Multilevel Modeling Paradigm in Profile Definition

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

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Multi-level Modeling Paradigm in Profile Definition

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

N° 6525

April 2008

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6525--FR+ENG

Multi-level Modeling Paradigm in Profile Definition

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

Thème COM — Systèmes communicants
Projets Aoste

Rapport de recherche n° 6525 — April 2008 — 17 pages

Abstract: Building a UML profile entails defining concepts required to cover a specific domain, and then, using stereotypes to map domain concepts onto UML meta-classes. Capture of domain concepts with an object-oriented language (like UML) may be inappropriate, and may impede the mapping, where more than two modeling levels are required. Use of only classes and objects may introduce *accidental complexity* into the domain model if other modeling levels (*e.g.,* meta-type level) are necessary. In such situations, a multi-level paradigm with *deep characterization* and *deep instantiation* is recommended to reduce complexity. However, this paradigm deserves to be further explored, and its value for definition of UML profiles assessed. We therefore propose a solution to put in practice the multi-level paradigm within a standard UML 2.x tool. Our solution involves a semi-automatic process that transforms a model annotated with multi-level characteristics into a profile-based implementation. Such automation lessens the gap between domain model and implementation and ensures consistency. As an example, we have taken an excerpt from the MARTE time profile. We then describe the new design opportunities inherent in our process and show how this process facilitates both domain specification and profile definition.

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

This report as also been published as a CEA research report: XXXXX

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

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

La modélisation multi-niveaux pour la définition de profils UML

Résumé : Pour le développement de systèmes logiciels, on hésite souvent entre l'utilisation d'un langage métier, très adapté au domaine mais qui en général bénéficie d'outils faiblement supportés, ou d'un langage général qui est moins adapté mais qui au contraire bénéficie d'un bon support et pour lequel on peut trouver de nombreux ingénieurs expérimentés qui seront immédiatement opérationnels. Une solution intermédiaire consiste à définir un profil UML spécialisé pour le domaine visé. La construction d'un profil spécialisé passe par la définition d'un modèle domaine puis par implantation du domaine en associant à chaque concept métier une méta-classe UML. Comme UML est un langage objet qui utilise deux niveaux de modélisation (objet et classe) et que le domaine peut intrinséquement s'appuyer sur plus de niveaux (utilisation de méta-types, par exemple), l'implantation peut s'avérer difficile voire erronée. Lorsqu'un modèle requiert plusieurs niveaux, la *deep instantiation* est un mécanisme qui réduit la distance entre la spécification et l'implantation. Ce mécanisme n'a pas encore été étudié dans le cadre de la construction de profils UML. Nous proposons donc une extension de ce mécanisme adaptée à la définition et profils ainsi qu'une implantation de la *deep instantiation* et de l'extension proposée dans un environnement UML 2.x standard. Dans cette démarche, un modèle domaine annoté est progressivement transformé dans un profil. Nous pensons que l'automatisation de ce procédé réduit la distance entre le modèle domaine et l'implantation et assure la cohérence.

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

1 Introduction

The development of software systems usually starts with a technology-independent description that can be used as a communication interface between the domain experts and the implementation team. Domain-specific languages (DSLs), which directly capture the concepts of a specific domain, are adequate for this purpose.

Building tools for domain-specific languages can, however, be quite expensive. The target community being generally small, there is little economic incentive for tool vendors to build such tools. The result has 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. An alternative is to use standard profile mechanisms that incrementally build on the general-purpose Unified Modeling Language (UML) [1] to define only the new concepts required to represent domain-specific elements. This dramatically reduces the cost of building tools by reusing existing ones and limits the investment required for the learning process.

Best practice for defining a UML profile [2] calls for first detailing the concepts required for a specific domain by building a so-called *domain model*. The domain model is then mapped onto the UML metamodel by defining stereotypes that extends existing UML concepts (meta classes) 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. 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 abandon to the traditional two levels and promote a multi-level modeling paradigm that combines *deep characterization* with *deep instantiation*.

Use of a multi-level modeling paradigm is particularly relevant to defining profiles. Profiles are cross-level mechanisms between the meta-model and the model levels and mix multi-level concepts: model libraries elements, meta-classes representing descriptors (*e.g.,* Classifier) and meta-classes 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 removed 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.

