Model Driven Production of Domain-Specific Modeling Tools

Bassem KOSAYBA, Raphal MARVIE, Jean-Marc GEIB

Laboratoire d'Informatique Fondamentale de Lille
UMR CNRS 8022
59655 Villeneuve d'Ascq
{kosayba,marvie,geib}@lifl.fr

26 July 2004

Abstract

The models built using visual forms which are representations of the domain concepts are easier to be understood and expressed by the people who work in this domain. Many projects produce modeling environments that offer only the domain concepts to the user but with a single graphic view that the user has to be satisfied with. In this paper, we present our process that produce domain-specific modeling tools. This process is independent of the graphic view. So, it can associate for the same domain-specific several graphic views. Also, the way followed by this process makes it very simple to re-use the description of the tool graphic view like that of the domain-specific.

1 Introduction

Most current graphic environments which help designers to express their models are rather general and are aimed at a large number of domains. Thus, they are never really adapted to any particular domain. These environments do not take the specifities of the domains into account. Without adaptation, these environments have an impact on the design of the solution suggested by the user because (s)he must express the concepts of his/her particular domain using the concepts proposed by the general modeling tool. So, it is important to provide the designer a modeling tool specialized to his/her domain. In other words, the environment must be adapted to the user's domain instead of the user having to adapt himself to the environment.

There are many approaches to adapt a modeling environment to the user domain. The majority of these approaches use a generator based approach in order to produce the modeling tool. Unfortunately such generators behavior is

hard coded. Then, in order to change an aspect of the produced tool, such as the graphic representation, the generator has to be modified and in the best cases a graphic representation already taken into account by the generator can be selected.

We are mainly interested by the meta-modeling approaches that recognize and represent visually the domains concepts. The DOME [4] (Domain Modeling Environment) project produces a domain-specific visual environment by a visual re-configuration of the DOME meta-model concepts (Node, Connector). The GME [3] (Generic Modeling Environment) project produces a domain-specific visual environment by two steps. First, one defines the domain-specific model using the GME meta-model concepts and after, one associates visually a graphic form to each concept of the specific-domain model already defined. The two projects provide good graphical modeling environments but these environments have an unique graphic interface. So, the tool user must be satisfied by this graphic interface. In our work, we insist on the separation between the tool graphic representation and the domain-specific knowledge in order to capitalize these two definitions for re-use. In other words, we want to change only the definition of the graphic representation in order to change the graphic interface and without re-defining the domain-specific knowledge. Thus, the modeling tool that we want to produce have two aspects. The first describe the tool graphic view and the second describe the tool functions depending directly on the user specific-domain. The separation between these two aspects allow us to design each aspect alone. In addition, we have to describe in another place how we will integrate the two aspects in order to obtain the final product that is the modeling tool in our case.

In order to implement this approach, we have decided to use the models to describe the tool aspects and the models transformation in order to integrate them togather. This solution makes it possible to re-use the model defining a tool aspect in order to give this same aspect to another tool produced by our process.

In this paper, we present a proposal in order to systematically generate a graphic tool adapted to the user's need, starting from a model expressing the concepts of a specific-domain and a model describing the graphic view. Section 2, presents our proposal inspired from the MDA process. Section 3, shows the models used to realize our proposal and it presents an example to illustrate our approach. Finally, section 4 presents some conclusions and future works.

2 Proposal

We think that the success of visual domain-specific modeling environments depends on their capacity to capture the domain-specific notations and to handle them. So, our framework allows, starting from domain and view description models, to obtain a suitable graphic interface which assists the design in a particular domain. Figure 1 explains our proposition.

This approach makes it possible to produce easily various graphic represen-

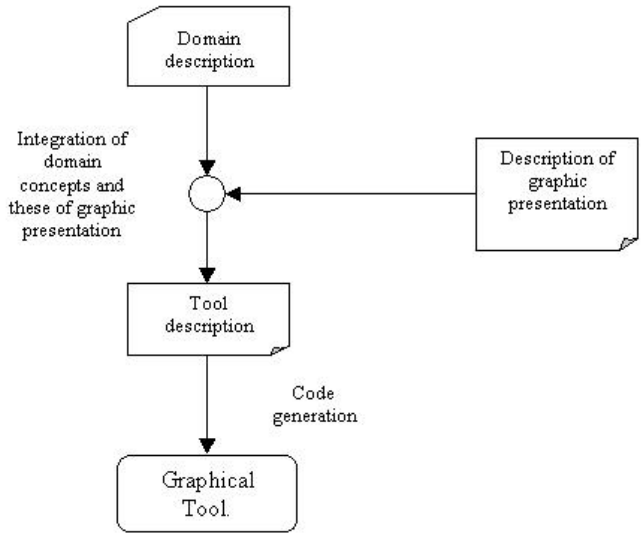

Figure 1: Overview

tations for the same application domain. This variety helps team members to discus the same problem while each one works with his/her favorite graphic representation. The definition of the domain-specific and of the graphic representations are reusable. Defining a modeling tool for a specific-domain becomes choosing a definition of this domain-specific and a definition of a graphic representation. Thus, the production of a tool adapted to the user becomes simpler. It becomes even possible to give various representations for users of the same domain-specific.

