DSL tool development with transformations and static mappings

Elina Kalnina

University of Latvia, IMCS, Raina bulvaris 29, LV-1459 Riga, Latvia
Elina.Kalnina@lumii.lv

Abstract. A tool development platform for domain-specific languages combining mapping and transformation based approaches is proposed in this research project. Combination should be made to use advantages and eliminate disadvantages of both approaches as far as possible. Initial results are described.

1. Introduction

Currently it is very popular to create and use specialized modelling languages for a domain area. Theses languages are called *domain-specific languages* (DSL). By using domain-specific languages users can operate with familiar terms. There can be graphical or textual domain-specific languages. Only graphical languages will be considered here. Operational semantics of DSL is also out of scope of this research project. A visual domain-specific language basically consists of two parts – the domain part and the presentation (visual) part. Sometimes they are called also the abstract and concrete syntax respectively.

The domain part of the language is defined by means of the *domain metamodel*, where the relevant language concepts and their relationships are formalized. Standard MOF [1] or similar notations are used for the definition of domain metamodel.

For the presentation part (concrete syntax) definition there is no universally accepted notation. The same metamodelling techniques are used, but with various semantics. Instances of classes in the presentation metamodel are *types* of diagram elements to be used in the diagram. A concrete set of graphical element types for a diagram definition is called the presentation type (or graphics) model (similarly to GMF [2]).

Due to the growing popularity of domain specific languages various graphical tool building platforms have been developed to improve the tool building process. Two different approaches are used in these environments. The first option is to use a mapping-based approach. This solution is quite appropriate for simple cases, where no complicated mapping logic is required. In this case tools for simple DSLs can be developed even during a presentation session. However, DSL support frequently requires much more complicated and flexible mapping logic. In this case the second approach is used: to define the correspondence by *model transformation languages*.

Mapping based platforms are MetaEdit [3], GMF platform [2], Microsoft DSL Tools [4], Generic Modeling Tool [5] and some other. Transformation based

platforms are METAclipse platform [6], Tiger project [7], ViatraDSM framework [8] and GrTP [9]. A detailed overview of platforms is given in [6].

The purpose of this paper is to show how these approaches could be combined.

In chapter 2 both approaches are briefly sketched. In chapter 3 ideas how these approaches could be combined are shown.

2 . Mapping and transformation approaches

A *mapping*-based approach prescribes by means of which presentation type model element each domain metamodel element must be visualized. Thus, the graphical tool functionality is basically defined by this mapping. The mapping itself can be defined as a mapping model according to the mapping metamodel. The mapping typically may be complemented by use of constraints, but only at few selected points.

Most of the frameworks (GMF, MS DSL,…) use the *generation step*, by means of which language classes are generated in the corresponding OOPL (Java, C#,…) from the involved models. The generated code ensures the relevant synchronization between the domain and presentation models in runtime. If the generated functionality is insufficient, the language code can be extended manually. Actually, mapping may be used without the generation step too - examples are MetaEdit+ [3] and Generic Modeling Tool [5], which are model interpreters.

It must be noted that the mapping approach is easy to use - if the generated code is sufficient (or should be accompanied by a small amount of manual code), the tool definition is mainly declarative and very fast. However, when the presentation type model is dissimilar to domain metamodel, a lot of code in an OOPL must be added.

A complete alternative to the mapping-based approach is the *model transformation* based approach. The correspondence between the domain and presentation is defined by *transformations* in a model transformation language, for example, MOLA [10, 11]. These transformations define what modifications must be done in one of the models, if the other changes (due to user actions or other internal activities). Therefore the correspondence between the domain metamodel and presentation type model may be arbitrarily complicated here. In fact, transformations control the complete tool behaviour.

From the first glance this approach is more complicated to use - though experience shows that programming model element mappings in an adequate model transformation language is much easier than in a standard OOPL. The usability of the approach is ensured also by the fact that a significant part of the transformations are domain-independent and are built only once, as part of the framework itself. Clearly, the transformation driven approach is more time consuming in simple cases.

Usually, for some parts of the tool the correspondence from domain to presentation is simple (fit for mappings) and for some complicated (fit for transformations). The best solution would be to combine both approaches. In this case for simple one-to-one relations between domain and presentation the mapping based approach could be used, but for complicated parts model transformations could be written. For example, for MOLA Editor [6] (built using transformations in METAclipse) the transformation size could be reduced approximately by 50% if mappings were applicable. Simple

visualisation could be defined by mappings, but for complicated consistency maintenance transformations would still be needed.

Currently there are only known two attempts to combine both approaches in a limited way. The latest versions of Tiger project [7] propose to add more complicated user commands to the mapping-based GMF environment by transformations in AGG language. ViatraDSM [8] proposes to extend basic mapping facilities in Eclipse GEF by means of graph rules in Viatra.

