Isolating and Reusing Template Instances in UML

Matthieu Allon, Gilles Vanwormhoudt, Bernard Carré, Olivier Caron

Manuscript.

The final publication is available at Springer via:
http://dx.doi.org/10.1007/978-3-319-42061-5_11

To cite this paper:
Allon M., Vanwormhoudt G., Carré B., Caron O. (2016) Isolating and Reusing Template Instances in UML. In: Wasowski A., Lönn H. (eds) Modelling Foundations and Applications. ECMFA 2016. Lecture Notes in Computer Science, vol 9764, pp. 173–187. Springer.

Isolating and Reusing Template Instances in UML

Matthieu Allon[1], Gilles Vanwormhoudt[1,2], Bernard Carré[1], and Olivier Caron[1]

[1] University of Lille, CRIStAL Lab. (UMR CNRS 9189), Villeneuve-d'Ascq, France
[2] Mines-Telecom Institute, Villeneuve-d'Ascq, France
Matthieu.Allon@etudiant.univ-lille1.fr, {Gilles.Vanwormhoudt, Bernard.Carre, Olivier.Caron}@univ-lille1.fr

Abstract. In MBE, design of systems can be improved and accelerated thanks to reusable models which are made available in model repositories or libraries. One answer for designing reusable models is parameterization as offered by UML templates and their binding relationship. The standard aims at embracing under the same constructs two distinct kinds of template usages, namely template instantiation and aspectual binding. Template instantiation is concerned with the capacity of UML templates to model generic components (like C++ templates or Java generics) and produce new models from their binding. Aspectual binding is much more concerned with the capacity of UML templates to specify functionalities to inject into models of systems (contexts) which must conform to a required parameter model. In this paper, we focus on the generative interpretation of UML template binding. On the basis of a deep analysis of the standard, it will be shown that template binding consists in template instantiation plus context merging. This allows to isolate the capacity of instantiating templates (under their generative view) to get reusable models coming from applicative contexts. Then the possibility of partial instantiation inspired by partial binding as promoted by the standard is studied. At a practical level, related functionalities are offered within Eclipse.

Keywords UML templates - Aspectual Templates - Template Binding - Partial Binding - Template Instantiation

1 Introduction

In Model-based Engineering, model reuse is a big challenge that aims to facilitate the capitalization of technology independent design efforts and logics ("off-the-shelves" model components libraries [8]) then to accelerate system design and improve their quality via early checking, by the reuse of proved models. Besides the composition of model pieces [10, 16], another way to face this challenge is model parameterization [3, 9], that is the capacity for a model to expose some of its elements as parameters, then produce other models through parameter substitution. This allows to capitalize models of a higher kind which capture

recurrent structure, so that they can be applied (reused) in multiple modeling contexts.

The UML standard answer to this need is the "template construct" and its binding relationship. This construct is general enough to support MBE reuse practices ranging from the representation at a model level of generic software components (such as C++ templates or Java generics) to the weaving of reusable functionalities into models, mainly the way aspect-, pattern- and view-oriented modeling do. In our prior work [18], we contributed to this research by enhancing the semantics of UML Templates for their aspectual interpretation. This leads to so called "aspectual templates" whose parameters must form a model of systems in which to inject the functionalities. This semantics enhancement ensures the "parameters as a model" requirement and its consistency throughout substitution and composition mechanisms, notably when binding is partial. Thanks to this enhancement, UML templates can be better controlled for their aspectual usage, particularly in case of complex assembly.

In this paper, we concentrate on template instantiation which underlies MBE practices related to their generative usage, that is the creation of new models from their generic modeling structure. This calls for the isolation of template instantiation from standard binding. Given this, consequences on template parameters, particularly when they form a model, need to be examined and we do so by the proper identification of template constituents as submodels. It will be shown that instantiation can be applied on any templates being aspectual or not. Similarly to partial binding, partial instantiation is provided when not all parameters are substituted. A study of this feature is offered and its interest for producing models with pieces from multiple contexts is presented. More generally, the isolation of template instances as stated here contributes to increase UML templates reusability and to enrich template-based MDE facilities [1].

The rest of the paper is structured as follows. After providing background on UML templates, we present an analysis of template binding in Section 3. This analysis will lead to the isolation of template instantiation. Then, Section 4 examines how instantiation relates to template parameters. Partial instantiation is studied in Section 5. Section 6 describes application of the results in modeling tools. Before concluding with perspectives, Section 7 discusses template instantiation in existing works.