In order to provide several graphic views to the graphic modeling tool and without developing several generators (one for each different graphic view), we have developed a framework independently of the graphic view description and based on models transformation. What makes it possible to re-use the domain-specific knowledge description with other graphic view descriptions or the reverse.

The idea of our process is to delay the insertion of the graphic representations in the different parts of the graphic interface when the domain concepts are recognized. A meta-model allows one to express the domain-specific concepts independently of the graphic view (GUI IM). Next, a meta-model describes the parts of the expected graphical representation. This latter allows one to express his/her own graphical representation for the modeling tool (GUI M). Next, a translator transforms the GUI IM (GUI Independent Model) to a GUI SM (GUI Specific Model) that contains the same information of the GUI IM and their relationships with the graphical representation. Finally, a generator maps the

GUI SM into the software components corresponding to the choosed graphical representation.

In fact, this process weaves the domain-specific and the graphical representation models without knowing the model definitions but only how they are defined. After that, it generates the code of the modeling tool according to the model that is the result of this weaving.

3 Implementation

In order to automate as possible the capture of specific-domain notations by visual forms, we propose that the representation of a domain concept is fixed according to the meta-modeling concept used to define this domain concept. For example, each domain's concept defined using the meta-modeling concept *class* is drawn as a rectangle containing the name of this domain concept.

3.1 Production driven by models

Our approach is organized into three levels, The meta-model level, the models transformation level and that of the component library. Figure 2 explains our plan and its entities are detailed.

3.1.1 Our process meta-models

This level defines the rules followed by the tool designer and by the models weaving. There are three meta-models :

The lightMOF is a version reduced of the MOF. It allows one to describe the domain concepts (The domain model) as (*component*, *port*, *container* .etc.) for example, necessary to describe applications models based on components.

The GUI meta-model defines the graphic interface elements and their relations. It allows one to describe his/her own graphical representation of the interface elements.

The merged meta-model defines the relations between each lightMOF concept and the GUI meta-model concepts. It makes it possible to know how this lightMOF concept will be represented in the different parts of the tool graphic interface. The models weaving process follow these relations in order to produce the graphic specific tool model (GUI SM).

3.1.2 The process models transformation

In this level, the models information are inserted in other models in order to enrich the new models or in order to be reorganized. There are three transformations :

Models weaving In this step, we build the tool specific graphic model by weaving the elemnts of the domain model and that of the GUI model according to the the relations between the concepts of the lightMOF and that of the GUI meta-models defined in the merged meta-model.

