Inductive UML

Franck Barbier and Eric Cariou

University of Pau, BP 1155, Avenue de l'université,
64013 Pau CEDEX, France
{franck.barbier, eric.cariou}@univ-pau.fr

Abstract. The increasing importance of metamodeling calls for metamodels that are free of ambiguities, contradictions and redundancies. This is specifically the case for the core of UML (Infrastructure). This paper proposes to rewrite a part of this core, the *Class* and *Property* metaclasses especially. To avoid infinite regression, the notion of meta-circularity is used. This rewriting is done by means of inductive types in constructive logic. The proposed specification is proven correct using the Coq automated prover. Proven lemmas and theorems about a "metaness" relationship are proposed.

Keywords: Unified Modeling Language, Metamodeling, Constructive logic.

1 Introduction

The *Unified Modeling Language* (UML) [1] is historically based on metamodeling [2-4]. Because models are "instances of" or "conform to" metamodels, they are tinged with errors when metamodels they come from have anomalies. This phenomenon is even more important when metamodels are implemented in operational environments like the *Eclipse Modeling Framework* (EMF) [5]. Model transformations occur at design time or there is a possibility of having executable models at runtime. In the latter case, persistent metamodels act as a reflection mechanism (metadata) and Java may act as an action language to manage models at runtime.

UML has chosen a four-layer metamodel hierarchy with an upper level named M3. This level is a set of booting notions called "Infrastructure" [6] that reuses elements from the *InfrastructureLibrary* and the *Meta Object Facility* (MOF) [7]. UML promotes the "anything must be an instance of something" adage. In this scope, the key *Class* and *Property* metaclasses at M3 (Fig. 1) must then be instances of something at M4. However, introducing a M4 layer leads to the introduction of a M5 layer that leads to… Avoiding such an infinite regression requires an appropriate specification named meta-circularity [8]; the *Class* and *Property* metaclasses must then be formally specified such that:

- They are instances of something without the need of extra metamodeling layers. This in particular supposes a clear (explicit) characterization of *Instantiation*;
- They are generative. All of the other highly useful core metaclasses like *Object*, *Type*, *Association*, *Generalization*… even some missing like the *Composition*

metaclass (black diamond in the UML notation)[1], may be defined through appropriate instantiation protocols. This approach called "inductive UML" is such that UML can be recursively defined.

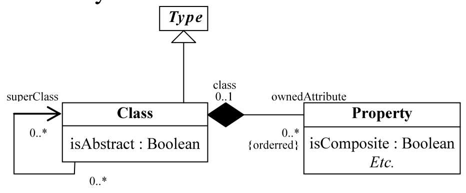

Fig. 1. The "very core" of UML with *Class* and *Property* as booting notions.

Numerous research works [8-12] (see also *Related works* section) are attempts to better clarify the semantics of metamodels in the UML universe: MOF, Infrastructure, Superstructure and any possible extension. Out of these, metamodel re-formalizations often rely on "theories" (*e.g.*, non-classical logics) beyond the set theory.

In this paper, metaclasses are rewritten in the form of inductive types coming from the constructive logic supported by the Coq automated prover [13] as follows:

```
Inductive X : Type := (* Type is a predefined Coq sort
among Type, Set and Prop *)
| God (* First constructor *)
| cons : X -> X. (* Second constructor *)
```

God and *cons* are the names of the two chosen constructors for the *X* type along with their signatures. Common functions may be defined as follows:

```
Definition father(x : X) : X := match x with
| cons source => source (* father x is equal to source
when x has been constructed by means of the 2nd
constructor, i.e., cons source *)
| _ => God (* Result is God for the remaining
constructor(s); underscore sign means "any" in Coq *)
end.
```

Proofs are based on "tactics" to converge towards a given goal from initial and intermediately computed hypotheses.

So, in this paper, we specify and prove the correctness of a metamodeling framework based on Coq. For that, Kühne's metamodeling framework [2-3] is the main stream of inspiration. In his categorization, Kühne proposes in [3, pp. 377-378] a general-purpose mathematical relationship called "metaness" having the following characteristics: acyclicity, antitransitivity and level-respecting.

To make explicit a proven metamodeling framework, we structure this paper as follows: Section 2 is a reminder about the current UML design principles and organization. We specify *Class* and *Property* in Coq and how to use them by introducing metanavigations and by showing how to instantiate any other metaclass. Accordingly, the *Instantiation* relationship is formalized. Section 3 is the specification

[1] This kind of relationship is intensively used at the M3 level without any formal semantics (see for instance Fig. 1).

of Kühne's metaness along with short proofs. Section 4 is about related work while Section 5 draws some conclusions and evokes some perspectives.

