Operators for Template-Based MDE

Matthieu Allon[1]

University of Lille - CRIStAL Lab. (UMR CNRS 9189)
France, Villeneuve d'Ascq
Matthieu.Allon@etudiant.univ-lille1.fr

Abstract

In MDE, design of systems can be improved and accelerated thanks to reusable models which are made available in model repositories or libraries. This paper focuses on the construction and exploitation of "off-the-shelf" model template bases. Model templates are parameterized models which are adaptable to various application contexts. Due to their parameterization, model templates have their own modeling space. In this paper, we present the main modeling activities that underlie this space, their dedicated engineering processes and their actors, and we contribute to the model reuse improvement by detailing new template operators for the described modeling activities. A software environment is shown to illustrate template based engineering in Eclipse.

Keywords: Model Templates, Model Reuse, Model Space, Template Engineering.

1 Introduction

In MDE, model reuse is a big challenge that aims to facilitate the capitalization of design efforts and logics ("off-the-shelf" model component libraries [12]), then to accelerate system design and improve their quality. In existing work, the main approaches are either based on reuse of composable model pieces, or on parameterized models which are adaptable to various application contexts [2,7,11]. We contributed to this research by studying model parameterization techniques such as the one offered by the UML "Template" construct. Templates differ from composable model pieces due to template parameterization which clearly identifies what is required for a reusable model: parameterization acts as an interface specifying what is required [14]. We have defined two approaches that allow the design of systems by assembling templates through parameterization [3,14,15].

Starting from these works, we focus now on the construction and exploitation of model template bases and the related engineering processes. Our objective is to increase the capacities for creating, composing and reuse templates within such bases by proposing new template operators. Resulting model spaces must be systematically characterized to master and exploit their structuring properties. After a reminder on model templates (Section 2), we present our vision of model template spaces (Section 3) and the related operators (Section 4). Then, we describe a software environment in Eclipse to construct and exploit model template spaces in UML (Section 5).

2 Aspectual Templates in UML

UML Templates [1] allow to capture modeling constructs which expose some of their constituents as parameters. Such constructs can be classes or packages (but not only). To specify its parameterization, a template owns a signature, which is a list of formal parameters where each one designates an element that is part of the templated model. It is the intent of templates to be reused. For template application, the standard defines a specific "template binding" relationship which allows to specify how the content of a base model is derived from a template through the substitution of its parameters.

In UML, template parameters form an unstructured set of model elements so that the construct is general and permissive enough to render much of model parameterization needs such as the modeling of generic classes (such as C++ templates) [16], the capture of Design Patterns [17], View [10] or Aspect Oriented Modeling [13]. In [18], we proposed a compatible enhancement of UML templates (*Aspectual Templates*). It consists in enforcing templates to have a full model as parameter (*parameter model*) to improve their consistency, notably for aspectual usages, but also to better specify the model of systems to which the functionalities will apply. Following this, the binding mechanism has been adapted to enable substitution of the model parameter by a conforming substructure of the base model.

Figure 1 gives an example of such an enhanced template: the observer pattern template is applied to a base model (the *CarHiringSystem* application context) for installing functionalities between an agency and a client for observing car availability. Each substitution in the binding follows the same mechanism as the one described for the *Subject* and *Agency* bound elements: *Subject* is bound to *Agency*, so all constituents of *Subject* are injected in *Agency*.

One can observe that (1) the template parameters (see the superimposed dashed box) form the *parameter model* and (2) its structure is well-preserved by the substituted elements in the binding.

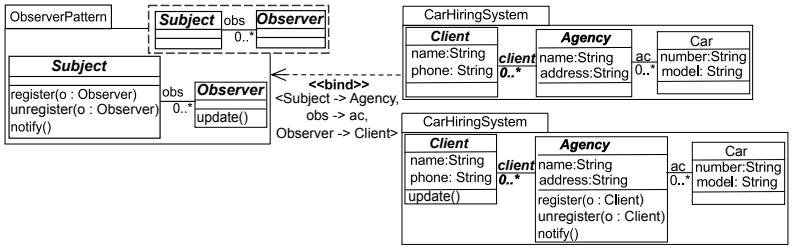

Fig. 1: Template application

The previous template construct and application mechanism allow to design complex systems from assemblies of templates but also to obtain richer templates from existing ones.

3 Template Based Model Engineering

On the basis of the previous model template technique, specific modeling spaces with their engineering practices and automatic processes emerge. Fig.2

shows an illustration of such a modeling space with involved actors and activities around a model repository containing templates and models that they share.

