An Automated Process for Designing UML Profiles

François Lagarde, Frédéric Mallet, Charles André, Sébastien Gérard, François Terrier

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

An Automated Process for Designing UML Profiles

François Lagarde — Frédéric Mallet — Charles André — Sébastien Gérard — François Terrier

N° 6599

July 2008

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6599--FR+ENG

An Automated Process for Designing UML Profiles

François Lagarde*, Frédéric Mallet†, Charles André†, Sébastien Gérard* , François Terrier*

Thème COM — Systèmes communicants
Projet Aoste

Rapport de recherche n° 6599 — July 2008 — 17 pages

Abstract: Building a UML profile may be a tedious and error-prone process. There is no precise methodology to guide the process or to verify that all concepts have been implemented once and only once. Best practices recommend starting by gathering concepts in a technology-independent *domain view* before implementation. Still, the adequation between the domain view and the implementation should be verified.

This paper proposes an automatic process to transform a domain model into a profile-based implementation. To reduce *accidental complexity* in the domain model and fully benefit from advanced profiling features in the generated profile, our process uses the *multi-level paradigm* and its *deep characterization* mechanisms. The value of this paradigm for the definition of UML profiles is assessed and applied to the subset of a recently adopted OMG UML Profile, an excerpt of the MARTE time profile.

As a by-product, our process involves an inexpensive profile-based implementation of the multi-level paradigm within UML2 tools.

Key-words: domain model, multilevel modeling, UML profiles, deep instantiation

This report is based on the INRIA research report RR-6525, which has also been published as a CEA research report: DTSI/SOL/08-115/FL

* Commissariat á l'Énergie Atomique, CEA-LIST
† Université de Nice-Sophia Antipolis

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Un processus automatique pour la génération de profils UML

Résumé : Construire un profil UML peut s'avérer fastidieux. Il n'y a actuellement aucune méthode reconnue pour guider cette définition, pour vérifier que tous les concepts métiers ont été implémentés une et une seule fois et pour éviter les pièges nombreux. Quelques contributions essayent d'éclaircir le chemin et il est généralement reconnu qu'il faut commencer par faire un modèle métier indépendant d'une technologie particulière (y compris d'UML) avant de se lancer dans des considérations spécifiques à l'implantation. Malgré tout, le passage du domaine métier à l'implantation reste difficile.

Nous proposons ici un procédé automatique pour transformer le modèle métier en un profil UML. Pour réduire la complexité inutile du domaine métier et obtenir toutefois un profil précis et avancé nous préconisons l'utilisation du *paradigme multi-niveaux* et de la *caractérisation en profondeur*. L'intérêt de ce paradigme pour la définition de profils est évalué et son utilisation est illustrée sur un extrait simplifié du profil UML MARTE, récemment adopté par l'OMG.

Notre procédé implique l'implantation dans les outils UML2 du paradigme de modélisation multi-niveaux et nous présentons cette implantation comme une contribution dérivée.

Mots-clés : modèle métier, modélisation multi-niveaux, profils UML, deep instantiation

1 Introduction

Building a profile for the Unified Modeling Language (UML) [1] may be a tedious and error-prone process. There is no precise methodology to guide the process or to verify that all concepts have been implemented once and only once. The quality of the profile should be assessed with metrics and its adequation with the target domain should be verified.

This lack of methodology results in collections of heterogeneous profiles that can exhibit both inconsistencies and overlaps. To overcome this issue, Bran Selic [2] suggests to start with a technology-independent description (a *domain model*) that can be used as a communication interface between the domain experts and the implementation team.

Starting from this domain model, several implementation choices are possible: using a *Domain-specific language* (DSL), a pure meta-modelling approach or building a profile. Building tools for domain-specific languages can be quite expensive. The target community being generally small, there is little economic incentive for tool vendors to build such tools. The result has often been the creation of incomplete tool suites and lack of support. For this reason, companies often turn to general-purpose languages, even if the modeling capabilities available are ill-suited to the domain requirements.

In this paper, we focus on the profile-based solution. *Profiles* incrementally build on the general-purpose UML to define only the new concepts required to represent domain-specific elements. This reduces the cost of building tools by reusing existing ones and limits the investment required for the learning process. The domain model is mapped onto the UML metamodel by defining stereotypes that extend existing UML concepts (metaclasses) in order to modify or refine their semantics. This two-stage process allows designers to focus on domain concepts and their relationships before dealing with language implementation issues. However, the gap between the domain model and the profile may be particularly difficult to fill because of the potential inability of UML or, more generally, of the object-oriented paradigm to capture domain concepts.

One of the essential modeling capabilities missing in the object-oriented paradigm is the ability to model multiple levels. This failing may lead to *accidental complexity* [3]. Well-known design patterns (like "Item Description" [4] or "Type Object" [5]) are artificial workarounds to mimic multi-level scenarios with only two levels (classes and objects). C. Atkinson and T. Kühne [6] propose to go beyond the traditional two levels and promote a multi-level modeling paradigm that combines *deep characterization* with *deep instantiation*.