2 Background on UML Templates

In this section, a reminder on UML templates and their aspectual enhancement [18] are presented to ground the study.

2.1 UML Template and the Binding Relationship

In UML [13], a template is a model which exposes some of its model elements as formal parameters using a signature (list of formal parameters). Examples are class or package templates. Graphically, the signature is contained in a small

dashed rectangle on the top right-hand corner of the template symbol. Templates can be applied, and thus reused to produce other models thanks to parameter substitution, through the standard *binding relationship*[3]. It links a *bound model* to a template (from which it was obtained) via a parameter substitution set that associates *formal parameters of the template* to *actual elements of the bound* model. Constraints of the standard only impose that the type of each actual model element must be a subtype of the corresponding formal parameter.

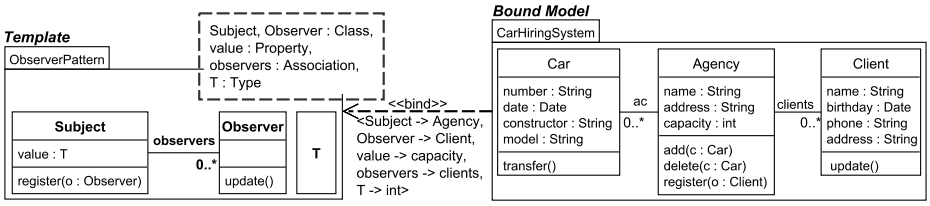

Fig. 1: UML package template example

Figure 1 shows an example of UML template and its binding for extending the model of a system. It shows a package template used to model the *observer* pattern parameterized by the *Subject* and *Observer* classes, the *value* property, the *T* type and the *observers* association. The system where the pattern must be applied represents a car rental agency with its clients and cars. In the figure, the *ObserverPattern* template is used to introduce the observer functionality between *Agency* and *Clients* for capacity observation by clients. This design choice is specified by the binding relationship between the *CarHiringSystem* model and the *ObserverPattern* template with the specified set of substitutions. As a result of the binding, *CarHiringSystem* includes the model structure of the *ObserverPattern* with respect to substitutions.

Finally, UML allows partial binding. Partial binding occurs when not all formal template parameters are substituted. For that, the UML specification states that the unsubstituted formal template parameters are formal template parameters of the bound element, which is itself a template as a consequence. Partial binding will be specifically studied in Section 5.

2.2 Parameters as a Model

Regarding template parameters, the standard does not require any structuring between them. The only constraint imposed by UML is the inclusion of the set of parameters into the set of template elements. Although this choice is permissive, it is underspecified to capture structuring constraints expected from candidate models to correctly apply the template functionality.

Figure 2 illustrates the issue. On the left of the figure, a variant of the Observer template compliant with UML is presented. As expected, all the parameters are model elements of the template core but one can observe that they do

[3] Informally specified in [13], page 650.

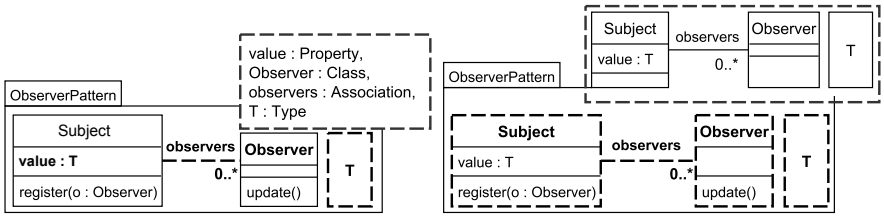

Fig. 2: Set of parameters vs model of parameters

not form a consistent model. Indeed, the *value* property is exposed without its owning class whereas the latter is required to enable its mapping with a property contained in a context class. Similarly, the *observers* association exposed as parameter is underspecified because one of its ends (the *Subject* class) is not declared as a parameter.

In our previous work [18], we deeply studied the aspectual interpretation of UML templates with the requirement that parameters have to form a well formed modeling structure to which candidate models must have to conform. Following this requirement, we stated a semantical enhancement of UML templates which consists in enforcing templates to have a full model as parameter. Its aim is to improve the consistency of templates, notably for aspectual usages, but also to better specify the model of systems to which the functionalities will apply. Following this enhancement, the (partial) binding mechanism has been adapted to enable substitution of the model parameter by a conforming substructure of the base model.