The contribution of our work is twofold. We propose a UML profile for domain specification that offers multi-level annotations and a practical solution for implementing the deep instantiation mechanism within a standard UML 2.x tool. This entails a two-step process. First, the profile is used to specify a domain model, then the model is semi-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 a profile for creating domain model, with the multi-level annotations, and we elaborate on the process to transform this domain model into the profile-based implementation (Section 4). We conclude by outlining the differences between a process that leverages the multi-level modeling paradigm and the approach followed by MARTE designers (Section 5).

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.

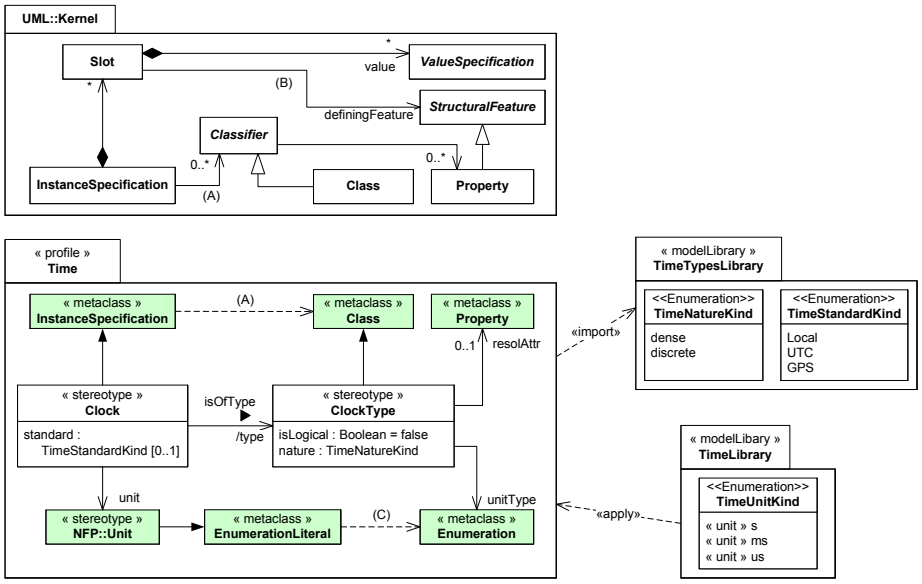

Figure 1: Excerpt from the MARTE Time Profile

All together, the two stereotypes (ClockType and Clock) provide for mechanisms to create new clocks and to put together (within a clock type) commonalities between related clocks. In this example, at least two modeling levels have been mixed together in one.

The first, or meta-model 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 association isOfType between the stereotypes Clock and ClockType. 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. This choice 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). Ultimately, the resolution value is given in a slot of an instance. 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 meta-model, 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.

2.2 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 the occurrences of an event. For this 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.

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

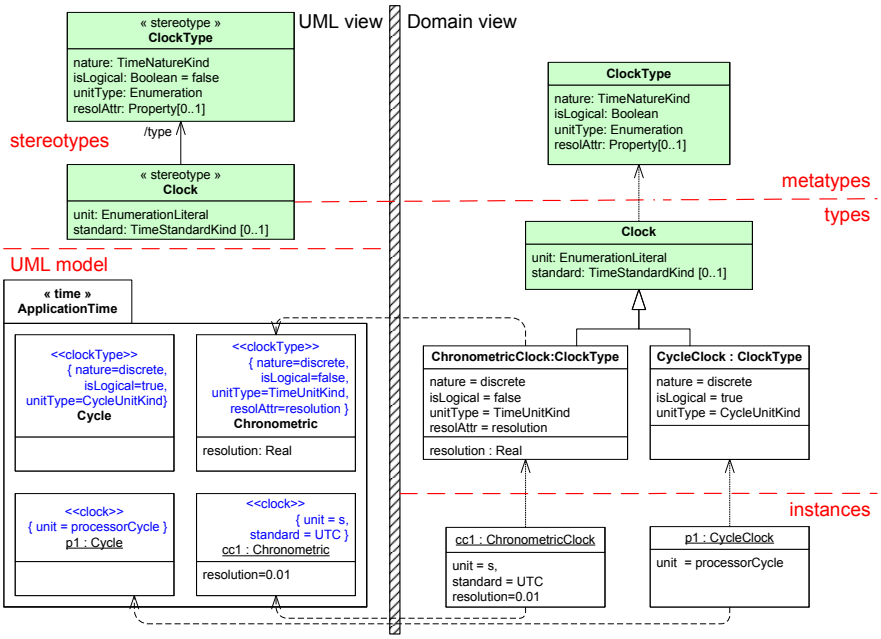

Figure 2: Examples of clocks and clock types

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 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.