Use of the multi-level modeling paradigm is particularly relevant to defining profiles. Profiles are cross-level mechanisms between the metamodel and the model levels and mix multi-level concepts: model libraries elements, metaclasses representing descriptors (*e.g.,* Classifier) and metaclasses representing items (*e.g.,* InstanceSpecification). Such concept mixtures can be found in the newly adopted UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE [7]), the result of significant collaborative effort by domain experts. In MARTE, the profile is sometimes far from the domain model. Designers apply sophisticated design patterns, which make it hard to ensure that every domain concept is actually implemented in the profile.

We propose an automatic process to generate a UML profile directly from the domain model. To reach the same expressiveness as in MARTE, our domain model relies on the multi-level paradigm. Therefore, as a by-product we propose a UML profile for domain specification that offers multi-level annotations and a practical solution for implementing the deep instantiation mechanism within UML 2.x-compliant tools. This profile is used to specify a domain model, then the model is automatically transformed into an equivalent profile-based implementation.

We have used the Time subprofile [8] of MARTE to illustrate our approach. We begin by taking an excerpt from the Time subprofile and we explain why it may seem unnecessarily complex (Section 2). In Section 3, we propose another domain model for the Time built on the multi-level paradigm. We then introduce our profile for Domain Specification and show how multi-level annotations are applied and used to transform the domain model into a profile-based implementation (Section 4). Section 5 outlines the differences between a process that leverages the multi-level modeling paradigm and the approach followed by MARTE designers. Finally, we discuss some related works in Section 6 before concluding.

2 Motivation

2.1 The MARTE Time Profile

Figure 1 is an excerpt from the MARTE Time profile. Note that this paper does not elaborate on the underlying concepts of the Time profile, which are described in a previous work [8]. This paper focuses on its design intents using a small number of concepts to justify our approach. The figure contains three parts: the left bottom part is a simplified description of the MARTE Time profile, the right-hand part shows libraries associated with the profile, and the upper part gathers UML metaclasses and relationships referred to by MARTE Time profile. The diagram in this latter part is extracted from UML::Classes::Kernel. Annotations (A) and (B) and annotated dependencies (A) and (C) are not part of the specifications, they are used below to ease the understanding.

Profile core consists of two stereotypes (ClockType and Clock). These stereotypes provide for mechanisms to create new clocks and to put together (within a clock type) features common to several clocks.

A close look at the package Time shows that Time, and more generally MARTE profiles make *advanced usages of stereotypes*. In the profile, the stereotype properties (tags) are not limited to primitive types. Some are typed by UML metaclasses and even by stereotypes.

The use of the metaclass Property for the attribute resolAttr avoids premature specification of the type and name of the property that models the clock resolution, since they may differ significantly depending on the system considered. Sometimes the type may be a real number, representing the precision of the clock relative to a reference clock, sometimes it may be given as an integer or even as a mere enumeration (*e.g.,* fine, medium, coarse). The value of attribute resolAttr identifies within a user's model the specific property that has the semantics of clock resolution, whatever its actual type and name. This mechanism

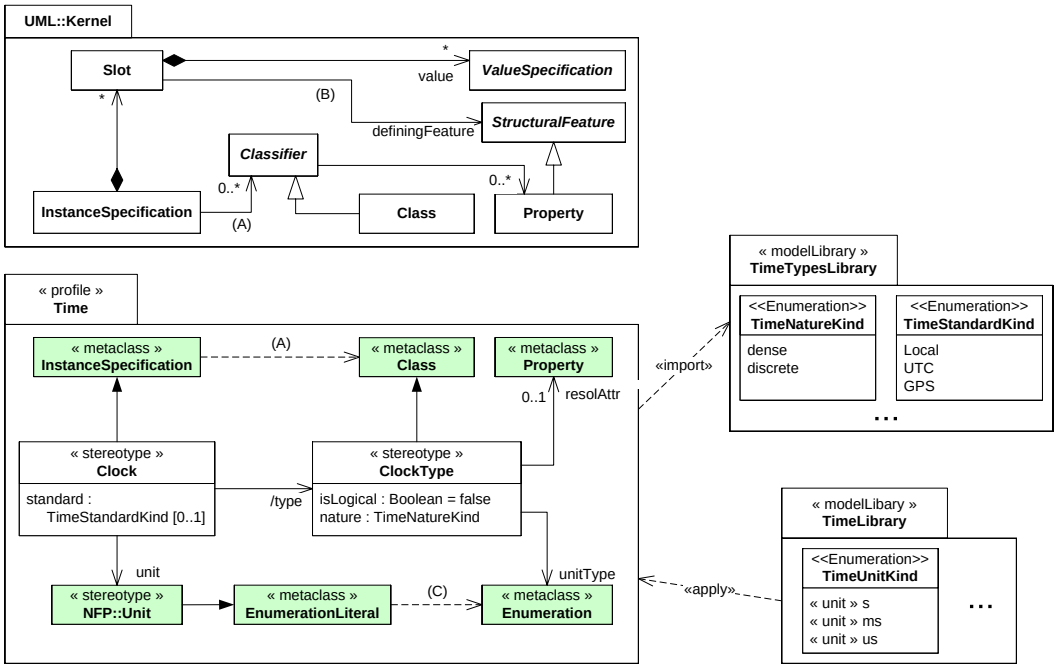

Figure 1: Excerpt from the MARTE Time Profile

is extensively used in MARTE; its usefulness has been illustrated on a realistic example of platform-based transformations of real-time operating systems [9].

