MDA Support by Transformation Based Tool

Audris Kalnins, Janis Barzdins,

University of Latvia, IMCS,   29 Raina boulevard, Riga, Latvia
{Audris.Kalnins, Janis.Barzdins}@mii.lu.lv

Abstract. The paper presents an example reuse-oriented Model Driven Software Development (MDSD) methodology. Requirements for modelling language and support tools are discussed and it is shown that development methodologies of this kind can be adequately supported by a metamodel and model transformations based tool framework (TTF), briefly described in the paper.

1 Introduction

Though Model Driven Architecture (MDA) and Model Driven Software Development (MDSD) have been buzzwords for several years, there are not so many true success stories. For a true MDSD methodology (development as a series of models) to become usable, there are serious requirements to the modelling language and to the tool support. UML has a lot of shortcomings in this role [1,2], and standard UML tools (IBM Rational RSA, Enterprise Architect, …) require extensions [2] comparable in efforts to the tool itself. If we want to combine MDSD with proper reuse of all development artefacts (models), the situation is even more complicated (extended requirements for traceability).

After presenting an example reuse-oriented MDSD methodology, this paper proposes a Transformation based Tool Framework (TTF), in which such a methodology could be easily implemented and with a rich user support.

2 An example methodology for MDSD

Let us start with a brief sketch of an example "reuse oriented" MDSD methodology. This methodology suggests that a system development is a *sequence of models*, where each model is obtained from the previous one partially automatically according to defined rules. At the same time, each model is then manually extended by modelling elements whose incorporation is most appropriate at this step. Fig. 1 shows a very generic development picture of this kind – there four models are present.

To be more specific, the model M1 is assumed to be a (formalized) *Requirements model*. This model consists of UML use cases, organized by means of UML use case diagrams. The behaviour of each use case is precisely specified by means of activity diagrams. All the concepts (objects, types etc.) appearing in these diagrams should be present in a simple (conceptual) class diagram (model). Thus actually a subset of the whole UML is used, or more precisely, only a *fragment* of the UML metamodel is used (e.g., the fragment could consist of UseCase, BasicActivities and Core packages

of UML metamodel). The model M1 actually is an instance of this metamodel fragment.

On the one hand, it seems to be a pretty standard use of UML. But for such a model to be precise and usable, the standard UML 2.0 is clearly insufficient. For example, we could want to provide a direct link from UseCase to its refining Activity (so that the user could easily navigate), but there is no such association in the UML metamodel. Similarly, we could require associations between Actors of use cases and Partitions (performers) of the relevant actions. All such extensions cannot be defined as UML 2.0 profiles because they require new associations added to UML metamodel. Therefore the only solution is to use a direct "*heavyweight*" extension of the metamodel.

Similarly, the next model (M2) is a platform independent (PIM) *Design model*, based on UML sequence and (full) class diagrams. There are several methodologies [3] providing guidance how operations should be in the best way assigned to classes, using sequence diagrams as an easy readable design document for this process. The by-product of this approach is that the resulting set of sequence diagrams contains the complete "control logic" for method bodies. To fully apply such methodologies, again some extensions to sequence diagram metamodel are required.

It should be noted that the "initial version" of Design model can be built automatically from the Requirements model – we already know some classes and their required operations. There may be more such heuristic rules if Requirements model is formal enough. Certainly, the complete design is clearly manual.
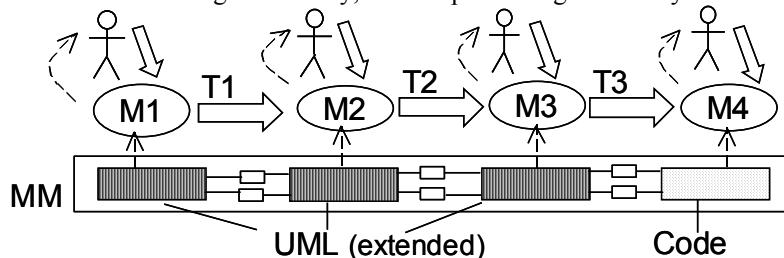

Fig.1 Example methodology

The next model (M3) could be the *PSM model*, where platform architecture elements are merged with the pure functional design provided in M2. The degree of automation in obtaining this model from M2 is very high. And finally, M4 is the OOPL (Java, C#,…) code model. Again, the percent of code obtained automatically from the design models is much higher than in conventional forward engineering since sequence diagrams provide a significant part of method bodies. Another highly desired automation support are various wizards, which provide local advices to the designer according to the methodology. If, in addition, scenarios from [2] are used as the initial model, Requirements model could be built from them almost automatically.

One more reuse-oriented aspect of this development process is the necessity to document all applications of automated transformations in the models themselves. Only in such a way the *traceability* is ensured – we can find out e.g. which elements of code are determined by the given system action in the requirements. Another highly desired usage of this feature is *change propagation*. The most natural solution again is to extend the UML metamodel by mapping (traceability) elements – classes

and associations linking the appropriate elements of models in the development chain. These traceability extensions are shown as "linking elements" in Fig.1.

3 Transformation based Tool Framework

The sample development process described in the previous section is hardly implementable by conventional modelling tools – the efforts would be too high, especially for heavyweight metamodel extensions.