2.3 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 [9, 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 one are regular attributes and fields of potency zero are regular slots. Fields of potency higher than one are meta-attributes, ... 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 two of the authors were part of the ProMARTE consortium in charge of defining the profile MARTE. Their solution uses artificial means to represent the "instance of" relationships in a model and the boundary between classifiers and instances becomes difficult to identify. Deep instantiation, on the other hand, makes multiple levels explicit and offers new design opportunities. In this approach, both stereotypes Clock and ClockType can be unified into a single concept. Instantiation levels are identified by potency independently from the underlying UML implementation details.

Figure 3 shows the intuitive process followed to merge the levels. 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. Particular attention must be paid to definition of units. As already mentioned above, units are described at two levels though they represent the same concept. Unification is possible provided we know how the type evolves in the instantiation chain (Enumeration becoming EnumerationLiteral).

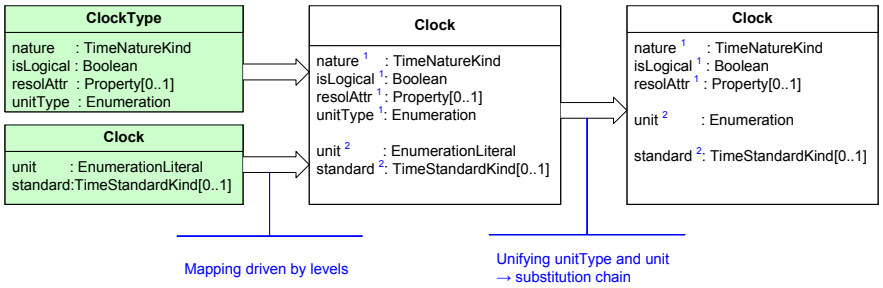

Figure 3: Clock definition with potency information

The final description (righthand side) is very concise and has almost 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. To have exactly the same expressiveness for the unit, additional information is required, *i.e.,* the way in which the type of this field evolves through the successive instantiations. In the first instantiation, the unit is a user-defined enumeration (*e.g.,* TimeUnitKind). In the second, it is an enumeration literal (*e.g.,* s).

Although at the first glance, it may seem that we only need the final type of a field (*i.e.,* EnumerationLiteral), having its intermediate type(s) provides for a way to ensure that every clock from a given clock type must use the same *set of units* (unit type). To maintain this chain of substitution we need to extend the original deep characterization mechanism. Such a situation is specific to profile mechanisms, which involve use of metaclasses in addition to regular types (classes and data types). In the next section, we introduce the concept of *deep substitution* as an extension of the deep characterization mechanism to take into account

this specificity. We then 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 use of 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, we have used the Time profile as an example for step-by-step illustration.

4.1 Domain model specification

We have defined a UML profile (called DomainSpecification profile) for domain specification. The stereotypes of this profile are used to annotate a UML model with information required for multi-level modeling. This enables to declare models in a similar way as presented in Figure 3, while accounting for the specificity of the profiling mechanism.

The Profile DomainSpecification

The profile DomainSpecification consists of two stereotypes (see Figure 4).

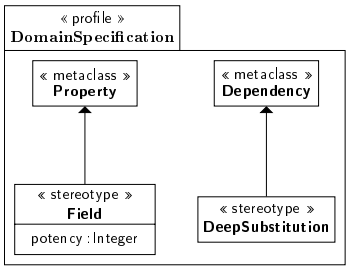

Figure 4: Domain specification profile

The stereotype Field carries potency information. The second stereotype, DeepSubstitution, extends the metaclass Dependency. It identifies dependency relationships between metaclasses to make explicit how substitution of metaclasses should take place at each instantiation level.

Domain Specification of Time

We use the profile DomainSpecification to specify the time-related concepts as concisely as the specification given in Figure 3.

We start with a Class model and apply our stereotypes to specify potency and the relevant substitutions (Figure 5).