Typing attribute unit directly by the stereotype NFP::Unit instead of the metaclass EnumerationLiteral leads to a more precise characterization of what unit is. Not any enumeration literal can be used as a time unit; it has to be stereotyped by NFP::Unit thus giving additional information like conversion factor. The same effect could have been achieved typing by the metaclass EnumerationLiteral combined with a constraint. This alternative solution is often preferred because of the failure of most commercial tools to support stereotype-to-stereotype associations.

2.2 Multi-level aspects in the Time Profile

In Figure 1, at least two modeling levels have been mixed together in one.

The first, or metamodel level is easily identified by the metaclass keyword (*e.g.,* InstanceSpecification, Class, Enumeration, EnumerationLiteral). A careful look at the existing relationships in the UML metamodel is, however, required to accurately qualify the modeling levels involved. The two relationships (A) and (B) in Figure 1, in the UML metamodel, are used to tell what belongs to classifier level from what belongs to instantiation

level. The relationship (A) tying an InstanceSpecification to one of its Classifier shows that clocks are instances whereas clock types are types. This relationship is made concrete in the profile by the derived attribute type of stereotype Clock. type is a subset derived attribute of attribute classifier from the metaclass InstanceSpecification.

The use of the metaclass Property for the attribute resolAttr is motivated by the relationship (B) between the metaclass Slot and the sub-class of StructuralFeature: the metaclass Property. Ultimately, the resolution value is given in an instance slot. The dependency relationship (C) is also a relationship between the metaclasses, but is only defined in the profile. It states that a clock type refers to the set of acceptable units (unitType), whereas a clock refers to a specific unit from among this set (*e.g.,* s, ms). This specification is enforced by an OCL constraint.

The second modeling level corresponds to the use of model libraries, which are specific UML constructs that purposely escape classification by level. Elements from model libraries can be used at any levels, in a metamodel, a profile or a user model. This is the case of the primitive type Boolean, defined in the UML standard model library, which is used to define whether a clock type is logical or not. However, certain of MARTE data types, those defined in TimeTypesLibrary, are intended for use in the profile and by users (*e.g.,* TimeNatureKind, TimeStandardKind). Others, defined in TimeLibrary, should only be used at user level (*e.g.,* TimeUnitKind) and are clearly situated "below" the profile level since TimeLibrary applies the profile. Using this library within the profile would result in cyclic definitions.

2.3 Applying the Time Profile

Figure 2 illustrates the use of the Time profile for two clock types. The left-hand side of Figure 2 shows a conventional usage of a profile. The clock type Chronometric is defined in the model libary TimeLibrary. It should be used to model discrete clocks, related to physical time, which are not necessarily perfect. The property resolution, whose type is Real, is selected to play the role of resolAttr. The clock type Cycle represents a discrete logical clock that uses units like processorCycle or busCycle to date event occurrences. For this second clock, there is no need for a property playing the role of resolAttr.

The chronometric clock cc1 completes the specification by selecting one specific unit (s) from among the literals defined in the enumeration TimeUnitKind. It also chooses a standard and a value for the resolution. The cycle clock p1 also selects a unit, but from a different enumeration, CycleUnitKind. Clocks of the same type must use compatible units (from the same enumeration). In that regard, the clock types acts as a dimension.

The right-hand side of the figure is a representation of the domain view using clabjects as defined by Atkinson and Kühne. This representation combines notational conventions from UML classes and instance specifications. Clabjects have *fields*, to unify meta-attributes, attributes and slots, and thus flatten the different modeling levels into one. Horizontal dashed lines serve to identify the logical modeling levels.

In the UML view, Clock and ClockType are both represented at the same level, as stereotypes. However, ClockType is a *descriptor* for a set of Clock (as defined by the pattern