Right of Figure 2 gives the enhanced version of the Observer template. One can observe that the template parameters (see the superimposed dashed box) form a full model. One can also verify that the structure of this model parameter is well preserved by the substituted elements of the binding in Figure 1.

3 Towards Explicit Template Instantiation

In this section, we analyze template binding. We will see that template binding underlies template instantiation. This analysis will serve to motivate our proposal which consists in isolating template instantiation separately from template binding. Constraints that template instantiation imposes on the parameters and the specific case of partial template instantiation will be studied in the next sections.

In UML, the semantics of the binding relationship is specified as follows:

> "The presence of a TemplateBinding relationship implies the same semantics as if the contents of the template owning the target template signature were copied into the bound element, substituting any elements exposed as a formal template parameter by the corresponding elements specified as actual parameters in this binding."([13], page 650)

Following this semantics, the bound model resulting from a template binding can be seen as the merging of an applicative context with the content of the template after substitutions were made. Figure 3 presents this construction principle on the scenario presented in Figure 1. It makes explicit (upper-right in the figure) the applicative context (car agency) to which both the template and the intermediate model apply, instance of the template (*ObserverPatternInstance* upper-left). It is the context that provides actual elements for the binding.

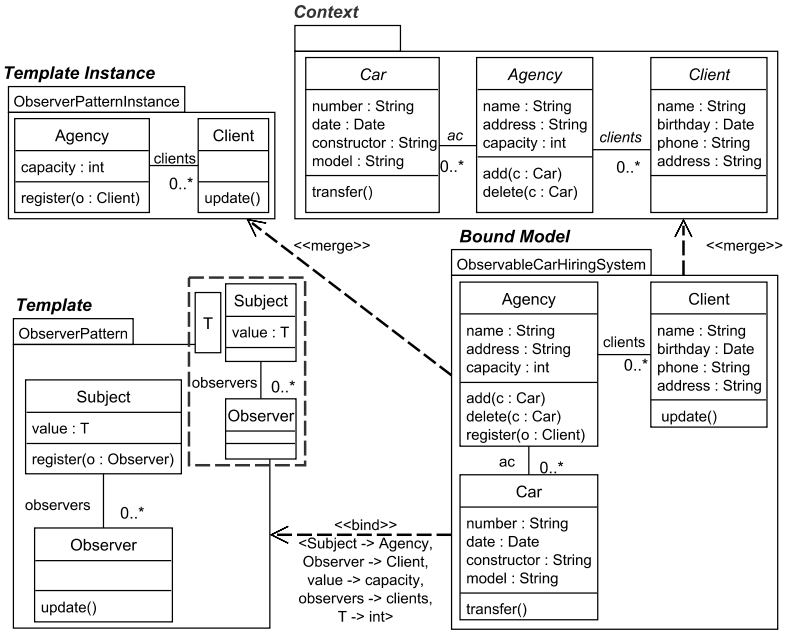

Fig. 3: Template binding = instantiation + context merging

As represented in Figure 3, the bound model (*ObservableCarHiringSystem*) contains all the content of the template instance plus the content of the context. This can be captured using the standard "merge" relationship from the bound model to the template instance and the context:

$$template\ binding = instantiation + context\ merging$$

Following this, once template instances have been isolated, they can be promoted as valuable artifacts of their own and then be reused. This new capacity is of interest when designers are much more concerned by the construction of new models from templates instead of enriching existing ones. This calls for giving a much more active role to template instantiation in the modeling space and its related processes.

As a consequence, we propose to isolate template instantiation from template binding. For representing template instantiation, we use a relationship named

instantiate. Like the binding relationship, this relation requires a template, a source modeling context plus a set of parameter substitutions. Its semantics consists in copying the content of the template and replacing the parameters by corresponding copies of actual elements from the source model.