3 . Research project description

The main topic of the given research project is how to add mappings to a transformation based tool development platform. The METAclipse platform [6] built by UL IMCS is chosen as the basis for research project realisation. It is completely based on transformations and uses the transformation language MOLA.

Mapping definition support will be added to this platform in the given research project. To ensure usability mappings and transformations should be smoothly integrated.

There are several options how to do this. The first one is to generate transformations from mappings. In this case the generated transformations can be later on modified manually. There is a problem with updates, when transformations should be regenerated. This approach is similar to one used in mapping-based tools.

The second option is to add extension points where custom transformations can be added to the mapping definition. Extension points are places where built-in mapping possibilities can be replaced or extended by custom transformations. In this case, there is a nontrivial problem how to choose extension points and how to integrate them with the defined mappings. In this case an interpreter or generator can be used to process the mappings.

One more solution could be to combine both approaches. Then well selected extension points would permit to eliminate the need for generated code modifications to a great degree.

3.1 . The platform from the user point of view

The proposed tool definition platform will be metamodel based. At the beginning the domain metamodel of a domain specific language should be built (e.g., by MOLA metamodel editor). The next step would be to define the presentation type model and mappings between the domain metamodel and presentation type model. All this will be done using wizard-style dialogs in the tool development platform.

If built-in mapping possibilities are not suitable for some task, the user will be able to select/create custom MOLA procedure (using the built-in MOLA editor). Appropriate parameters to and from this procedure should be passed, to ensure integrity with the mappings. For each extension point parameters passed to procedures used in this extension point are predefined.

When the tool development is complete, the user can press the button “Build tool”. Thus the tool executable in one step is obtained. Alternatively, if there is such a need the user can edit the generated code and then compile it.

3.2 . Mapping definition

Mappings are based on typical mapping patterns. A large set of mapping patterns has been identified in Generic Modeling Tool [5] and they will be reused in this project.

Mapping definition is based on the mapping and presentation type metamodels as the abstract syntax of the “mapping language”. The visible form of this language will show up as wizard-style dialogs, which will build instances of these metamodels. Appropriate tool support can be built with a small effort using METAclipse platform.

For the mapping metamodel the most important task is a seamless integration of mappings with custom MOLA procedures. The mapping metamodel granularity and structure should be chosen so that each action could be replaced with an appropriate custom MOLA procedure. The transformation based approach permits to use a more detailed mapping granularity than in traditional mapping based tools.

Presentation definition in a graphical tool consists of several parts: property dialogs, diagrams as well as model tree, menus etc. In this paper only a subset from the property dialog part of the presentation and mapping metamodels is briefly sketched [Fig 1.], in order to demonstrate the proposed integration ideas. We assume here that typical Eclipse-style dialogs are used.

When a property dialog for a domain class is to be defined, at first an appropriate property dialog type (i.e., its structure, element types and functionality) is designed, then it is mapped to domain metamodel elements. A property dialog consists of tabs, which can be either a FieldList (for displaying class attributes and linked class instances) or a Grid (for displaying child instance properties in a tabular form). The basic element of both is a Field, whose type definition is the central point in the approach. For each field type it must be defined what must be shown there when the corresponding class instance is selected. For many field kinds (e.g. combobox) the available valid value set (e.g., a set of selected class instances) must be generated and visualized. Finally, it must be defined what has to be done when the value is modified (in Eclipse-style dialogs the model update follows immediately).

As the metamodel fragment [Fig. 1] shows, for all these situations possible typical cases are defined via mappings to domain metamodel elements (e.g., which class attribute must be visualized in a field in the simplest case). The metamodel contains also structuring elements defining various typical ways how these elementary mappings can be combined, e.g., expressions built over elementary mapped values. In all cases the corresponding mapping-based definition can be replaced by a call to a specified custom MOLA procedure, in many situations these calls can be added for pre- or post-processing. One more novel idea is to use MOLA patterns for defining custom instance set filters, e.g., for selection of relevant child instances. This close integration of mappings and procedural approach is a key factor in reaching the goal when the transformations generated from mapping need only be combined with the specified custom MOLA procedures, but require no direct manual modification.

