Generic Model-to-Model Transformations in MDA: Why and How?

Jernej Kovse
Dept. of Computer Science
University of Kaiserslautern
P.O. Box 3049, D-67653 Kaiserslautern, Germany
E-mail: kovse@informatik.uni-kl.de

1 Motivation

The OMG's *Model Driven Architecture* (MDA) [OMG01] defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform (e.g. CORBA, J2EE, .NET). Both specifications are expressed as models: *Platform Independent Models* (PIMs) specify the structure and functions of a system while abstracting away technical details; *Platform Specific Models* (PSMs) are derived from PIMs and specify how the functionality is to be realized on a selected platform. Thinking in terms of automating system implementation, MDA makes a strong case for *model-based generators* that will accept a PSM and generate code artifacts from it. In case the objective is in automating the implementation of systems that share common properties, but differ in (i) a predefined set of structural and functional features as well as (ii) implementation aspects, the question of dealing with the variability within such a system product line emerges. This position paper introduces the notion of generic model-to-model transformations for supporting variability within product lines in the context of MDA.

2 Generic Model-to-Model Transformations

MDA already identifies four types of model-to-model transformations (mappings) within the software development lifecycle [OMG01]: *PIM-to-PIM transformations* relate to platform independent model refinement and are applied when PIMs are enhanced, filtered or specialized; *PIM-to-PSM transformations* are used to project a PIM to the selected execution infrastructure; *PSM-to-PSM transformations* relate to platform dependent model refinement; *PSM-to-PIM transformations* abstract models of existing implementations into platform independent models.

Taking variability into account, developers should be given the possibility to specify a PIM and a PSM that are specific to a selected IT system family member with respect to its structural and functional features (captured in the PIM) and realization decisions (captured in the PSM). Models specific to such a member cannot be specified by applying a series of predefined transformations. For this reason, we suggest that PIM-to-PIM, PIM-to-PSM and PSM-to-PSM transformations should be generic (half-fabricated) rather than predefined. A *generic transformation* [KH02] is a representation of a set of specialized transformations that can be used to manipulate an existing model. Each of the specialized transformations belonging to the same set can be derived by configuring the generic transformation with a unique set of parameter values. Specialized transformations differ in a predefined set of structural and behavioral characteristics (expressed by attributes, cardinalities, constraints, etc.) of model constructs they introduce to the existing model; in this way, specialized transformations express properties that are specific to a given product family member.

In this scenario, PIMs and PSMs for family members evolve in a series of refinement steps, each step consisting of selecting the appropriate generic transformation, configuring it to derive a specialized transformation, and finally applying the specialized transformation. The configuration of a generic transformation will generally consist of two substeps: In *local configuration*, constructs to be added to the existing model are specialized independently of this model; in *global configuration*, the properties of the existing model are taken into account (e.g. what are the associations of new constructs with existing model constructs, how do new constructs specialize the existing constructs, etc.)

Developers benefit from generic transformations in two ways: First, the specification of a given system family member is semi-automatized; second, the way the model evolves on successive refinements steps is controlled (in general, manual model manipulation is allowed but limited via preconditions checked prior to applying the transformation) – this leads to *executable models* [MB02] that can be used as an input for model-based interpreters and compilers (e.g. code generators).

3 How to Express Generic Transformations

XML-based Metadata Interchange (XMI) [OMG02] supports the interchange of any kind of metadata that can be expressed using the MOF specification, including both model and metamodel information. In addition to supporting the exchange of complete models, XMI supports the exchange of differences between two models. For this purpose, it provides four elements (we refer to them as *differential elements*): `<XMI.difference>`,

`<XMI.delete>`, `<XMI.add>` and `<XMI.replace>`. There are two scenarios of using differential elements: (a) *Model differencing*: The differences between the new and old model (both of which already exist) have to be evaluated; (b) *Model merging*: The new model is constructed by applying a series of differences to the old model, provided that the old model and the difference information already exist. The scenario (b) can be used as an implementation strategy for model-to-model transformations: We express a specialized transformation as a series of XMI differential elements and apply it to an existing model to obtain the successive model.

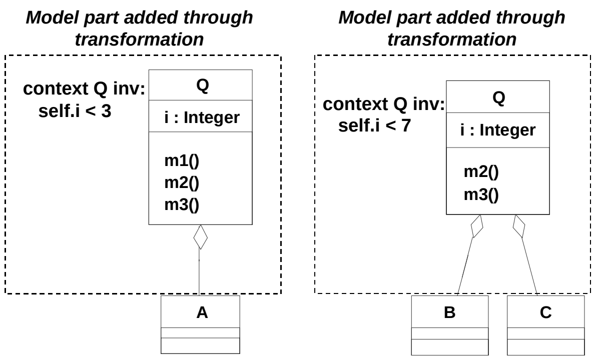
Model part added through transformation
context Q inv: self.i < 3
Q
i : Integer
m1()
m2()
m3()
A
Model part added through transformation
context Q inv: self.i < 7
Q
i : Integer
m2()
m3()
B
C

Fig. 1. Applying two specialized transformations.