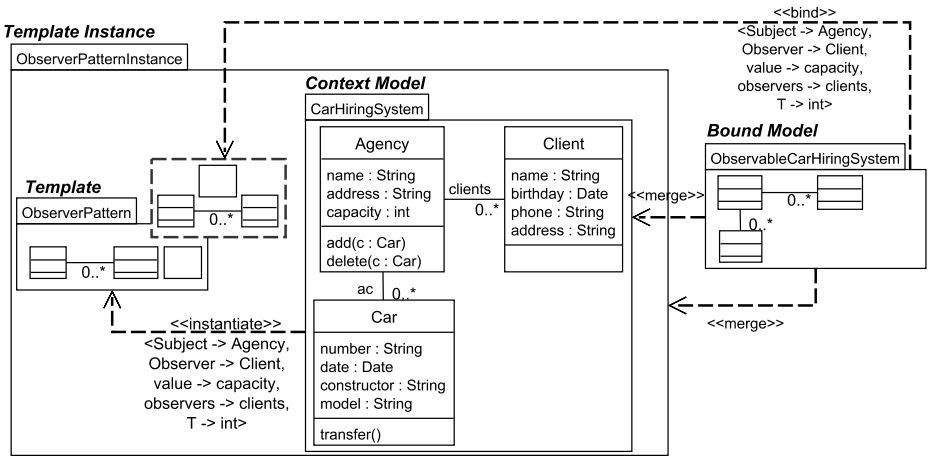

Fig. 4: Template instantiation

Figure 4 illustrates template instantiation with the same template and a source model equal to the context used in Figure 3. The result of this instantiation is the template instance (*ObserverPatternInstance*) presented in Figure 3. Regarding substitution specified in the instantiate relationship, substituted elements from the context model must conform to the modeling structure formed by the parameters. In Figure 4, one can verify that it is actually the case.

Finally, to isolate the instantiate relationship, we showed the treatment of template with a well formed parameter model. More generally, how instantiation relates to the structure of templates needs to be deeply examined. It is the intent of the next section.

4 Instantiation Regarding Kinds of Templates

To study how instantiation relates to template structure, we decompose a template into two complementary constituents: its *parameter* and its *specific submodels*. Main questions are: which constituent provides the template structure and what are the requirement on template parameters when instantiation is considered ?

Consider a template with a well formed parameter submodel. It is this submodel which provides the modeling structure of the template core. This is illustrated in Figure 5. In this situation, the lack of well-formedness for the specific model is due to the fact that classes owning the *register* and *update* methods are parameters, so are not part of the specific model.

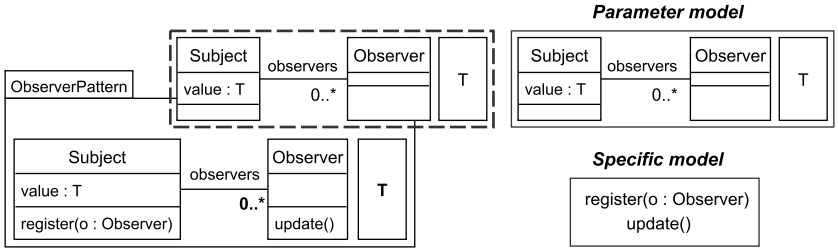

Fig. 5: Template model = parameter + specific submodels

The observation regarding the well-formedness of template constituents raises the question of alternative cases. It can be questioned whether a parameter submodel that is not well formed (respectively a specific model that is well formed) is of interest and what are the specific usages. Indeed, depending on the well-formedness or not of each template constituent, other cases can be considered in addition to the previous one, whether the parameter submodel is well formed or not. Cases when the parameter model is not well formed are shown in Figure 6.

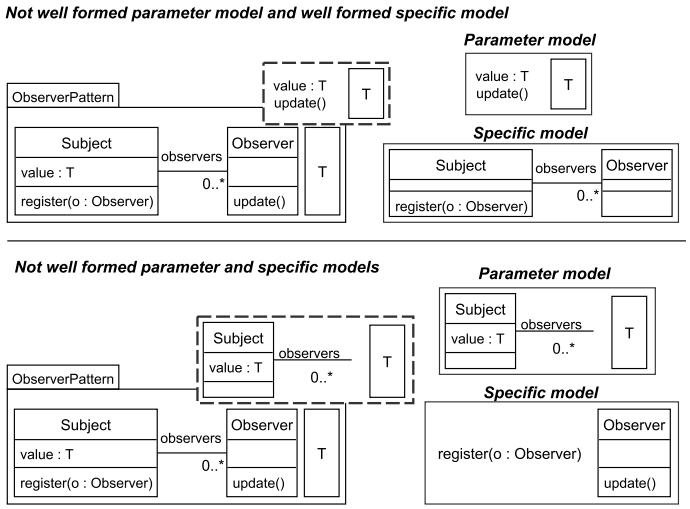

Fig. 6: Possible forms of template constituents

Not well formed parameter model, well formed specific model: This case is shown on top of Figure 6. Here, it is the specific submodel that is well formed and therefore provides the modeling structure of the template.

Not well formed parameter model, not well formed specific model: This last case is illustrated in bottom of Figure 6. Compared to the previous case, this one has

both submodels which are not well formed. Here, the structure of the template content is in the template in its entirety.