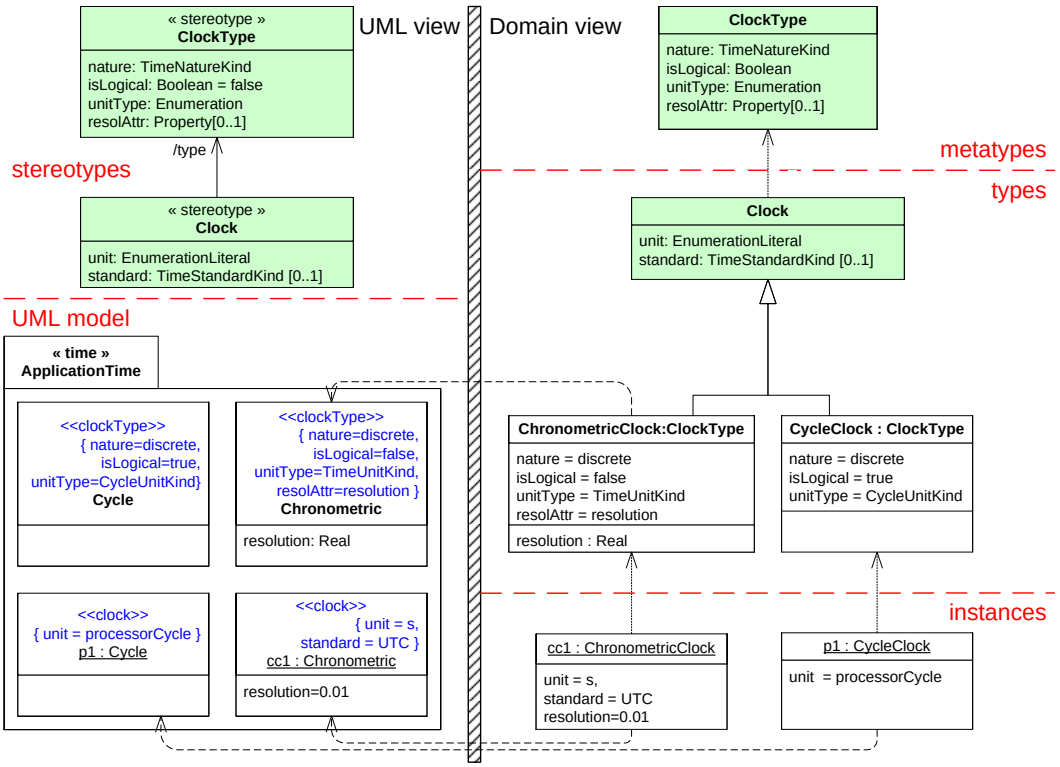

Figure 2: Examples of clocks and clock types

Item Descriptor). They therefore belong to a different modeling level. In the domain view, the three levels are clearly separated by the horizontal dashed lines.

2.4 Limitation

Such a design strategy defines clock properties in several steps. Part of the information is given at the class level and, the rest, at the instance level. Some features are progressively refined using the relationship between Classifier and InstanceSpecification. This is the case for the property resolAttr that references a user-defined property of the clock type Chronometric at the class level. The value of this property is given at the instance level in the related slot. The same is true for the unit. At class level, a set of possible units is identified, but choice of the actual unit takes place only at the instance level. Certain features such as nature are only relevant at class level and others only relevant at instance level (*e.g.*, standard).

To comply with UML requirements, the adopted solution involves complex, albeit "neat", workarounds. The most visible drawback is that meaningful information is scattered across

different locations, corresponding to the different modeling levels: stereotype, class, instance specification, and consequently, obscurs the domain information. Information is further hidden under elements that do not contribute to definition of the domain but are required by limitations of the modeling language. This is the case of the property resolAttr.

3 Multi-level modeling with deep instantiation

The inability of languages like UML to represent multiple classification levels in a same modeling level has already been pointed out by C. Atkinson and T. Kühne [10,6].

Their initial observation was that information about instantiation mechanisms was bound to one level. The most visible effect is that information carried by attributes cannot cross more than one level of instantiation. This obstacle is deemed to be a major impediment and also tends to increase the complexity of models. The abovementioned authors therefore call for flattening the modeling levels through use of deep instantiation. The key concept is a *potency* that characterizes any model elements. The potency is an integer that defines the number of instantiations (depth) that may be applied on elements. Properties and slots are uniformly called *fields*. Fields of potency higher than one are meta-attributes. Fields of potency one are regular attributes and fields of potency zero are regular slots. Fields are thus made to persist throughout each of the instantiations (as opposed to the *shallow* instantiation). Each instantiation of a field (property with potency annotation) decreases the value of potency by one.

Designers of the Time profile obviously had to solve similar problems. We know it for sure since some authors of this paper were part of the ProMARTE consortium in charge of defining the profile MARTE. Their solution plays on the "instance of" relationship between InstanceSpecification and Class to put, at the appropriate level, information concerning the single concept Clock. This results in information scattering. Deep instantiation, on the other hand, makes multiple levels explicit and offers new design opportunities. In this approach, both stereotypes Clock and ClockType represent the same concept. They need not be apart in the domain model, even though they are separate in the final implementation. Instead, instantiation levels are identified by potency independently from the underlying UML implementation details.

Figure 3 shows how we can start from the actual specification requirements and progressively refine it into a usable domain model on which the automatic transformation applies. This refinement goes in three steps.