To overcome these problems, we propose a new approach to modelling tool building by combining the well-known metamodel based generic tool idea [4,5,6] with the completely novel idea of using *model transformations* for defining the complete tool functionality. To test this idea, recently UL IMCS has started a new project within the Latvian State Research Program in Information Technologies for building a *Transformation based Tool Framework* (*TTF*). Fig.2 presents a general schema of this framework.
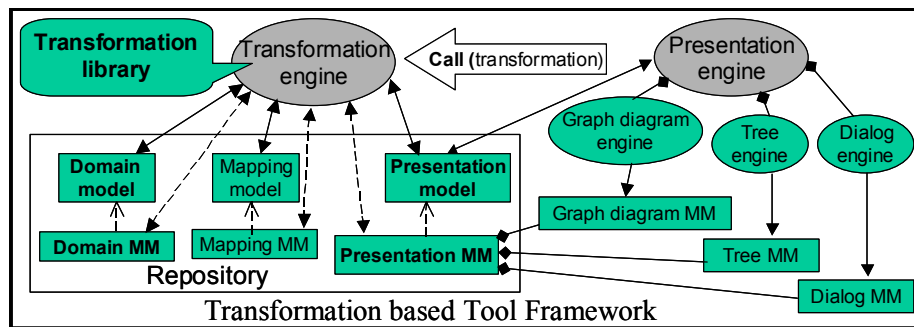

Fig.2 General schema of TTF

Existing metamodel based modelling tools [4,5,6] typically support arbitrary domain metamodel, fixed presentation metamodel (usually diagram) and some sort of simple mapping definition between them. On the contrary, TTF will provide a very flexible correspondence between a domain model (such as the UML domain specified by the UML 2.0 metamodel) and its various visible presentation models (diagrams, model tree, textual property dialogs etc). This correspondence will be defined in an easily usable graphical model transformation language (for TTF in the MOLA language [7]). The parts of presentation model (diagrams, model tree,…) are specified by the corresponding presentation metamodel parts (Graph diagram metamodel, Tree metamodel,…). Each part is serviced by the corresponding presentation engine, which simply visualizes the relevant presentation model and notifies back to transformations the user actions on this visual view. Since all the nontrivial correspondence logic between models is handled by transformations, the engines will be relatively simple. Certainly, for all this to be usable in practice, an efficient implementation of transformation language and a fast model repository is required. Both are now under construction and the first experiments confirm the viability of the approach.

The main gain from this approach is that defining the correspondence between a domain and its visible representation in an appropriate transformation language is by

an order of magnitude faster than in an OO programming language. First experiments show that e.g. a UML Class editor could be built in a week.

4 MDSD support tool on the basis of TTF

Let us assume that a UML modelling tool has already been built on the basis of TTF (it is one of the first intentions at IMCS). Then obtaining an MDSD tool supporting the methodology sketched in section 2 would be a quite straightforward job.

The first advantage is the ease of metamodel extension – we just have to add the relevant elements (in our case, associations) using the metamodel definition facilities of TTF. Certainly, some transformation language procedures specifying how the added elements must be visualized (e.g., as additional navigations between existing model elements) must also be added. Larger metamodel extensions together with supporting transformations can be "packaged" for reuse.

The "main" transformations between models of the MDSD chain are added as responses to the relevant user actions, these transformations build a new model (only its domain part, support for building the presentation part from the domain already exists). The chosen transformation language will permit to incorporate easy various heuristic approaches. If traceability elements are included in the extended metamodel it is quite easy to provide the model traceability support in these transformations.

One more advantage of using TTF is the ease of building "intelligent" user support. First, there could be prompters (wizards) which support the manual model building steps according to the methodology (a prompter actually is a transformation filtering out the relevant model elements). "Methodology checkers" can be built in a similar way. User support for traceability could be implemented as a set of queries, with results in textual form or as special diagrams (also implementable easy in TTF).

We plan to complete the first version of TTF within the next half-year and to check it on real tool examples like the support for MDSD methodology described in this paper. We hope that a framework like TTF is the best way to merge extended modelling facilities with a wide use of model transformations.

References

1. R. B. France, S. Ghosh, T. Dinh-Trong: *Model-Driven Development Using UML 2.0: Promises and Pitfalls*. IEEE Computer, Vol. 39, 2 (2006) 59-66
2. M. Smiałek, J. Bojarski, W. Nowakowski, T. Straszak: *Scenario Construction Tool based on Extended UML Metamodel*. LNCS Vol. 3713, Springer (2005) 414-429
3. C. Larman: *Applying UML and Patterns*. Prentice-Hall, 3rd Edition (2004)
4. A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, et al.: *The Generic Modeling Environment*. Workshop on Intelligent Signal Processing, Budapest, Hungary (2001)
5. Graphical Modeling Framework (GMF) URL: http://www.eclipse.org/gmf/
6. E. Celms, A. Kalnins, L. Lace: *Diagram definition facilities based on metamodel mappings*. Proc. OOPSLA'2003, Workshop on DSM, Anaheim (2003) 23-32
7. A. Kalnins, J. Barzdins, E. Celms: *Model Transformation Language MOLA*. Proceedings of MDAFA 2004, LNCS, Vol. 3599, Springer (2005) 62-76