Model Driven Production of Domain-Specific Modeling Tools

Bassem KOSAYBA, Raphaël MARVIE, Jean-Marc GEIB

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

1 October 2004

Abstract

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 framework for producing domain-specific modeling tools. This framework is independent of the graphic view. So, it can associate for the same domain several graphic views. Without focusing on the modeling graphical interface itself, this article presents a solution to support the generation of such interfaces. In addition, this framework improves the potential re-usability of view and domain descriptions.

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-purpose 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 existing approaches to adapt a modeling environment to the user domain. Most of these approaches rely on a generator based solution in order to produce modeling tools. Unfortunately, such generators' behavior is hard coded. Then, in order to change an aspect of the produced tool, such as the graphic appearance, the generator has to be modified, and in best cases a graphic representation already taken into account by the generator can be selected.

We are mainly interested by the meta-modeling approach that identify and represent visually the domains concepts. The DOME [4] (Domain Modeling Environment) project produces a domain-specific visual environment through 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 in two steps. First, one defines the domain-specific model using the GME meta-model concepts. second, one associates visually a graphic form to each concept of the specific-domain model defined.

The two projects provide good graphical modeling environments but these environments have a 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.

The modeling tool 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 allows us to design each aspect alone. In addition, we have to describe in another place how we integrate the two aspects in order to obtain the final product that is the modeling tool.

In order to implement this approach, we have decided to use the models to describe the tool appearance and the models transformation [11] in order to integrate the tool appearance with other tool characteristics. This solution makes it possible to re-use the model defining a tool appearance in order to give this same appearance to another tool produced by our process. Then, the tool appearance is one characteristic of the tool.

This paper presents our proposal 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 [9]. 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 ability to capture the domain specific notations and to handle them. So, our framework allows, starting from a domain and a view description models, to obtain a suitable graphic interface which assists the design in this particular domain. Figure 6 outlines our approach.

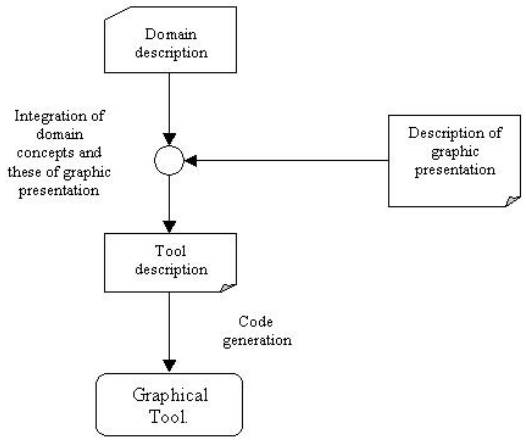

Figure 1: Overview

This approach makes it possible to produce easily various graphic representations for the same application domain and to use the same graphic representation for various domain. The definition of the domain and of the graphic representations are reusable. Defining a modeling tool for a particular domain is equivalent to selecting a definition of this domain and a definition of a graphic representation. Thus, the production of a tool adapted to the user becomes simpler. It becomes even possible to provide various representations of the same domain.

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

The idea of our approach 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 permits the description of the expected graphical representation. This latter allows one to express his/her own graphical representation for the modeling tool (GUI M). Then, a translator transforms the GUI IM (GUI Independent Model) to a GUI SM (GUI Specific Model) that contains the same pieces of information as the GUI IM and their relationships with the graphical representation items that have been selected. Finally, a generator maps the GUI SM into the software components corresponding to the chosen graphical representation.

In fact, this process weaves the domain 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 much as possible the expression of 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. Not focusing on the choice of a box for a class, it is the way we are achieving this weaving that is central in our work.

3.1 Production driven by models

Our approach is organized into three levels : the meta-model level, the models transformation level, and of the component library. Figure 2 explains our organization 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 MOF [10] allows one to specify the domain concepts necessary to describe applications models. These domain concepts definitions represent the meta-model for designer to express their system models in the particular domain. For example, in the context of component based applications, the main concepts are *component*, *port*, *container* .etc. We have chose the MOF first as it is a good (among other) means for meta-modeling, and second as it is a standard (even if this experiments does not make use of all the MOF concepts).

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 relation meta-model defines the relationships between MOF concepts and the GUI meta-model concepts. It makes it possible to know how a MOF concept will be represented in the different parts of the tool graphic interface. The model weaving process follow these relationships in order to produce the graphic specific tool model (GUI SM).