First, an analysis of different usages and interpretations of time in various application domains related to real-time and embedded systems shows that a clock can be characterized by its nature (dense or discrete), whether it is logical or chronometric, its resolution, unit and standard. More features were actually identified, however we have selected here a representative subset to illustrate our approach. This results in the left-most part of the figure. Sometimes, the type of these features is obvious, sometimes there is no obvious common type. In such a case, like for the resolution, we can use the mechanism presented

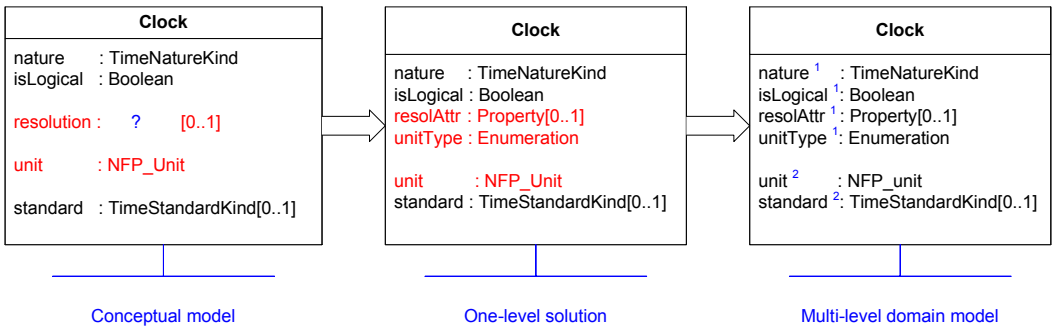

Figure 3: Clock definition with potency information

in Section 2.1 to allow maximum flexibility. In the middle-part of the figure, resolution is substituted by resolAttr of type Property, a higher order feature.

Second, a careful look at the unit shows that clocks can be classified according to the kind of units they use. Therefore, we introduce the concept of unitType to represent a set of acceptable units. Since an NFP_unit is an enumeration literal, it seems natural to define unitType as an Enumeration.

Third, it appears that part of the information is common to a collection of clocks whereas another part is specific to each individual clock. Without the multi-level paradigm, this would result (as in MARTE) in splitting Clock into two parts, one to represent the clock type and the other to represent the clock instances. Using the multi-level paradigm, we simply assign the potency to features according to the level they belong to. Adequate potencies are shown in the right-most part of the figure. In the end, fields that operate at instance level (formerly Clock) should have the highest potency. Here, the highest potency is two, since we have three levels (metatypes, types, instances) as shown by the horizontal dashed line in Figure 2. Fields that operate at class level (formerly ClockType) should have a potency of one.

The final description is very concise and has the same expressiveness as the original Time profile (Figure 1). The potency value makes it clear whether a property must obtain its value at the first or the second instantiation level.

In the next section, we propose a practical means for implementing such mechanisms within a UML tool.

4 Our proposal

This section presents the mechanisms that have been devised for the creation of UML-based domain specific languages that supports the multi-level modeling paradigm. Our proposal is based on a three-step process. The first step is to specify the domain model. Elements of the model have properties that are annotated with a *potency*. This artifact is used, in

a second step, to automatically derive a UML profile from the specification. Our premise is that using an automated transformation reduces the gap between the domain and the profile and ensures that every concept in the domain is actually implemented in the profile. Application of this profile, in a third step, enables modeling of elements that comply with the domain model specification. In subsequent sub-sections follows a step-by-step illustration based on the Time profile.

4.1 Domain model specification

We have defined a UML profile (called DomainSpecification) for domain specification. It is used to annotate a UML model with information required for multi-level modeling. This enables to declare models in a way similar to the one presented in Figure 3. This profile consists of stereotype Field (see Figure 4, left part) that extends metaclass Property and carries potency information.

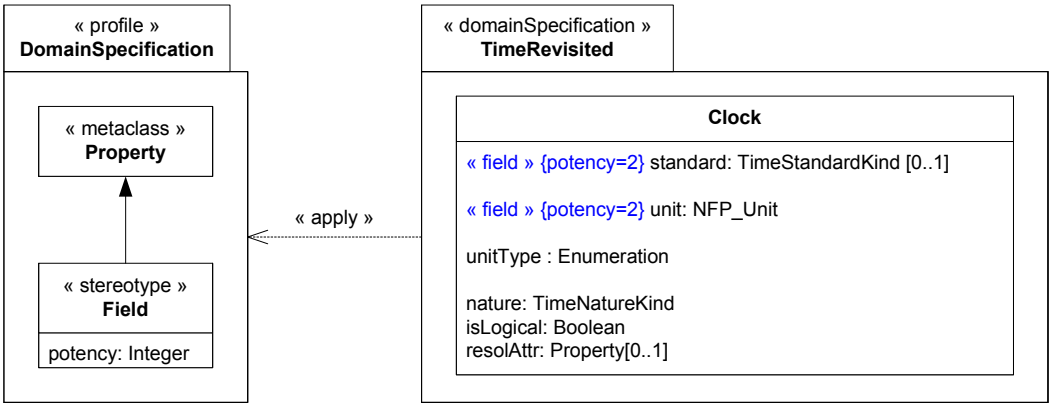

Figure 4: Domain specification profile and its usage

We start with a class model and apply the stereotype Field to specify potency (package TimeRevisited in Figure 4). Potency is optional. A property without potency is considered a *regular* attribute and is equivalent to a potency of one.