2 UML core organization as dependent inductive types

The model in Fig. 1 means:

- A *Class* instance is composed (black diamond) of either zero or many *Property* instances (*ownedAttribute* role). A given *Property* instance belongs (or not) to at most one *Class* instance (*class* role); unsharing applies, *i.e.*, a given *Property* object cannot belong to distinct *Class* objects;
- A *Class* instance is linked to either zero or many *Class* instances having the *superClass* role[2]. The reverse navigation means that a given *Class* has (or not) direct descendant classes; this metarelationship embodies *Generalization* links at the immediately lower metamodeling level;
- *Class* inherits from *Type*;
- Classes are either abstract (in italics) or they are not. For instance, the *Type* metaclass is abstract. Moreover, the *Class* metaclass has a Boolean attribute named *isAbstract*. This means that any instance of *Class* owns this attribute with a value among *true* or *false*. So, *Type* is an instance of *Class*[3] with value *true* for this attribute. In terms of instantiation, one thus cannot construct a new metaclass[4] as direct instance of *Type*.

For conciseness, other key metaclasses (*e.g.*, *NamedElement*), metaattributes (*e.g.*, the *name* attribute inherited by *Class* from *NamedElement*) are ignored. Moreover, "hidden" features of the model in Fig. 1 are:

- *Class* is an instance of itself. In the four-layer metamodel hierarchy of UML (M3 to M0), a *Class* element at the M2 level is an instance of a *Class* element at the M3 level. There are no reasons to distinguish between *M2::Class* and *M3::Class*. Conceptually, they are the same (same set of features especially). Accordingly, we consider the existence of an *Instantiation* link from *Class* to itself.
- The *Type* and *Property* elements are instances of *Class*;
- *isAbstract* in *Class* at M3 is an instance of *Property* at the immediately upper level with *isComposite = true*. So, *isAbstract* is semantically equivalent to a composition relationship from *Class* to *Boolean* with the *1..1* cardinality and the *isAbstract* role both being next to Boolean.
- *isComposite* in *Property* is an instance of *Property* with *isComposite = true*;
- The *Composition* link from *Class* to *Property* is an instance of *Property* with *isComposite = true* (for brevity, some original attributes of *Property* are omitted);
- The *Association* link from *Class* to *Class* (*superClass* role) is an instance of *Property* with *isComposite = false*;

[2] This association materializes direct inheritance, *i.e.*, it does not represent all of the super classes of a class (transitive closure).

[3] This *Instantiation* link does not appear in Fig. 1. In common practice, links that cross metamodeling layers are omitted.

[4] While *Type* belongs to the M3 level, such a hypothetical metaclass would belong to the M2 level.

– Finally, the *Generalization* link (*i.e.*, inheritance) from *Class* to *Type* is an instance of the *Association* link from *Class* to *Class*.

2.1 Inductive definition of *Class* and *Property*

In this section, *Class* and *Property* are introduced as Coq types while *BBoolean*, *CClass*, *PProperty* and *TType* are UML concepts (*i.e.*, Coq constants). It is also shown that *Class*, *Property* and *NonAbstractClass*[5] are mutually dependent types.

```
Inductive Class : Type :=
    BBoolean | (* UML Boolean type *)
    CClass |
    PProperty |
    instantiate : NonAbstractClass -> Property ->
Property -> Class |
    inheritsFrom : Property -> Property -> Property ->
Class
with Property : Type :=
    Null | (* Null is introduced in [7, p. 11] *)
    set_isAbstract : Property |
    set_isComposite : Property |
    set_ownedAttribute : string -> Class -> nat -> nat
-> Property -> Property | (* Expected order: attribute
name, attribute type, lower bound, upper bound,
isComposite or not *)
    set_superClass : Class -> Property (* Inheritance
*)
with NonAbstractClass : Type :=
   instantiate' : Class -> NonAbstractClass.
```

2.2 Metanavigations

The definition of metanavigations is straightforward. For example, *ownedAttribute* in Fig. 1 is specified as an ordered list of *Property* objects:

```
Definition ownedAttribute(c : Class) : list Property :=
match c with
  | CClass => cons (set_ownedAttribute isAbstract_label
BBoolean 1 1 set_isComposite) nil
  | PProperty => cons (set_ownedAttribute
isComposite_label BBoolean 1 1 set_isComposite) nil
  … (* other constructors here*)
  | _ => nil (* remaining cases *)
```

[5] This type is introduced for preventing abstract classes are to be instantiated.

```
end.
```

So, by construction, computing the expression *ownedAttribute CClass* leads to a one-element list: its *isAbstract* attribute (see Fig. 1).