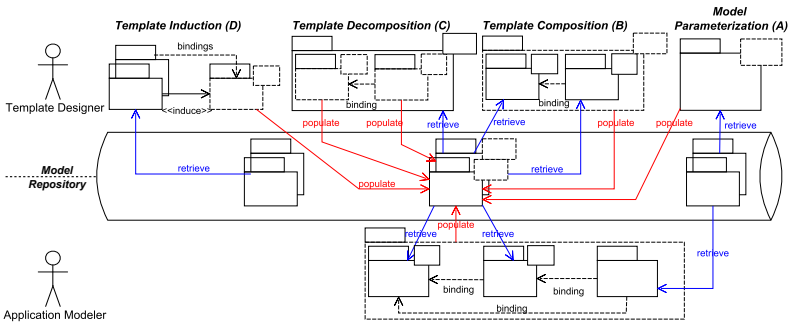

Fig. 2: Template Based Model Engineering

In this modeling space, designers of model templates are mainly concerned with "design for reuse" and the constitution of libraries. They have to identify candidate models of functionnalities and render them as model templates for enabling their reuse. Several activities (see Fig.2) are of interest for this engineering task :

- New template creation by the parameterization of unparameterized models from the selection of model constituents (activity (A)).
- Composition of existing templates in order to build richer ones, by template merging or template-to-template binding (activity (B)).
- Decomposition of a previously identified complex template which leads to identify finer ones (activity (C)).
- Induction of new templates from previously designed models of systems which share common functionalities (activity (D)).

Application modelers are much concerned with "design by reuse" methodology (right of Fig.2). From template designers, they get the possibility to exploit model templates for their application needs in a safe manner through "template binding". This latter allows complex system construction by successive application of templates.

All these practices must be controlled in an homogeneous and consistent manner. This requires a precise formalization and characterization of model templates as well as their relationships and dedicated operators.

4 Aspectual Template Operators

To support activities described previously, we can think of many useful template operators. To our knowledge, there are approaches using templates [2, 8, 9] and other offer model operator [4, 5, 7]. However, neither approach offers operators on templates, i.e. operators considering model parameterization. Table 1 gives the studied operators, with their functional signature (operands and result), their meaning description and their related activity.

In our previous work, we deeply studied the operators for applying templates through substitutions of their parameters that are the *apply* [14] and the *instan-*

Table 1: Studied Operators

Operators	Signature	Explanation	Modeling Activities
Promote	$AT \times Core\ constituents \rightarrow AT$	New template with template constituents promoted to the parameter model.	Parameterization
Restrict	$AT \times Parameter\ constituents \rightarrow AT$	New template with parameters which are restricted to not parameterized template constituents.	
Apply	$AT \times AT \times Substitution\ Set \rightarrow AT$	New template by injecting template constituents in a second template.	Composition
Instantiate		New template by replacing template constituents by selected other constituents in a second template.	
Merge	$AT \times AT \rightarrow AT$	New template by merging two templates.	
Extract	$AT \times Core\ constituents \rightarrow AT$	New template by extracting constituents from a template.	Decomposition

tiate [3] operators, while the *merge* operator corresponds to the UML one [1]. We are currently studying the remaining ones.

Fig.3 illustrates the *promote* operator that can be used for adapting aspectual template parameters for new application contexts. This is done by giving a parameter status to template constituents. The *restrict* operator performs dually: it allows to remove model elements in the parameter model.

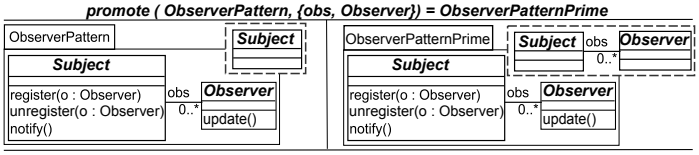

Fig. 3: Promote Operator

Both previous operators are useful to allow a template designer to create various versions of a template, i.e. templates with the same constituents, but with different parameter models. This is interesting for applications designers to compose a template [1] with a model which does not need a binding with all parameter constituents. Concerning the third new operator, *extract*, it allows a template designer to obtain a part of template, which can be useful to extract, for instance, design patterns from others (e.g. a singleton from a factory).

5 EMF technology

We are implementing a software environment dedicated to template based model engineering in Eclipse, with plugins based on the official EMF (Eclipse Modeling Framework), UML and OCL plugins. They offer core functionalities to specify and verify templates well-formedness and their binding in a compliant way with UML thanks to a specific profile. In addition, these plugins provide original facilities to support other modeling tasks targeting templates or user assistance (e.g. signatures and bindings inference and automatic completion). All these functionalities are reusable for modeling tools handling model templates.

[1] For instance, representing a design pattern.

In our case, they are experimented through an in-progress case study[2] in a CASE tool prototype[3] (see Fig. 4).