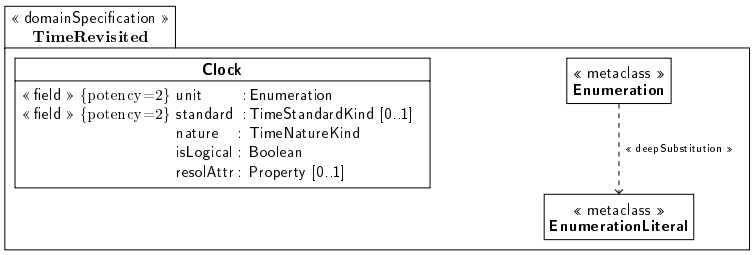

Figure 5: Time concepts with multi-level modeling

The deepSubstitution dependency calls for an Enumeration at the first instantiation level to become an EnumerationLiteral at the next instantiation level. Thus, at the first level, the unit is specified as an enumeration (set of acceptable units). At the second level, the unit must be one of the elements in the set, *i.e.,* an enumeration literal.

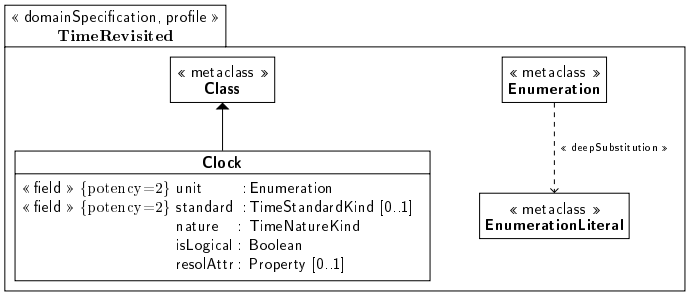

Figure 6: Choice of extended metaclasses.

Potency is optional. A property without potency is considered a *regular* attribute and is equivalent to a potency of one. Specification of *deep substitution* is only necessary for fields whose type is a metaclass. If so, the user may use this stereotype to describe the evolution of the type of the field through successive instantiations. If no type substitution is required, no deep substitution chain need be specified, such as for the property resolAttr. In our example, there is only one deep substitution (Enumeration becoming EnumerationLiteral).

To generate a UML profile, we need to declare the UML concept (metaclass) used to support our domain concept. The next step is then to transform our model into an actual profile and to choose the metaclass to extend for each domain concept (Figure 6).

This profile is simply an intermediate model. The following subsection describes the third step in our process, which is to automatically derive a final profile from the intermediate model.

4.2 Automated profile-based implementation of the domain model

In this last step, we use the domain specification as an artifact to build an equivalent final profile-based implementation. The result is a profile with enough stereotypes to describe each instantiation level for each concept. Consequently, starting from Figure 6, we automatically derive the profile illustrated in Figure 7. This final profile is taken by the end-user to declare concepts defined in the domain specification.

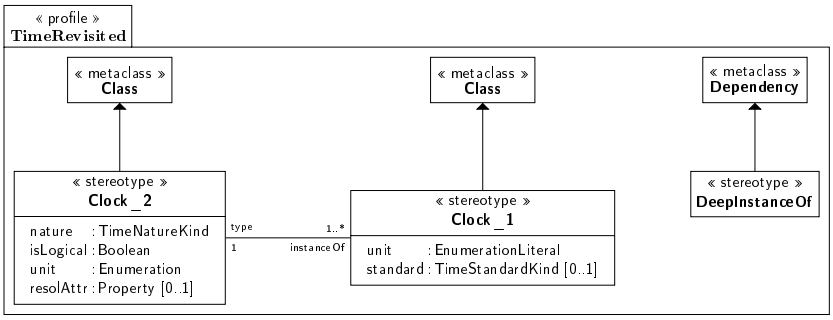

Figure 7: Time profile generated from the domain specification

A straightforward algorithm is used for transformation. Each stereotype gives rise to as many stereotypes as instantiation levels. The name of each stereotype is suffixed by an integer that reveals the level of instantiation. In our example, there are two levels (potency=2 and potency=1), so we derive two stereotypes (Clock_2, Clock_1).

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). When the type of the field is connected to another type by a «deepSubstitution» dependency, an attribute is added to each stereotype with regards to the sequence of substitutions. For instance, if chain length is two, two attributes are created. The type of the attribute is then given by following the chain of substitutions.