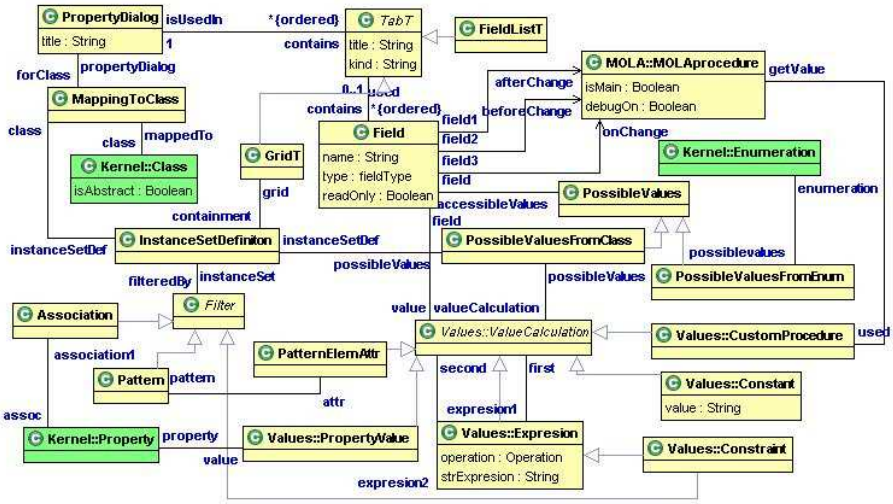

Fig. 1. Mapping and presentation metamodel subset describing property dialogs

The metamodel part for diagram presentation can be built on the same principles, only more classes would be present since it is more complicated.

3.3 . Facilities required to implement the approach

The approach requires a sort of generator generating MOLA transformations from the defined mappings. The most straightforward approach would be to define this generator in MOLA language. However, a more interesting solution requiring less effort to be implemented can be provided in this project.

It is possible to define a “MOLA template language”. This language is a direct generalisation of popular textual template languages (of the kind model-to-text) to graphical languages. Certainly, only the abstract syntax form (model) of MOLA can be easily generated, but this is sufficient for the subsequent compilation. The planned MOLA template language would contain two kinds of MOLA statements: standard ones to be executed during the generation process and those to be “copied” to the generated “code” (in fact, model) with template parameters (in fact, expressions) replaced by the appropriate generation time values. Some interesting solutions could appear here, for example how to generate a set of similar procedures from one template procedure. The template part of the language most probably would require some natural extensions of MOLA syntax, for example, reference to a parameterized class in the MOLA pattern definition.

The first experiments show that generator algorithms in such a template MOLA could be defined quite easily. The implementation of template MOLA itself would also not be very complicated. The existing MOLA editor in METAclipse could be extended for this purpose. In turn, the “pre-processor” converting template MOLA to ordinary MOLA also seems to be not very complicated, after that the existing MOLA

compiler can be used. It should be noted, that the template MOLA has a value of its own as a general purpose macro-processor for MOLA.

Another possible way to implement the transition from mapping definitions to MOLA would be to build a universal interpreter in MOLA which would directly interpret them. Some experiments show that the interpreter would consist of procedures quite similar in form to those used in generator. Certainly, some true extensions to MOLA language and compiler would be required in this case. Also, there would be no possibility for tool builder to modify the "generated" code. However, the total effort for interpreter approach could be less.

4. Conclusions

The overall goal of this research project is to develop the scientific basis required to create a DSL tool development platform with integrated mapping and transformations support. The main target is to develop language and metamodel facilities for this platform. Only an experimental version of the platform is planed to validate the proposed approach and languages.

Currently draft requirements for such a tool development platform have been developed. The first version of mapping definition and generation/interpretation languages has been developed. These languages should be improved and tested on real life examples. Tool support for them should be developed. Detailed architecture of the tool should be developed.

Though there are many open questions, first experiments (redefining some parts of MOLA tool) seem to be very promising.

References

1. Meta-Object Facility (MOF), http://www.omg.org/mof/
2. Graphical Modeling Framework (GMF), http://www.eclipse.org/gmf/
3. MetaEdit+ Method Workbench User's Guide, Version 4.0, http://www.metacase.com/support/40/manuals/mwb40sr2a4.pdf, 2005.
4. S. Cook, G. Jones, S. Kent, A. C. Wills: Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley, 2007.
5. E. Celms, A. Kalnins, L. Lace: Diagram definition facilities based on metamodel mappings. OOPSLA'2003, Workshop on DSM, Anaheim, California, USA, October 2003, pp. 23-32
6. A. Kalnins, O. Vilitis, E. Celms, E. Kalnina, A. Sostaks, J. Barzdins: Building Tools by Model Transformations in Eclipse. Proceedings of DSM'07 workshop of OOPSLA 2007, Montreal, Canada, Jyvaskyla University Printing House, 2007, pp. 194–207.
7. C. Ermel, K. Ehrig, G. Taentzer, E. Weiss: Object Oriented and Rule-based Design of Visual Languages using Tiger. Proceedings of GraBaTs'06, 2006, pp. 12
8. I. Rath, D. Varro. Challenges for advanced domain-specific modeling frameworks. Proc. of Workshop on Domain-Specific Program Development (DSPD), ECOOP 2006, France.
9 J. Barzdins, A. Zarins, K. Cerans, et. al. *GrTP: Transformation Based Graphical Tool Building Platform*, Proc. of Workshop on MDDAUI, MODELS 2007, Nashville, USA.
10. A. Kalnins, J. Barzdins, E. Celms: Model Transformation Language MOLA. Proceedings of MDAFA 2004, Vol. 3599, Springer LNCS, 2005, pp. 62-76
11. UL IMCS, MOLA pages, http://mola.mii.lu.lv/