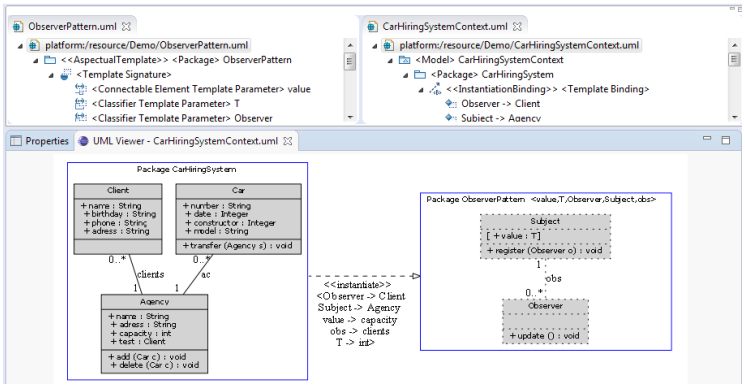

Fig. 4: Tooling - Instantiate operator example

Following main features are currently under study to support plain template based engineering:

- An engine for determining inclusion and typing relationships between templates and their constituents. Such relationships should permit template hierarchical structuring, useful to template-based activities. We are developing this by exploiting our generic submodel engine [6].
- A richer set of template operators. Currently, only parameterization, merging, instantiation and aspectual binding with their checking, completion and inference facilities are available. Other operators are under study.
- Template searching capacities in model repositories. Using the relationships above, a searching system similar to the one described in [19] but specific to templates can be developed. From a selected template, this system will enable to find similar templates according to its constituents into repositories and hierarchically present them.

6 Conclusion

In this paper, starting from our previous work, we sketch the model template modeling space and its associated engineering. This work presents new operators, a general method for developing and using the templates, and describes a work-in-progress tool to support the management and use of templates.

Defining the envisioned set of operators raises a large number of issues, and first results with the in-progress case study shown that there are still many open questions: the operator semantics and usages, their algebraic properties and composition. However, more Fundamental issues need to be investigated to the help of full template-based modeling and engineering, especially the questions of model inclusion and model typing within the same meta-modeling space [6].

[2]The case study concerns the modeling of a REST News Server using design patterns.

[3]`http://www.cristal.univ-lille.fr/caramel/MBE_Template/`

This work will contribute to better theoretical understanding and generalization of templates for the quest of model reuse and model space structuring.

References

1. UML 2.4.1 Superstructure Specification, 2011. http://www.omg.org/spec/UML/2.4.1/.
2. O. Alam, J. Kienzle, and G. Mussbacher. Concern-oriented software design. In *International Conference on Model Driven Engineering Languages and Systems*, pages 604–621. Springer, 2013.
3. M. Allon, G. Vanwormhoudt, B. Carré, and O. Caron. Isolating and Reusing Template Instances in UML. In *12th European Conference on Modelling Foundations and Applications*, Vienna, Austria, 2016.
4. P.A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *CIDR*, volume 2003, pages 209–220, 2003.
5. G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh. A manifesto for model merging. 2006.
6. B. Carré, G. Vanwormhoudt, and O. Caron. From subsets of model elements to submodels: A characterization of submodels and their properties. *Software & Systems Modeling*, 2013.
7. S. Clarke. Extending standard UML with model composition semantics. 2002.
8. S. Clarke and R.J. Walker. Generic aspect-oriented design with theme/UML. *Aspect-oriented software development*, pages 425–458, 2005.
9. J. de Lara and E. Guerra. From types to type requirements: genericity for model-driven engineering. *Software & Systems Modeling*, pages 453–474, 2013.
10. D. D'Souza and A.C. Wills. *Catalysis: Objects, Components, and Frameworks with UML*. Object Technology Series. Addison-Wesley, 1998.
11. D. Del Fabro and J. Bézivin. Generic model management: from theory to practice. In *First International Workshop on Towers of Models - TOWERS 2007*, pages 1–9, 2007.
12. M. Herrmannsdörfer and B. Hummel. Library concepts for model reuse. *Electronic Notes in Theoretical Computer Science*, pages 121–134, 2010.
13. J. Kienzle, W. Al Abed, F. Fleurey, J.M. Jézéquel, and J. Klein. Aspect-oriented design with reusable aspect models. In *Transactions on Aspect-Oriented Software Development*, volume VII, pages 272–320. Springer, 2010.
14. A. Muller. *Construction de systèmes par application de modèles paramétrés*. PhD thesis, University of Lille 1, 2006.
15. A. Muller, O. Caron, B. Carré, and G. Vanwormhoudt. On some properties of parameterized model application. In *Model Driven Architecture–Foundations and Applications*, pages 130–144. Springer, 2005.
16. C. Pérez and J. Bigot. Increasing Reuse in Component Models through Genericity. *Formal Foundations of Reuse and Domain Engineering Lecture Notes in Computer Science*, 5791:21–30, 2009.
17. G. Sunyé, A. Le Guennec, and J-M. Jézéquel. Design Patterns Application in UML. In *Proceedings of 14^{th} European Conference on Object-Oriented Programming (ECOOP'2001)*, pages 44–62. Springer, 2000.
18. G. Vanwormhoudt, O. Caron, and B. Carré. Aspectual templates in UML: Enhancing the semantics of UML templates in OCL. *Software & Systems Modeling*, April 2015.
19. G. Vanwormhoudt, B. Carré, O. Caron, and C. Tombelle. Recherche de sous-modèles. In *CIEL 2014, Troisième Conférence en IngénieriE du Logiciel*, pages 126–130. HAL, 2014.