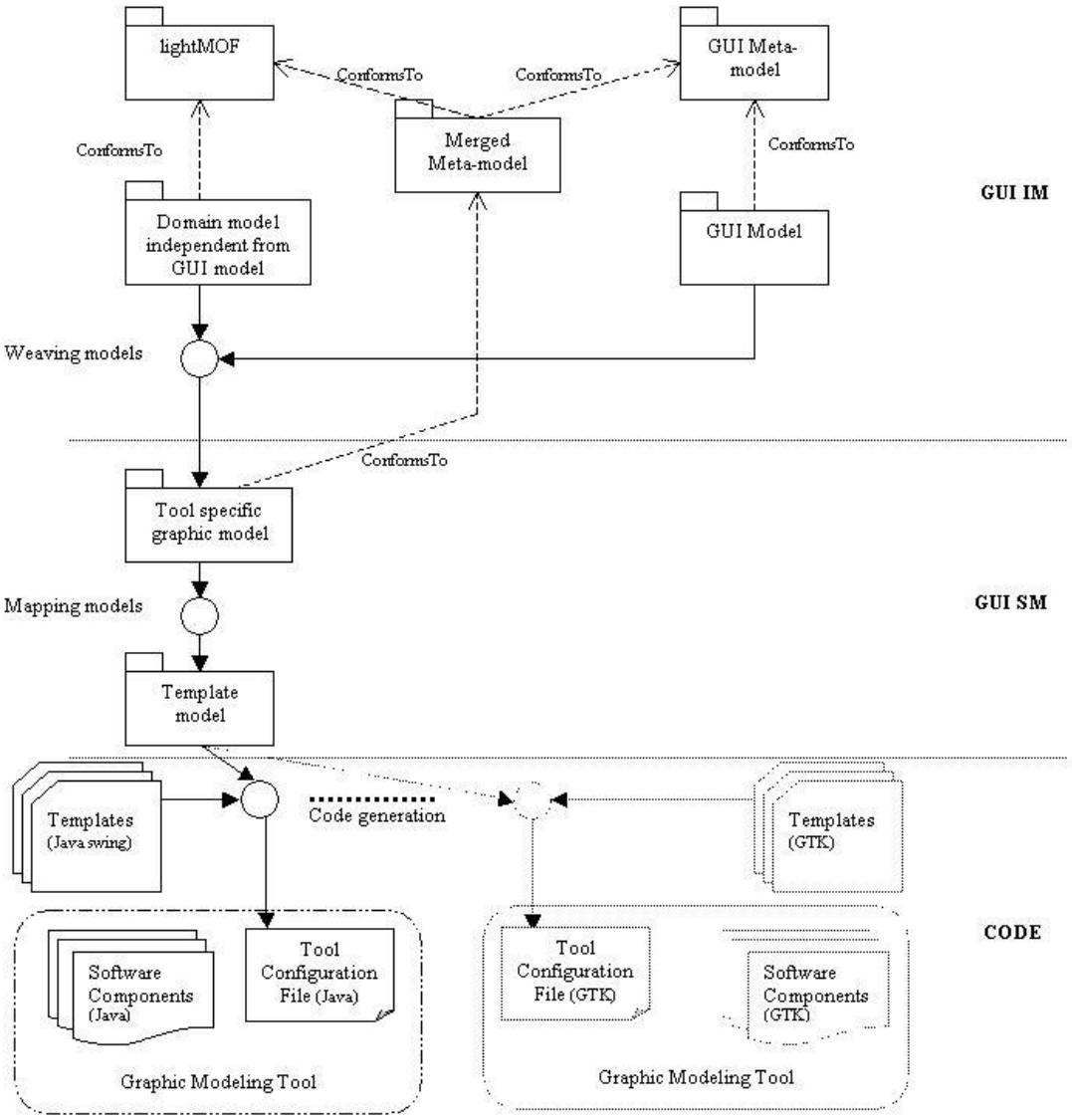

Figure 2: Production process based on the models transformation

Models mapping In this step, we build a template model by extracting its variables from the tool specific graphic model. We use this model to configure the library built-in components. We add this step in order to making this model independent of the technology used to implement the library components. Thus, the template model is an intermediate stage whose the goal is to support several technologies for the library components.

Code generation In this step, we generate the tool configuration file by the configuration of the suitable templates using the template model variables. The role of the file generated is to configure the library components in order to define the model repository and the graphical interface of the modeling tool.

3.1.3 Software component library

This library contains the components that represent the lightMOF concepts and that are used to build a model repository for a domain specific. Moreover, it contains the graphic components that may be choosed in the GUI model.

The architecture of the produced tool will be composed of two parts : the model repository and the graphic interface. This latter must display the definition of the application model and provide the actions allowing the user to handle the application model. In its turn, the model repository use the GUI acts upon the model definition in the repository. This tool architecture guarantees a separation between the model view and the model data. So, we can visualize the same model data using several graphic interfaces generated for its domain-specific and thus we can obtain several model views that discuss the same application model.

3.2 Exemple

In this section, we give examples of our main models in order to illustrate our proposal.

3.2.1 Specific-domain model

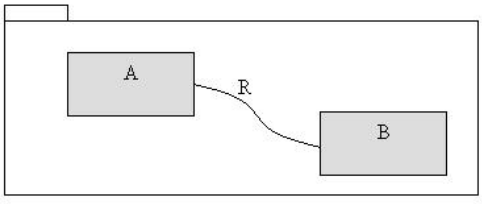

Figure 3: The domain model

We take a simple domain that is composed from two class A and B and an

association R between the two concepts A, B. Figure 3 explains our specific-domain model example.

3.2.2 GUI meta-model

This meta-model states that the graphical interface will be composed of two parts. One to reference the domain concepts and the another to handle the instances of the domain concepts at the application model level. Figure 4 shows this meta-model.

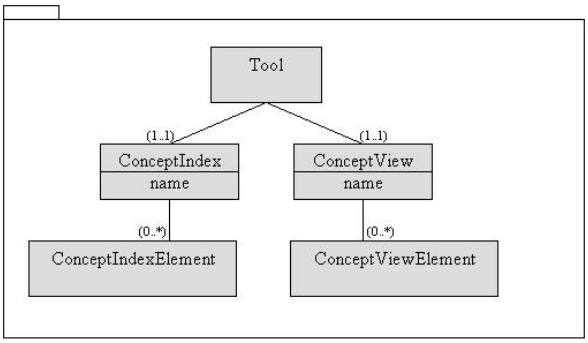

Figure 4: Structure of the graphical interface

3.2.3 Merged meta-model

In this meta-model, we say that the domain-specific concepts defined by light-MOF *class* and *association* concepts will be organized by the *ConceptIndex* and will be handled by the *ConceptView* as it is shown in the Figure 5.