4.2 Automated profile-based implementation of the domain model

In this second step, we use the domain specification as an artifact to build a profile-based implementation. The result is a profile with enough stereotypes to describe each instantiation level for each concept. Consequently, we automatically derive the profile shown in Figure 5.

A straightforward algorithm is used for transformation. Each class gives rise to as many stereotypes as instantiation levels. The name of each generated stereotype is derived from the name of the initial class suffixed by an integer that reveals the level of instantiation. In

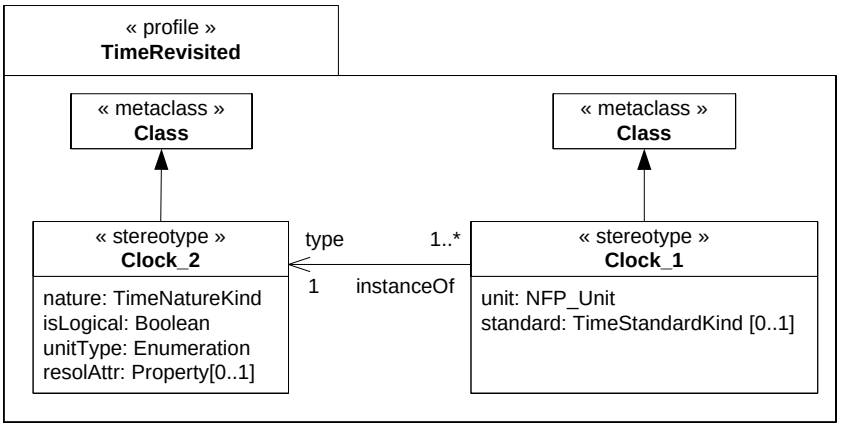

Figure 5: Time profile generated from the domain specification

our example, there are two levels (potency=2 and potency=1), so we derive two stereotypes (Clock_2, Clock_1). Each stereotype systematically extends the metaclass Class.

We start by creating the stereotypes at the highest level. After each step, the potency of each field is decreased by one. When the potency is one, an attribute is added to the stereotype with the same name and type as the related field. If the potency becomes lower than one, the field is discarded (and no longer used).

On Figure 5, the clock concept has been mapped onto two stereotypes. The stereotype Clock_2 carries information belonging to the first instantiation level (formerly ClockType) and Clock_1 is the final level (formerly Clock). The association between the stereotypes of two consecutive levels is added to unambiguously relate a deep instance (stereotyped by Clock_1) to its deep type (stereotyped by Clock_2).

4.3 Applying the generated profile

In this section, we apply the generated profile to declare the chronometric and cycle clocks (Figure 6).

Modeling of Cycle clock entails two new classes, one stereotyped by «clock_2» and the other by «clock_1». The property type avoids mixing Cycle clocks with Chronometric clocks. For instance, cycleClk is associated with CycleClock, not ChronometricClock.

The structure of this model is, at first glance, similar to the model using the original MARTE constructs (Figure 2). One obvious difference, however, appears in the metaclasses used as bases for our stereotypes. In MARTE, clocks were instance specifications of classes ChronometricClock and CycleClock, themselves stereotyped by «clockType». The instance specifications carry information about the slots of this class and also provide information like values relating to properties defined by the stereotype Clock. With the generated profile, since both Clock_2 and Clock_1 extend Class, we must express the fact that an instance

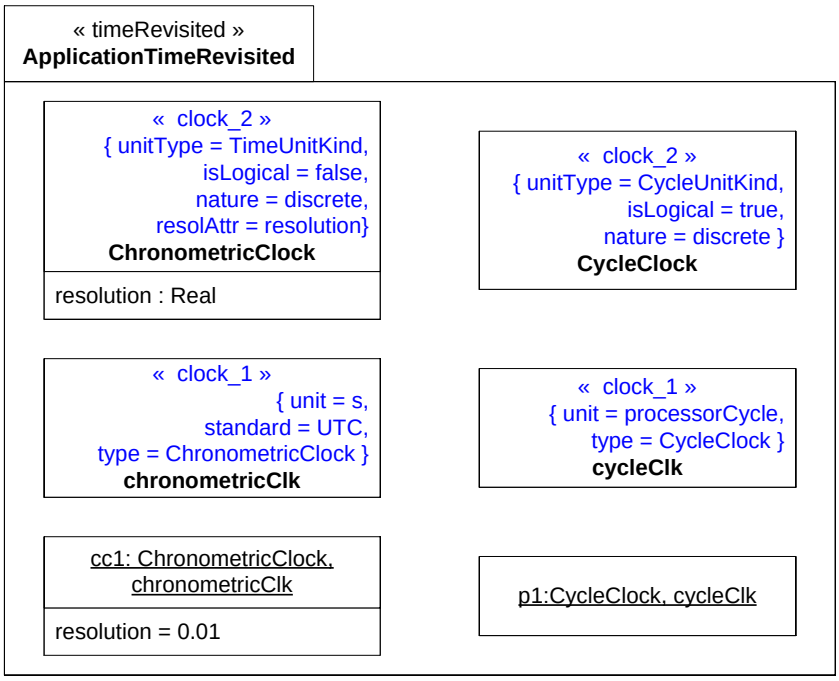

Figure 6: Clock definition with the generated profile.

of a clock has two classifiers, each of which carries properties related to one level. These differences are discussed thoroughly in the next section.