On Figure 7, 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). An association between the stereotypes of two consecutive levels is added to identify a deep type, potentially with, several of its deep instances. The stereotype DeepInstanceOf unambiguously relates 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 8).

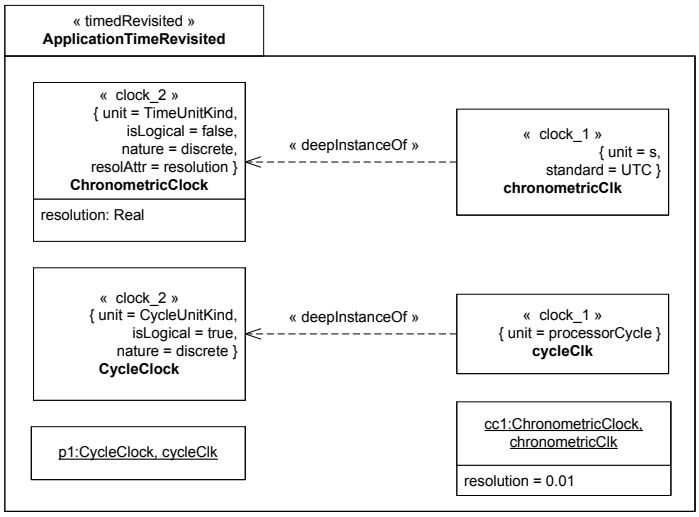

Figure 8: Clock definition with the generated profile.

Modeling of Cycle clock entails two new classes, one stereotyped by «clock_2» and the other by «clock_1». The dependency relationship stereotyped «deepInstanceOf» avoids mixing Cycle clocks with Chronometric clocks. For instance, cycleClk depends on CycleClock, not on 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, is replacement of the OCL constraint with an explicit relationship between elements contributing to the definition of a same concept.

How instances of the concepts are represented constitutes another difference. In the former approach, this was achieved by instance specifications of the classifiers ChronometricClock and CycleClock, stereotyped by «clockType». The instance specifications simultaneously carry information about the slots of this classifier, but also provide information as values relating to properties defined by the stereotype Clock. With the generated profile, we must express the fact that an instance of a clock has two classifiers, each of which carries properties related to one level.

5 Discussions of our approach

The following 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 compares the resulting profiles for both approaches.

5.1 Design flow comparison

Figure 9 shows comparison of the two process workflows, with their different outputs, from conceptual domain definition to profile creation.

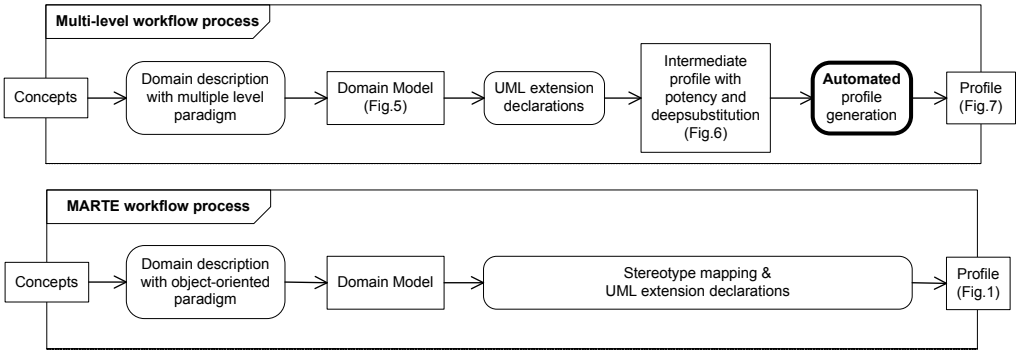

Figure 9: Design activity flow comparison

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.

Use of a multi-level modeling paradigm to build the domain model leads to a semi-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. The only design decision that remains manual is mapping of stereotypes to the relevant UML meta-classes. This is required to be compatible with legacy profile practices but can also be avoided by choosing, as base class, the default meta-class Class. The process thus becomes fully automatic and design activity can focus on the domain specification.

5.2 Profile Comparison

We have combined the deep instantiation mechanism with a deep substitution mechanism that specifies type field substitutions throughout its instantiation cycle. This is a powerful

tool for reducing the number of properties relating to a concept, but also, for controlling allocation of information between the stereotype and the stereotyped element.