3.1.2 The process models transformation

At this level, there are three transformations :

Model weaving In this step, we build the domain graphic specific model by weaving the elements of the domain model and that of the GUI model according to the the relations between the concepts of the MOF and those of the GUI meta-model. These relationships are defined in the *relation meta-model*.

Model mapping In this step, we build a template-based model by extracting its variable values from the domain graphic specific model. We use this model to configure the components library. This step permits the tool configuration model to be independent of

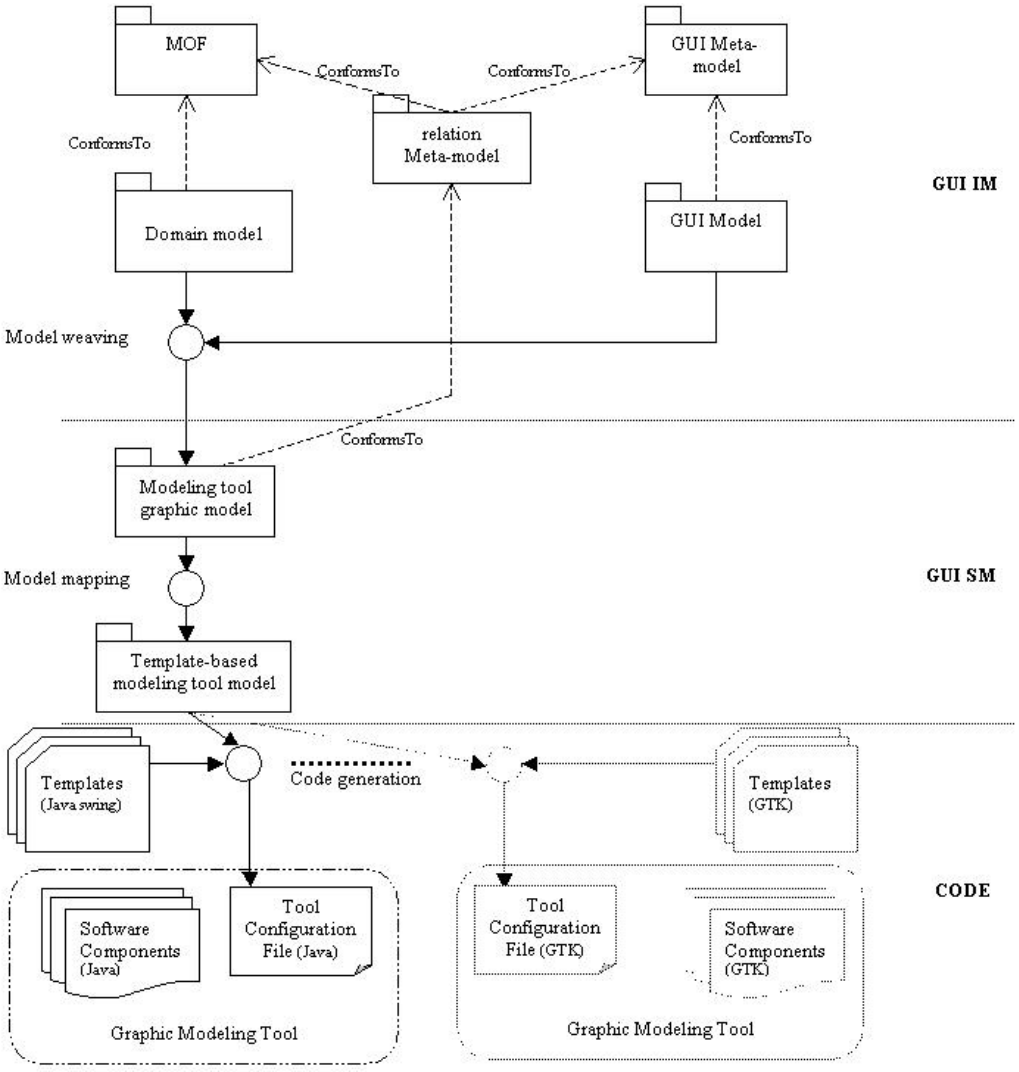

Figure 2: Production processbased on the models transformation

any technology used for implementing the library component. Thus, the template model is an intermediate stage which goal is to support several technologies for the components library implementation.

Code generation In this step, we generate the tool configuration file for a particular technology using the template-based model. The goal 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

The tool architecture is made of two distinct parts: a graphical user interface and a model repository. Then, the software component library contains two categories of components: graphical components and components representing MOF concepts.