5 Discussions on our approach

This section compares our proposed approach with the one followed by the MARTE designers and by profilers in general. It describes the process workflow differences, then discusses possible extensions to our approach.

5.1 Design flow comparison

Figure 7 shows comparison of the two process workflows, from conceptual domain definition to profile creation.

MARTE designers applied a two-stage process: domain description and manual mapping of domain concepts onto profile constructs. The second stage is a very sensitive activity, since different designers may use different design solutions to map a given concept. This makes it difficult to assess the implementation.

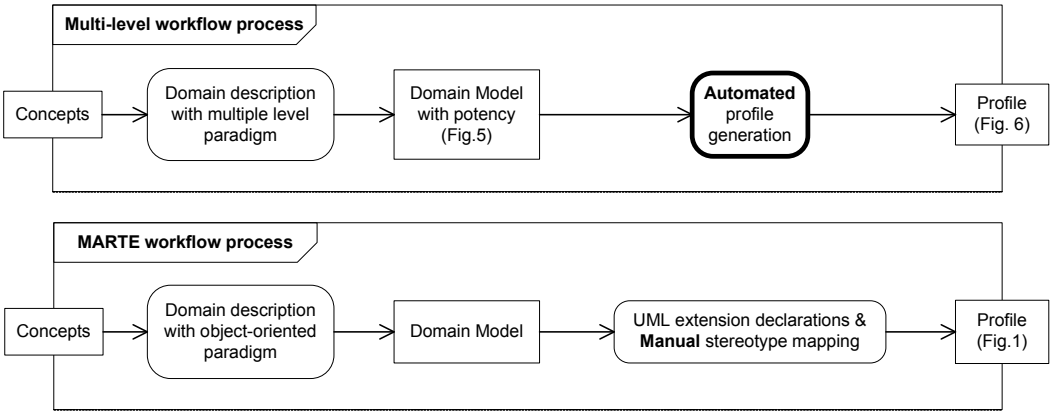

Figure 7: Design activity flow comparison

Use of a multi-level modeling paradigm to build the domain model leads to an automatic process that reduces the gap between domain description and profile. It is no longer necessary to look for equivalent stereotype constructs. The domain specification embodies information about all levels. Level separation takes place automatically, by specifying field potency, thus providing a reliable decomposition. Maintenance of models is made easier and consistency is afforded between domain model and profile.

5.2 Possible extensions

Currently the profile DomainSpecification is minimal. Nevertheless, new stereotypes can be introduced to guide the profile generation. Remind that, one of the original reason for building a profile was to be able to customize a model according to a particular point of interest. Compliant tools should (even if it is mostly not the case now) provide an easy support to hide or restore annotations of a given profile. Following that idea, our profile could be customized differently to build a profile fitting the design team legacy. Each team can implement its own generation rules. Explicit profile generation process and tool support allow designers to assess different candidate generation processes.

As possible customizations, we can allow a systematic naming convention and the selection of base metaclasses more adequate than the by-default Class. This kind of customization would allow us to generate the exact same result as in the actual MARTE specification where several metaclasses other than Class have been used (InstanceSpecification, Event, Observation, Activity...).

As illustrated in this paper, using generic metaclasses like Property and Class often leads to very flexible solutions. However, this may also lead to solutions with little semantics. It is often preferrable to choose types and metaclasses that better represent domain concepts. For instance, we could also have defined the unit as being a Property instead of an enumeration

literal. In that case, we had the constraint of being compatible with NFP_unit defined in MARTE NFP subprofile. Being more general than an enumeration literal would have prevented us from using NFP_unit-specifics, like its conversion factor.

6 Related work

The core motivation of this work is to facilitate building of UML-based domain-specific languages. In that matter, two communities confront each other, the meta-modeling and the profiling communities. Light-weight solutions often involves the creation of a profile, whereas more complete solutions require the use of meta-modeling. This work is only concerned with the building of profiles. However, use of multi-level paradigm simplifies the domain model. This multilevel-aware domain model could also be used by meta-modeling tools to produce more faithful code.

In this paper, and in multi-level modeling in general, we focus on the relationship "instanceOf". Other approaches, related to meta-modeling, consider relationships in general, "instanceOf" being just one of them. Other relationships include association, dependencies, conformance, composition....