This approach is used to merge the property unit of the stereotype Clock and the property unitType belonging to ClockType into a single field unit. The same strategy could have been applied to create a single field, called resolution, of type DataType, with a potency of two, instead of using resolAttr from ClockType to reference a property on the end-user model (*e.g.,* the property resolution from the ChronometricClock class). Such modification would require a substitution chain from DataType to ValueSpecification. In that case, the first instantiation level of the clock would stand for the declaration of resolution type (*e.g.,* Real) and the second would give access to the value. Fig. 10 depicts the new definition of the clocks. All clock-related information resides at the stereotype level, without external input.

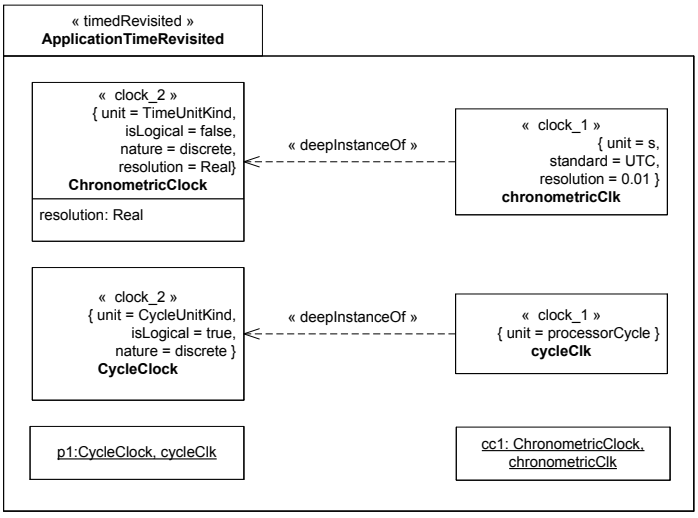

Figure 10: Clock definition using merged resolution field

The decision to locate information at the stereotype level or with the stereotyped element is clearly a delicate one and there are no systematic criteria for making it. This design choice closely depends on the purpose of the domain and its expected profile usage. A stereotype may be used to identify a feature from the base class for subsequent model analyses or model transformations. This is the scheme proposed by Thomas et al. [10] to depict software platform domain and facilitate their transformations. This modeling requirement calls for properties of stereotypes to maintain a link with base class features.

6 Related work

The core motivation of our research is to facilitate building of UML-based domain-specific language. The standard mechanism is profile designs. Despite the ever increasing number of profiles being built in many domains, there is a little published literature available to support the process.

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 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 [9]. 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 research, as well as our proposal, are somewhat different. We assess values of deep instantiation mechanisms in the context of UML profile definitions, then demonstrate that the current UML standard already includes mechanisms for accessing the realm of multi-level modeling.

Difficulties related to declaring domain types and subsequent declarations at implementation level have already been partially addressed by the AUTOSAR (AUTomotive Open System ARchitecture) project. A modeling framework has been defined to build templateable metamodels [13]. This framework uses a UML profile to define a set of common patterns occurring in (meta)modeling (*e.g.,* types, prototypes, instances) and makes the duality of type versus instance explicit. However, the framework does not cover the creation of profiles or models complying with the domain model.

7 Conclusion

This paper presents a process for building UML-based domain-specific languages 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 with multi-level information: concepts are classes, and properties are fields with a potency information that indicates their intended instantiation level. This profile also contains elements to declare the meta-class substitutions to control evolution of types throughout instantiation cycle. 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 the stereotypes required to represent the different instantiation levels of a concept. Application of this profile thus enables deep instantiation and modeling of elements complying with the domain model specification.

We have illustrated our approach using 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 simplification. It facilitates the domain specification by limiting implementation considerations. It makes the domain description more concise and make the modeling levels explicit. The resulting domain model includes in a single class all the information related to a given concept, which was previously scattered over several classes.

We are currently automating our proposal in an Eclipse environment. 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.

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)

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

[10] Thomas, F., Delatour, J., Terrier, F., Gérard, S.: Toward a Framework for Explicit Platform Based Transformations . In: 11th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC - 2008), Orlando (FL) (May 2008)

[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

[13] AUTOSAR GbR: Template UML Profile and Modeling Guide. (2008) Ver. 2.1.1, `http://www.autosar.org/download/AUTOSAR_TemplateModelingGuide.pdf`.

Contents

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