The domain concepts components represent a logical profile allowing the tool user to build an application model. For defining such components, we configure built-in components. For each MOF concept, a built-in configurable component is defined. A model repository dedicated to a specific domain defined using MOF concepts, is generated after configuration of the appropriate built-in components.

Graphical components are implementations of the concepts of the GUI meta-model. These components respect some interfaces that define their main functions and their interactions with the other graphical components and the model repository. In order to add another graphical representation to a GUI meta-model element, these interfaces need to be implemented.

The architecture of the produced tool is composed of two parts : the model repository and the graphical user interface. The latter must display the definition of the application model and provide the actions allowing the user to handle application models. The graphical user interface acts upon the model definition in the repository. This tool architecture guarantees a separation between the model view and the model data. Therefore, we can visualize the same model data using several graphical user interfaces generated for this domain.

3.2 Example

In this section, we give an example of our approach applied to a simple domain in order to illustrate our proposal. First we present our meta-models. Then an example of domain and graphical view definition.

3.2.1 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 3 shows this meta-model.

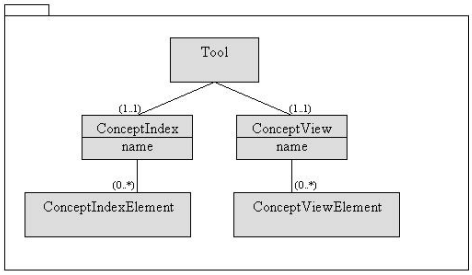

Figure 3: Structure of the graphical interface

3.2.2 Relation meta-model

In this meta-model, we say that the domain-specific concepts defined by 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 4.

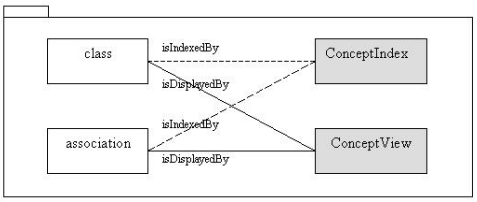

Figure 4: The mapping of the domain concepts into the the graphic interface elements

3.2.3 domain definition

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 5 explains our specific-domain model example.

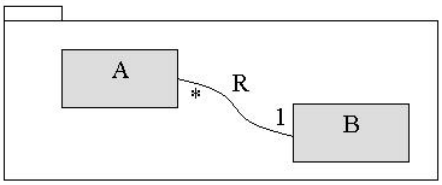

Figure 5: The domain model

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 *Drawing Board* or *Form Board*.

3.2.5 A domain-specific tools

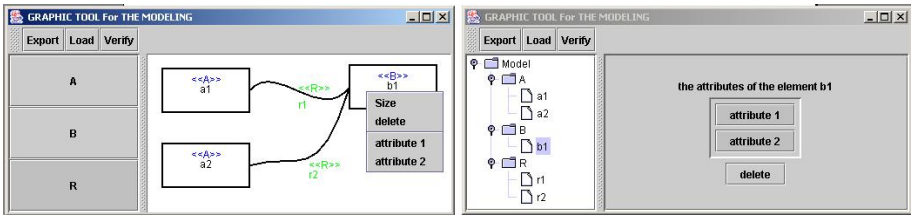

Figure 6: Two graphic views for the same domain

The figure 6 shows two modeling tools produced by our process for the same domain shown in the figure 5 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 has presented our framework for producing graphical modeling environments. The two specificities of this framework is first its ability to produce modeling support dedicated to domains, as well as to user's taste, and second to be based on a model driven approach. The use of a model driven approach introduces two related benefits : Domain and graphical appearance definitions—using meta-models—are capitalized, and building a graphical modeling environment is reduced to choosing the proper domain and appearance definitions, or defining new ones that will be re-usable. This has been achieved through the definition of a model driven process composed of several model transformations and code generation steps.

The work presented in this paper is mainly 'technical', being the result of a tool providing objective. Our framework is intended at allowing tool providers to deliver the best suited graphical modeling environment for designers in a particular domain. Future trends of this work are both to study the cooperation of designer using multiple views on a

model like we started in [12] and to study how to define a graphical appearance definition according to users (designers) wishes.

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 Development*, Proceedings of 20th International Computers in Engineering Symposium (ASME-ETCE98), January 1998,

[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.*

[12] R. Marvie *(Separation of Concerns and Metamodeling applied to Software Architecture Handling)*, PhD thesis, LIFL, December, 2002.

[13] S.A. White C. Lemus-Olalde *Architecture Reuse through a Domain Specific Language Generator*, Eighth Annual Workshop on Software Reuse, March 1997,