2.3 Constructing new metaclasses

The generative nature of the above specification allows the creation of other core concepts through different protocols. For example, instantiating a Coq *Class* object (that is equivalent to a UML *CClass* object):

```
Definition Object : Class := instantiate (instantiate'
CClass) Null Null. (* [7, p. 15] *)
```

Here, the first *Null* occurrence means that *Object* coming from the UML kernel is not abstract while the second means that it has no "owned attribute" (note that simplified *instantiate* methods may be easily provided to avoid using *Null*).

2.4 A formal version of the UML *«instanceOf»* relationship

To solve the problem of assigning a mother class to *CClass* (meta-circularity), we specify the recursive *class* function over the *Class* inductive type:

```
Fixpoint class(c : Class) : Class := match c with
| instantiate (instantiate' c') _ _ => c'
| inheritsFrom (set_superClass super) _ _  => class
super
| _ => CClass (* BBoolean => CClass | CClass => CClass
| PProperty => CClass *)
end.
```

Consequently, the UML *«instanceOf»* relationship can be easily derived from the *class* above function as follows:

```
Inductive instanceOf(c' : Class) : Class -> Prop := (*
e.g., instanceOf CClass Object *)
def : forall c, c' = class c -> instanceOf c' c.
```

In Coq, predicates using recursive constructions (*def* constructor above) may also be inductively defined.

3 Proven metamodel infrastructure for UML

3.1 Metaness

Kühne lays down the principle of composition of the *class* function for expressing metaness. Metaness is viewed "as a two-level detachment of the original".

In Coq, we pose the possibility of recursively computing the meta$_i$class of any UML element *e* for any natural number *i* with *meta$_0$class e = e* and *meta$_1$class e = class e*. The *i* index materializes levels in metamodeling.

```
Fixpoint metaness(n : nat) (c : Class) : Class := match
n with
| 0 => c
| S m => class (metaness m c) (* S m is the successor
of m for natural numbers in Coq *)
end.
```

So, *metaness 0 c* is the *c* entity itself while *metaness 1 c* is the direct class *c*. *metaness 2 c* is the class of the class of *c*, namely the metaclass of *c*, etc. An interesting lemma to be proven is, when *n* is not equal to *0*, *c = metaness n c* is only possible when *c = CClass*:

```
Lemma Metaness_majorant : forall c : Class, forall n :
nat, n <> 0 -> c = metaness n c -> c = CClass.
```

3.2 Metaness acyclicity, antitransivity and level-respecting

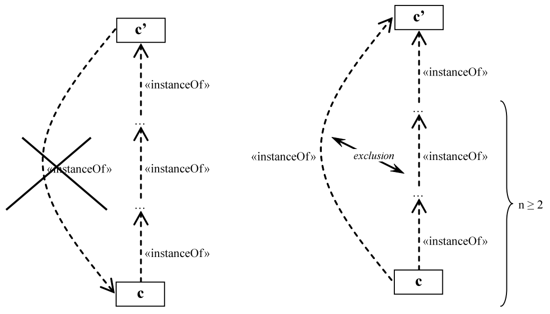

Fig. 2. Metaness acyclicity (left hand side) and antitransitivity (right hand side).

The proof of metaness acyclicity is based on the following Coq theorem:

```
Theorem Metaness_acyclicity : forall c c' : Class,
forall n : nat, c <> CClass -> c' = metaness n c -> c
<> class c'.
```

This rule is illustrated through Fig. 2. A proof by contradiction is necessary to justify this theorem (for conciseness, we hide Coq tactics). We imagine the absurd consequence that *c = class c'*. If so, from the initial assumption *c' = metaness n c* (*c* <> *CClass*), we are able to write:

```
class c' = class (metaness n c)
c = metaness (S n) c (* absurd hypothesis is used *)
```

From the inductive specification of the *nat* type in Coq, we know that *S n* <> *0*. From the *Metaness_majorant* lemma, we conclude that *c = CClass*. This result is in contradiction with our initial hypothesis part: *c* <> *CClass*. So, *c = class c'* is absurd.

The final conclusion is therefore: *c <> class c'*. In other words, this paper's specification of metaness is acyclic as advocated by Kühne in [3].