To support the production of multiple specialized transformations (each of which is expressed in XMI) from the same generic transformation, a generic transformation must support the following concepts (To illustrate these concepts, an intentionally simple example in Fig. 1 shows the application of two specialized transformations, produced from the same generic transformation; `Q` is the class added in each transformation):

- *Static parameterization*: It should be possible to define *placeholders* within a generic transformation that are going to be substituted by concrete properties of a specialized transformation (e.g. concrete class names, role names, visibility kinds, static invariants, pairs of operation pre- and post-conditions, etc). For example, the invariant in Fig. 1 limits the value of `i` to two different ranges.
- *Iteration*: The number of desired occurances of a segment of a generic transformation in the specialized transformation may vary (e.g. in case the purpose of the transformation is to add a container class for a set of classes already present in the existing model, the number of aggregation relationships to be created to associate the container with these classes will depend on the total count of the latter). Note that in the first specialized transformation in Fig. 1 a single aggregation relationship had to be created, while the second specialized transformation required two such relationships.
- *Conditional includ*e: It should be possible to force the omission of a certain segment (e.g. an attribute, a method, a static invariant, a precondition, etc.) of a generic transformation from the specialized transformation. For example, in Fig. 1, the method `m1` has been omitted from class `Q` in the second specialized transformation.

An implementation strategy for generic transformations includes:

- Extending XMI DTDs (e.g. UML DTD) with element tags that will mark the use of above concepts. A generic transformation is then expressed as an XML document conforming to this DTD.
- Writing a single XSLT [W3C02] transformation that will accept (i) a generic transformation and (ii) configuration parameters (both in XML) as source trees and produce a specialized transformation as the result tree. XSLT supports the concepts we identified above as follows: Static parameterization is supported via *XSLT template rules* (element `xsl:template`); iteration is supported via *XSLT template rules* and *XSLT repetition* (element `xsl:for-each`); conditional include is supported via *XSLT conditional processing* (elements `xsl:if` and `xsl:choose`). The XSLT transformation that can be reused with any generic transformation.

Each generic transformation may define a set of pre- and post-conditions. Configuration of a generic transformation not only specializes the transformation, but also specializes these conditions. Specialized preconditions are used to check whether the initial state of the model allows the application of the specialized transformation; specialized postconditions are used to check the consistency and integrity of the obtained model. Both pre- and post-conditions can be expressed in a dedicated constraint language appropriate for the models (in the case of UML, OCL is the obvious choice).

4 Challenging Questions: What Kind of Infrastructure is Needed?

The process of specification refinement on the basis of generic model-to-model transformation should be alleviated by providing a dedicated tool infrastructure with amenities described below:

- The model repository should support at least basic version management capabilities: Users should be able to revert to previous versions of transformed models; it should be possible to carry out distinct transformations on two separate parts of a model and afterwards recombine both parts into a configuration.
- Visual tools should be capable of demarcating model parts that have beed added to the model through different specialized transformations in different colors. It should be possible to trace the history of a part

back to (i) generic transformations that were used to add or manipulate the part as well as (ii) parameter values that were used to configure these transformations.

- In the course of domain analysis [CE00], common and variable properties of the systems and the dependencies between the variable properties are captured in a domain model. A feature model, which is a component of the domain model, describes system features, defines feature composition rules, indicates why a feature should or should not be chosen and hierarchically decomposes the features in form of a feature diagram. There is an exciting application of associating generic model-to-model transformations with feature models: Suppose designers of generic transformations define mappings of transformations (along with parameter values) to sets of features. This allows the developers to track what system features have been covered in a PIM or PSM gradually refined by specialized transformations.
- Implementation of a family member will be automated by providing a set of reusable assets (e.g. reusable components, generators, etc). In case domain implementation is highly heterogenous (consisting of a large number of reusable components, multiple cooperating generators, etc.) relationships of generic transformations to reusable assets that will be applied in the automated implementation of a family member should be carefully defined. This allows developers to associate implemented product parts with generic transformations that were used to manipulate the part on the specification level.
- To arrive at a consistent specification of a family member, generic transformations cannot be chosen and configured in a random order. Tool infrastructure should support the definition of a workflow model to track the refinement of a PIM or PSM through transformations. The workflow model defines which generic transformations can be chosen on certain refinement step and thereby determines the allowed sequences of transformations. In addition, it may specify the steps where manual model manipulation is permitted.

5 Conclusion and Planned Contribution to the Workshop

This position paper introduced the notion of generic model-to-model transformations as a strategy to support the variability within system product lines at the model level in the context of MDA. A generic model-to-model transformation can be configured with parameter values; this allows the production (via XSLT) of specialized model-to-model transformations, expressed as a series of XMI's differential elements.

I see my contribution to the workshop in:

- Critically exposing the issue of whether generic model-to-model transformations are an efficient approach aiming at variability in system product lines in the context of MDA.
- Discussing the challenging issues related to tool infrastructure described in Sect. 4.

References

[CE00] Czarnecki, K., Eisenecker U.W. : Generative Programming : Methods, Tools and Applications, Addison-Wesley, 2000.

[KH02] Kovse, J., Härder, T: Generic XMI-Based UML Model Transformations, in: Proc. OOIS'2002, Montpellier, Sept. 2002, Springer-Verlag, pp. 192-198.

[OMG01] OMG Architecture Board MDA Drafting Team: Model Driven Architecture – A Technical Perspective, OMG Document ormsc/01-07-01.

[OMG02] OMG: XML Metadata Interchange (XMI) Specification, OMG Document formal/02-01-01.

[MB02] Mellor, S., Balcer, M.J.: Executable UML: A Foundation for Model Driven Architecture, Addison-Wesley, 2002.

[W3C02] W3C: XSL Transformations (XSLT) Specification, version 2.0, Working Draft.