Table 1 gives a summary of all possible cases of template constituents.

Table 1: Possible forms of template constituents

		Specific Model	
		well formed	**not well formed**
Parameter Model	**well formed**	Case 1	Case 2
	not well formed	Case 3	Case 4

These cases being identified, they can be analyzed with regard to template instantiation. It is done in the following.

Let us consider cases 1 and 2 corresponding to a well formed parameter submodel. The situation was examined in Section 3. It leads to a resulting model where the well-formedness of its structure is brought by substituted elements of the context model. So, similarly to aspectual template binding, instantiation can be applied to any template having a well formed parameter submodel. This is an interesting result for the reuse of templates. It means that any template having this property can be applied both for aspectual and generative usages depending on the modeling needs.

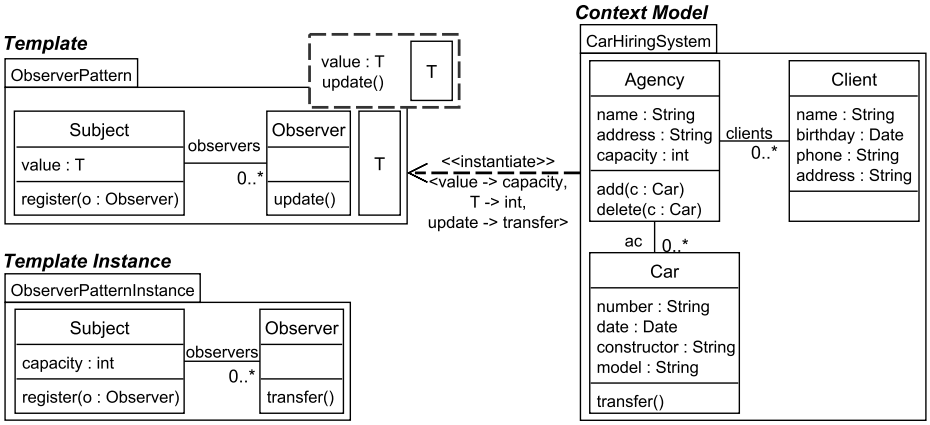

Fig. 7: Template instantiation with a not well formed parameter model

Let us continue with the two remaining cases corresponding to the situations where the submodel parameter is not well formed (Figure 6), regardless of its specific submodel. Figure 7 shows an example of template instantiation for case 3, accompanied with the resulting model. As can be observed, the parameter submodel is not well formed contrary to the specific model. Thus, in this case, the

template structure is provided by the specific model. Concerning case 4, similar comments can be made regarding template instantiation. Even if both template submodels are not well formed, the template they form can be instantiated. To be convinced, it suffices to modify the status of the *Subject* class as parameter in Figure 7. Despite this change, such a template continues to be applicable through an *instantiate* relationship.

The last two cases typically occur when modeling generics (e.g. C++ or Java) but also in partial template instantiation. Thus, it is important to handle them and ensure they are treated consistently through template instantiation. The following section specifically studies partial instantiation and concludes on the comparison between aspectual binding and template instantiation.

5 Partial Instantiation

UML templates allow partial binding when not all parameters are substituted and unsubstituted ones remain parameters in the resulting model which is therefore a template. Partial binding is a powerful feature that allows modelers to obtain richer templates through the composition of templates. It was deeply studied in our preceding work [18] for aspectual templates. Following UML principles concerning parameter substitution and propagation of the unsubstituted parameters, partial instantiation can be offered to benefit from additional capacities. It gives the ability to produce new templates from instantiated ones and, thus, sequences of instantiations. It also enables instantiation in multiple contexts.

Figure 8 gives an example of a partial instantiation between the *Observer-Pattern* template and the *CarHiringSystem* model. In this example the *Subject*, *value* and *T* parameters are bound in the substitution set of the instantiate relationship while the *observers* and *Observer* parameters are unbound. This figure also shows the result of this partial instantiation which is a new template named *ObservableAgency*. For this template, the parameter model contains the unbound *observers* and *Observer*, following the propagation strategy of UML for unsubstituted parameters.

One observation can be made concerning the propagation of unsubstituted parameters in the new template. This propagation is achieved with respect to specified substitution causing adaptation of method parameters. See for example the substitution of *Observer* by *Resource* in the *register* method of the *Agency* parameter, in the *ObservableAgencyResource* template instance. Depending on the substituted parameters, the resulting template may have a well formed model as parameter or not.