The proof of metaness antitransitivity (Fig. 2) is based on the following Coq theorem:

```
Theorem Metaness_anti_transitivity : forall c c' :
Class, forall n : nat, c <> CClass /\ n >= 2 -> c' =
metaness n c -> c' = class c -> c' = CClass.
```

The proof of level-respecting is based on the following Coq theorem:

```
Theorem Level_respecting : forall n m : nat, (exists c
: Class, exists c' : Class, c <> CClass /\ c' =
metaness n c /\ c' = metaness m c) -> n = m.
```

4 Related work

There are two general-purpose categories of research works that stress the weakness of UML. In [9] for instance, the authors use conceptual graphs to re-formalize the *Class* (renamed *Node* in the proposed formalization), *Association* (renamed *Link*) and *Specialize*[6] (renamed *super*) metaelements. An interesting point in this contribution is the introduction in the foreground of the *Instantiation* relationship through a *meta* predicate. The paper offers conceptual graphs as a set of first order predicates including the specification of meta-circularity as follows:

```
[NODE:NODE]->(meta)->[NODE:NODE] (* [t:i] means i of
type t *)
```

This pioneering work also introduces the *sem* predicate (*instanceOf* inductive predicate above) as the counterpart of the *meta* predicate. However, no proofs are offered to show that these two constructions are mutually consistent, even though it is written: "The *sem* relation is derived from the *meta* relation." The *Instantiation* relationship is recursively defined without termination capabilities:

```
[LINK:meta]->(meta)->[NODE:LINK]
```

One observes that this specification is not generative in the sense that there is no bootstrapping: *meta* is an instance of *Association* which is an instance of *Node*.

Another paper inspired by a graph theory is [10]. Authors propose the introduction of a formal semantics that in particular crosses over all diagram types. More ambitious works [8] [11-12] consider the pure invention of a metamodeling framework (even theory) and/or a dedicated language (*e.g.*, MML in [8]). For example, Paige *et al.* in [12] benefit from using another automated prover (PVS). They demonstrate how models may be accordingly checked in an Eiffel-like fashion: invariants, pre-conditions and post-conditions are kept in PVS to limit Eiffel as a formal language only.

[6] This metaclass has been removed from the last versions of MOF and UML.

5 Conclusions and perspectives

From version 1.1 in 1997, the overall UML metamodel has undergone many changes. However, the current formalization (metamodels expressed in the Entity/Relationship paradigm along with OCL constraints, *i.e.*, well-formed rules) has not gone beyond the set theory. Based on this style, precise metamodeling does not preclude from having rules that contradict each other, that create overlapping or that are silent on hot topics. The latter issue may be illustrated by the absence of a formal semantics of the *Composition* relationship in UML.

This paper's research seeks to be faithful to the original UML spirit. As much as possible, we intend to avoid any restructuring of the existing dependencies between metaconcepts. However, as shown in Section 4, new constructs seem useful to move from precise metamodeling to formal metamodeling. In this scope, constructive logic and Coq are powerful helpers.

References

1. OMG Unified Modeling Language™ , Superstructure, Version 2.3 (May 2010)
2. Atkinson, C. and Kühne T.: Model-Driven Development: A Metamodeling Foundation. IEEE Software 20(5), pp. 5-22 (2002)
3. Kühne, T.: Matters of (Meta-) modeling. Software and Systems Modeling 5(4), pp. 369-385 (2006)
4. France, R. and Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. In: The ICSE 2007 Future of Software Engineering workshop (Minneapolis, USA) (2007)
5. Steinberg, D., Budinsky, F., Paternostro M. and Merks, E. EMF - Eclipse Modeling Framework, Second Edition. Addison-Wesley (2008)
6. OMG Unified Modeling Language™ , Infrastructure, Version 2.3 (May 2010)
7. Meta Object Facility (MOF) Core Specification, Version 2.0 (January 2006)
8. Clark, T., Evans A. and Kent, S.: The Meta-Modeling Language Calculus: Foundation Semantics for UML. In: The 4th International Conference on Fundamental Approaches to Software Engineering, pp. 17-31 (Genova, Italy) (2001)
9. Bézivin J. and Gerbé, O.: Towards a Precise Definition of the OMG/MDA Framework. In: Automated Software Engineering, pp. 273-280 (San Diego, USA) (2001)
10. Kuske, S., Gogolla, M., Kreowski, H.-J. and Ziemann, P.: Towards an integrated graph-based semantics for UML. Software and Systems Modeling 8(3), pp. 403-422 (2009)
11. Jackson, E. and Sztipanovits, J.: Formalizing the structural semantics of domain-specific modeling languages. Software and Systems Modeling 8(4), pp. 451-478 (2009)
12. Paige, R., Brooke, P. and Ostroff, J.: Metamodel-based model conformance and multiview consistency checking. ACM Transactions on Software Engineering and Methodology 16(3) (2007)
13. Bertot Y. and Castéran, P.: Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions. Springer (2004)