become a PSM, MMTT technology needs to have enough power to perform quite complex transformation tasks.

MMTT and QVT are closely related by topic, and the QVT language will provide many of the needed MMTT features. It is therefore natural to use the QVT standard as an extension point for creating MMTT. The additional features needed are already present in the group of transformation languages used in tools today; the key issue will be to define the set of wanted features for standardisation.

There will be tradeoffs between language usability and expressiveness. At one point, model transformation writing is not for everyone to worry about. On the other hand, if it gets too complex, no one will use it, leaving the whole process of standardisation pointless.

Acknowledgements. The work reported in this paper is carried out in the context of MODELWARE, an EU IP-project in FP62003/IST/2.3.2.3.

References

1. *MOF Model to Text Transformation Language - Request For Proposal*. 2004, Object Management Group: Needham,http://www.omg.org/cgi-bin/apps/doc?ad/04-04-07.pdf.
2. Miller, J. and J. Mukerji, eds. *MDA Guide Version 1.0.1*. 2003, Object Management Group: Needham.
3. QVT-Merge Group, *Revised submission for MOF 2.0 Query/Views/Transformations RFP*. 2004, Object Management Group,http://www.omg.org/cgi-bin/apps/doc?ad/04-04-01.pdf.