Templates resulting from partial instantiation can be further applied. They can help to construct other parts of the same system or serve as valuable artifacts in order to build parts of new systems. For applying a template resulting from partial instantiation to a new context, complete or partial instantiation can be used. Additionally, such a template can also be bound for aspectual usages as long as their parameter model is well formed. In our example, only instan-

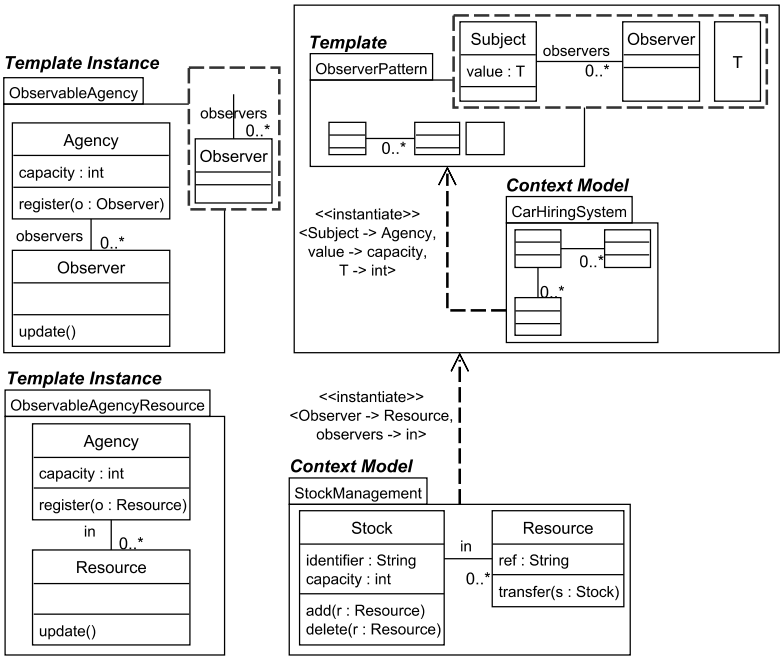

Fig. 8: Partial instantiation

tiation is enabled because the parameter model is not well formed. Figure 8 shows a partial instantiation of the obtained template *ObservableAgency* to get *ObservableAgencyResource*. This instantiation takes place in a new applicative context related to stock management. In this instantiation, the *Observer* and *observers* are substituted in order to produce a final model that combines ingredients from the two modeling contexts. Such a model can be useful for observing state changes between the two parts.

In this example, it is interesting to highlight that the same result could be obtained through an alternative sequence of instantiations: first, partially instantiating the *ObserverPattern* template in the *StockManagement* context and then instantiating the intermediate result in the *CarHiringSystem* context. Moreover, obtaining the same *ObservableAgencyResource* model with a complete instantiation would require merging *CarHiringSystem* and *StockManagement* models into a single model followed by a complete instantiation of *ObserverPattern* to this merged model. These equalities emphasize the compatibility between partial and complete instantiation and, thus, their consistency.

The preceding example also illustrates successive template instantiations from an initial template. Along a sequence of instantiations, the set of parameters decreases in intermediate templates. This may cause relaxation of structural constraints on the parameters. Such relaxation has the effect to enlarge the set of potential elements for substituting parameters in instantiation. This flexibility at the level of substitution is visible in the previous example. In the

obtained template *ObservableAgency*, *Observer* and *observers* parameters form a less-constrained structure than their counterpart in the initial template. This enables mapping them on a larger set of candidate elements when compared with the initial template.

Table 2 summarizes the situations studied in this paper. It characterizes applicability and results of examined relationships depending on whether the template is aspectual or not. This table should help to have a better understanding of the UML Template modeling space, particularly in case where parameters form a model.

Table 2: Template applicability and kinds of resulting models

	Aspectual Template ***(Well formed Parameter Model)***		**Not Aspectual Template** ***(Not Well formed Parameter Model)***	
	Applicable ?	*Kind of Resulting Model*	*Applicable ?*	*Kind of Resulting Model*
Aspectual Binding	Yes	Model	No	No Resulting Model
Instantiation	Yes	Model	Yes	Model
Partial Aspectual Binding	Yes	Aspectual Template	No	No Resulting Model
Partial Instantiation	Yes	Template	Yes	Template

6 Tool Support

A software environment dedicated to template based model engineering in Eclipse was previously implemented [18]. This environment [4] is composed of plugins which are based on the official EMF (Eclipse Modeling Framework), UML and OCL plugins. These plugins offer core functionalities to specify and verify aspectual templates well-formedness but also apply their binding in a compliant way with the UML plugin thanks to a specific profile. In addition, the plugins provide general and original facilities to support other modeling tasks targeting templates or user assistance such as automatic completion of template signature and binding inference. All the plugins functionalities are reusable in modeling tools that handle model templates.