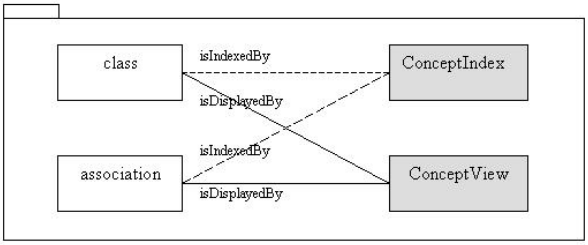

Figure 5: The mapping of the domain concepts into the the graphic interface elements

3.2.4 GUI model

In this model, we define the type of the component used to list domain concepts and that support the user actions in order to add these concept instances in the application model. Therefore, we choose for example, the *conceptIndex* type either *Tree* or *Buttons List*. Moreover, we define the type of the component used to display the concept instance in the application model and to provide the user actions allowing to handle their attributes. So, we choose for example, the *conceptView* type either *Drowing Board* or *Form Board*.

3.2.5 A domain-specific tools

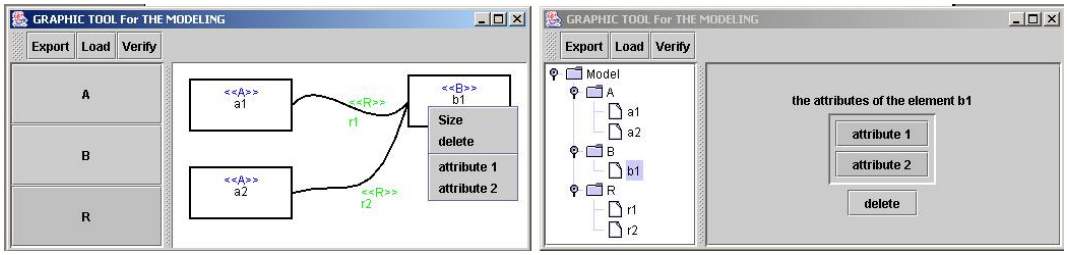

Figure 6: Two graphic views for the same domain

The figure 6 shows two modeling tools produced by our process for the same domain shown in the figure 3 and using two graphic models. The first uses a *Buttons List* to list the domain concepts and a *Drawing Board* to handle the application model elements. While the second use a *Tree* to list the domain concepts and a *Form Board* to handle the application model elements.

4 Conclusion

This paper presents our work which objective is to provide a framework that may be useful for the MDA process. The purpose of this framework is to produce graphical tools allowing the user to express his/her applications models in a domain-specific and independently of execution technologies. At the same time, we tried to benefit from the MDA ideas especially the models re-use, models transformation and the separation of concerns.

The approach followed to implement our proposition may be interesting because it presents a manner to divide a tool into several aspects and allows the re-use of the definitions of these aspects in other tools.

References

[1] R. Esser, W. Janneck *A FrameWork for Defining Domain-Specific Visual Language*. In The OOPSLA Workshop on Domain Specific Visual Languages, 2001.

[2] B. KOSAYBA R. Marvie J-M. GEIB *Production of Domain Oriented Graphic Modeling Environments*, In The MDAFA03 Workshop on Model Driven Architecture Foundations and Applications, 2003.

[3] A. Ledeczi M. Maroti and al *The Generic Modeling Environment*, http://www.isis.vanderbilt.edu/Projects/gme/ .

[4] Honeywell, Inc. *Dome Guide*. Version 5.2.2, 2000.

[5] S.A. White C. Lemus-Olalde *ARCHITECTURAL REUSE IN SOFTWARE DEVELOPEMENT*, .

[6] G. Karsai A. Agarwal and A. Ledeczi *A Metamodel-Driven MDA Process and Tools*, In the UML Workshop W2 Workshop in Software Model Engineering, 2003.

[7] C. Schmidt P. Pfahler U. Kastens C. Fischer *SIMtelligence Designer/J: A Visual language to Specify SIM Toolkit Applications*, .

[8] MetaCase WHITE PAPER *DOMAIN-SPECIFIC MODELING: 10 TIMES FASTER THAN UML*.

[9] Model Driven Architecture (MDA) Guide version 1.0.1, *OMG Document, 2003*, *http://www.omg.org/mda/*.

[10] MOF (*M*eta *O*bject *F*acility *S*pecification), *OMG*, *O*bject *M*anagement *G*roup.

[11] QVT (MOF 2.0 Query, Views and Transformations – Request for Proposal) *OMG*, *Object Management Group*