In profile-based approaches, despite the ever increasing number of profiles being built in many domains, there is little published literature available to support the process as a whole.

Fuentes and Vallecillo [11] point to the need for first defining a domain model (using UML itself as the language) to clearly delineate the domain of the problem. In a more recent paper [2], Bran Selic describes a staged development of UML profiles and gives useful guidelines for mapping domain constructs to UML.

Our proposal also leverages the use of a domain model but explores multi-level modeling capabilities at this stage. Almost all the material available on multi-level modeling can be found in research efforts conducted by Kühne and Atkinson. They have studied the foundations of such modeling and proposed an implementation based on UML 1.x constructs [10]. This work now needs to be aligned on UML 2.x. More recently, Kühne and Schreiber [12] explored possibilities for support of deep instantiation in Java.

The context of our proposal is somewhat different. We assess values of deep instantiation mechanisms in the context of UML profile definitions, then demonstrate that the current UML specification already includes mechanisms for accessing the realm of multi-level modeling.

7 Conclusion

This paper presents an automatic process for generating a UML profile from a domain model by leveraging the use of the multi-level modeling paradigm. The process begins with the specification of concepts required to cover a specific domain. This design activity uses a profile to annotate concepts (mainly classes and properties) with multi-level information. Properties become fields and carry potency information that indicates their intended in-

stantiation level. The domain specification is then used to map elements onto equivalent profile constructs. The result is a profile-based implementation of the domain model that contains all stereotypes required to represent the concepts of the domain and their different instantiation levels. Application of this profile thus enables deep instantiation and modeling of elements complying with the domain model specification.

The proposal is illustrated with an excerpt from the Time sub-profile part of the recently adopted MARTE Profile. Use of the multi-level modeling paradigm provides new design opportunities and enables simplifications. It facilitates the domain specification by limiting implementation considerations. The domain description is more concise and clarifies the modeling levels. The resulting domain model gathers in a single class all information related to a given concept, whereas it was previously scattered over several classes.

Our proposal entails several model transformations. They are being automated in an Eclipse environment as a plugin of the open source Papyrus[1] UML tool. This tooling support will enable generation of profiles that support domain elements and include the necessary OCL rule enforcements. Assessment of the user's model should be made automated.

We advocate for a well-defined process that could consistently be used to define coherent profiles. This process must rely, as much as possible, on automatic transformations. The use of the multi-level paradigm is not specific to profiling and could also be used in meta-modeling approaches. Such a paradigm deserves to be further explored and its adequation with more general approaches still need to be assessed.

References

[1] Object Management Group: Unified Modeling Language, Superstructure Version 2.1.1 formal/2007-02-03.

[2] Selic, B.: A Systematic Approach to Domain-Specific Language Design Using UML. International Symposium on Object and Component-Oriented Real-Time Distributed Computing **00** (2007) 2–9

[3] Brooks, F.P.: No silver bullet essence and accidents of software engineering. Computer **20**(4) (1987) 10–19

[4] Coad, P.: Object-oriented patterns. Communications of the ACM **35**(9) (1992) 152–159

[5] Johnson, R., Woolf, B. In: Type Object. Volume 3. ADDISON-WESLEY (October 1997) 47–65

[6] Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software and Systems Modeling (2007)

[1] `http://www.papyrusuml.org/`

[7] Object Management Group: UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE), beta 1. (August 2007) OMG document number: ptc/07-08-04.

[8] André, C., Mallet, F., de Simone, R.: Modeling time(s). In Engels, G., Opdyke, B., Schmidt, D.C., Weil, F., eds.: MoDELS. Volume 4735 of Lecture Notes in Computer Science., Springer (2007) 559–573

[9] Thomas, F., Delatour, J., Terrier, F., Gérard, S.: Towards a framework for explicit platform-based transformations. In: ISORC, IEEE Computer Society (2008) 211–218

[10] Atkinson, C., Kühne, T.: The Essence of Multilevel Metamodeling. UML - The Unified Modeling Language. Modeling Languages, Concepts, and Tools **2185** (October 2001) 19–33

[11] Fuentes-Fernández, L., Vallecillo-Moreno, A.: An Introduction to UML Profiles. UML and Model Engineering **V**(2) (April 2004)

[12] Kühne, T., Schreiber, D.: Can programming be liberated from the two-level style: multi-level programming with DeepJava. In: OOPSLA '07, New York, USA, ACM (October 2007) 229–244

Contents

1 Introduction **3**

2 Motivation **4**
2.1 The MARTE Time Profile 4
2.2 Multi-level aspects in the Time Profile 5
2.3 Applying the Time Profile 6
2.4 Limitation 7

3 Multi-level modeling with deep instantiation **8**

4 Our proposal **9**
4.1 Domain model specification 10
4.2 Automated profile-based implementation of the domain model 10
4.3 Applying the generated profile 11

5 Discussions on our approach **12**
5.1 Design flow comparison 12
5.2 Possible extensions 13

6 Related work **14**

7 Conclusion **14**

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399