Following the present work, this environment has been extended to include the capacity of instantiating templates. For that purpose, we added the following enhancements to existing plugins :

– An adaptation of OCL constraints for checking the consistency of template parameters and their substitutions when instantiation is applied. The constraints applied in that case are a subset of the ones for checking aspectual

[4] Eclipse plugins and modeling tool snapshots are available at `http://www.cristal.univ-lille.fr/caramel/MBE_Template/`

templates. These constraints mainly enforce that parameters and the substituted elements involved in a template instantiation have a compatible structure.
- An implementation of total and partial instantiation. The implemented algorithm proceeds by copying the core of the template into a new model and replacing copied parameters by substituted elements from the context model. Thanks to their compatibility with instantiation, this implementation is also applicable to aspectual templates.
- An extension of the current profile dedicated to templates with a new stereotype related to template instantiation, i.e., *InstantiationBinding*. It specializes the *TemplateBinding* UML metaclass and provides an OCL context for applying constraints due to template instantiation.

7 Related Works

This study started from UML template and showed that this concept is a general construct both for aspect-oriented and generative modeling based on parameterized models. A detailed review of related works regarding aspectual templates can be found in our previous work [18]. In these works, instantiation is sometimes mentioned as an underlying mechanism for binding aspectual templates [2, 9, 15] but it is not isolated as a full-right mechanism, as studied here. In the following, we review existing works that explicitly support template instantiation since it is the focus of this paper.

As already indicated, one motivation for template instantiation is the modeling of generic classes. [5] is a work that studies this need in UML by means of template classifier. The authors mainly focus on mechanism offered by this construct for expressing constraints on type parameters of represented generic classes and checking their substitutability in bindings. The work presented in [7] addresses the similar modeling need but aims to offer a stronger conformance for the binding of template classifiers. For that purpose, it proposes a set of well-formedness rules, additional to that of UML, to enforce the correctness of bound attributes and methods regarding their types, their membership and some of their meta-attributes. By offering generic classes similar to Java, Ecore (the metamodel of EMF) [17] can also be cited as a work dealing with this need. Thanks to this feature, models expressed in Ecore can contain declaration of generic classes or use instantiation of generic classes for typing attributes and methods. This feature also improves the capacities delivered by EMF for code generation (i.e., generate generic code).

The Catalysis approach [6] proposes model frameworks in order to design reusable packages. A model framework is a form of parameterized package containing placeholders which are names that can be substituted with actual type names when the framework is instantiated. Each instantiation of the framework provides its own substitution of the placeholders. The names of attributes and associations of placeholder types are themselves placeholders. This approach, based on string substitution, is realized in the XMF tool.

[14] studies the support of genericity in component models. This work proposes a structural pattern to extend an existing component model with concepts for genericity. With this pattern, a component model can be made generic through parametrization. Elements that can be exposed as parameters are types of input and output ports, types of component implementation and the number of nested subcomponents. This work also introduces an algorithm to instantiate a generic component model. The use of the pattern is demonstrated by extending the SCA component model.

In [11], the authors present a notion of model template and its instantiation mechanism in the context of the MetaDepth framework. In this work, a model template specifies its generic modeling structure by means of the "concept" construct which is a separate model expressing both the parameters and a set of structural requirements on these parameters. In some way, the "concept" construct is related to the notion of "parameters as a model" and has a similar purpose but it is not part of the template body. Instantiation of a model template is done by importing substituted elements from a model conforming to the concept into a new model constructed from the template body.

Compared to these works, the present contribution differs on several main points. First, only one of the existing works [5] takes place inside the scope of standard UML but only for template classifiers. Second, all these works except [11] do not consider template parameters as a fully structured model. As discussed in the paper, this requirement allows us to overcome possible inconsistencies when instantiating templates. It also provides a way to better characterize and control the usages of templates during processes. A last difference between the present work and existing ones is related to partial instantiation. To our knowledge, no existing approach offers capacities comparable to this feature. As a result, interesting capacities are enabled like instantiation of template in multiple contexts or the construction of complex assembly, mixing aspectual and generative usages of templates through partial binding and instantiation.

8 Conclusion

Starting from UML templates and their binding relationship, this work isolated instantiation of templates with "parameters as a model". As a consequence, new capacities were offered for the reuse of templates and the construction of new models from their complete or partial instantiation. This work also provided a characterization of the resulting UML Template modeling space. More generally, it contributes to enrich template-based MDE capacities.

In future work, we plan to focus on order and equivalence of instantiation sequences, the way we did in one of our works [12] for aspectual binding. Another interesting perspective to investigate is the study of alternative strategies for unsubstituted parameters, like no propagation or the use of default values for them. Lastly, we are working on the formalization of the template construct by exploiting our previous work on model inclusion and the notion of submodel [4]. We

expect this formalization will help to achieve a better theoretical understanding and generalization of template for the quest of model reuse.

References

1. Allon, M., Vanwormhoudt, G., Carré, B., Caron, O.: Template Based MDE. In: 4ème Conférence en Ingénierie du Logiciel (CIEL 2015) (2015), https://hal.archives-ouvertes.fr/hal-01162652
2. Berg, H., Møller-Pedersen, B.: System Analysis and Modeling: Theory and Practice: 7th International Workshop, SAM 2012, Innsbruck, Austria, October 1-2, 2012. Revised Selected Papers, chap. Type-Safe Symmetric Composition of Metamodels Using Templates, pp. 160–178. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
3. Bottoni, P., Guerra, E., de Lara, J.: A language-independent and formal approach to pattern-based modelling with support for composition and analysis. Information and Software Technology 52(8), 821–844 (2010)
4. Carré, B., Vanwormhoudt, G., Caron, O.: From subsets of model elements to submodels, a characterization of submodels and their properties. Software and Systems Modeling 14, 861–887 (May 2015)
5. Cuccuru, A., Radermacher, A., Gérard, S., Terrier, F.: Constraining type parameters of UML 2 templates with substitutable classifiers. In: Model Driven Engineering Languages and Systems, pp. 644–649. Springer (2009)
6. D'Souza, D., Wills, A.: Catalysis: Objects, Components, and Frameworks with UML. Object Technology Series. Addison-Wesley (1998)
7. Farinha, J., Ramos, P.: Extending UML templates towards computability. In: Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD'2015). pp. 122–133. SciTePress (February 2015)
8. Herrmannsdörfer, M., Hummel, B.: Library concepts for model reuse. Electronic Notes in Theoretical Computer Science 253(7), 121–134 (2010)
9. J.Klein, J.Kienzle: Reusable aspect models. In: 11th Aspect-Oriented Modeling Workshop, Nashville, TN, USA. Citeseer (2007)
10. J.Whittle, P.Jayaraman, A.Elkhodary, A.Moreira, J.Arùjo: MATA: A unified approach for composing UML aspect models based on graph transformation. In: Transactions on Aspect-Oriented Software Development VI, pp. 191–237. Springer (2009)
11. de Lara, J., Guerra, E.: From types to type requirements: genericity for model-driven engineering. Software & Systems Modeling pp. 453–474 (2013)
12. Muller, A., Caron, O., Carré, B., Vanwormhoudt, G.: On Some Properties of Parameterized Model Application. In: Proceedings of 1st European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'05). LNCS, vol. 3748, pp. 130–144. Springer (November 2005)
13. OMG: Auxiliary Constructs Templates, chapter 17. UML 2.4.1 Superstructure Specification (2011)
14. Pérez, C., Bigot, J.: Increasing Reuse in Component Models through Genericity. Formal Foundations of Reuse and Domain Engineering Lecture Notes in Computer Science 5791, 21–30 (2009)
15. Reddy, Y., Ghosh, S., France, R., Straw, G., Bieman, J.M., McEachen, N., Song, E., Georg, G.: Directives for composing aspect-oriented design class models. In:

Transactions on Aspect-Oriented Software Development (I). vol. I, pp. 75–105. Springer (2006)
16. S.Melnik, P.A.Bernstein, A.Halevy, E.Rahm: A semantics for model management operators. Microsoft Technical Report pp. 1–12 (2004)
17. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Addison-Wesley (2008)
18. Vanwormhoudt, G., Caron, O., Carré, B.: Aspectual templates in UML. Software and Systems Modeling pp. 1–29 (2015), http://dx.doi.org/10.1007/s10270-015-0463-3