A Framework to Formalise the MDE Foundations

Xavier Thirioux, Benoît Combemale, Xavier Crégut, Pierre-Loïc Garoche

A Framework to Formalise the MDE Foundations

Xavier Thirioux, Benoît Combemale, Xavier Crégut, and Pierre-Loïc Garoche

Institut de Recherche en Informatique de Toulouse (CNRS UMR 5505), Toulouse, France
first_name.last_name@enseeiht.fr

Abstract. Domain-Specific Language (DSL) are getting more and more popular and are being used in critical systems like aerospace and car industries. Methods for simulating and validating DSL models are now necessary in order to make the new software generation more reliable and less costly.
Developing analysis tools for DSL requires the definition of models semantics. In this paper, we propose a framework to give a formal foundation of the Model-Driven Engineering (MDE) approach. We separate the usually common notions of models and modelling languages associating to each of them a different goal. In order to prove the consistency of our proposal we express a subset of EMOF, its static semantics and validate its meta-circularity.

1 Introduction

After the object-oriented approach of the Eighties and its core principle “everything is an object”, nowadays software engineering follows the Model-Driven Engineering (MDE) approach with its core principle “everything is a model” [1]. The UML (*Unified Modelling Language*) consensus [2,3] was decisive in this transition towards the model-driven techniques. Once the key concept of *metamodel* has been accepted as a model description language, many metamodels emerged. Each with its own specificities according to the corresponding domain (software development, process engineering, data warehouse. . .). For instance, aside UML, the OMG has defined several metamodel such as SPEM (*Software Process Engineering Metamodel*) [4] for process modelling.

In order to prevent this wide variety of metamodels to emerge in an independent and incompatible manner, there was an urgent need to define a general framework for their description. Among the proposals, the OMG introduced a language to define metamodels, which takes itself the form of a model: the MOF meta-metamodel (*Meta-Object Facility*) [5] was born. This meta-metamodel is self-descriptive, i.e. MOF may be defined using concepts defined in MOF (meta-circularity). Because MOF is self-descriptive, there is no need for further languages to define MOF. Based on these principles, the OMG organisation introduced the MDA® (*Model Driven Architecture*) [6] view of software modelling. This MDA approach is described in a pyramidal form as presented in Figure 1. The OMG calls M0 the real world, M1 its models, M2 the metamodels and M3 the self-defined meta-metamodel.

Unfortunately, MOF can only define the abstract syntax of the modelling language. It still lacks a concrete syntax and a definition of the semantics of this language. As users wish to have the same capabilities with DSL as the one they have with usual tools, there is an increasing demand for editing facilities, simulation, verification or

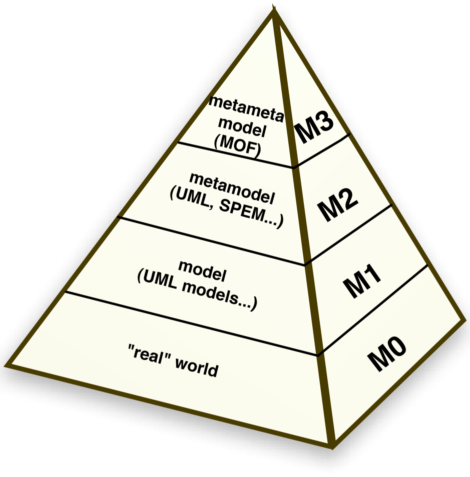

Fig. 1. OMG Pyramid

code generation features. To fulfil these wishes it is required to define the semantics of the DSL, that is defining a semantic mapping to a semantic domain [7]. Being able to formally define a semantics for a DSL and being able to check the consistency of such semantic mapping become a tremendous challenge that has to be tackled.

First, languages such as OCL [8] appeared and allowed to specify structural semantics through invariants defined over the abstract language. They are used to limit the number of models that conform to the DSL. OCL can also be used to specify behavioural semantics through the definition of pre- and post-conditions on operations. However, being side-effect free, OCL does not allow to alter the state of a model.

During the last years many works have addressed execution semantics in order to execute models. Different solutions have been proposed that may be classified into two main trends [9]. The first one consists in defining an operational semantics directly on the abstract syntax of the DSL using techniques like meta-programming languages (e.g., using Kermeta [10], xOCL [9]. . .), an action language such as UML actions [3, §11] or endogenous transformations as proposed in [11]. The second one consists in defining a mapping to an abstract syntax of another technical space (TS) [12] which is formally defined (i.e., for which a mapping to a semantic domain is defined). This translational semantics has been used in [13] where is proposed a semantic anchoring to a model of formal semantics built upon ASM (*Abstract State Machine* [14]), using the transformation language GReAT (*Graph Rewriting And Transformation language* [15]). Using ATL (*ATLAS Transformation Language* [16]), a direct mapping from the abstract syntax to ASM is proposed in [17] and a mapping to time Petri nets for process models validation is described in [18].

Operational semantics is generally more straightforward as it builds atop the DSL concepts and thus the domain concepts well known by the domain experts. On the contrary, mapping to another TS requires a deep knowledge of both the DSL domain and the targeted TS. Its main advantage is to be able to reuse tools available in that TS like simulators, model checkers, etc. Both approaches may be combined by (1) defining an operational semantics used as a reference and (2) defining several mappings to reuse tools of the target TS. One important point is to be able to assert that all the defined semantics are consistent. It may be achieved through a proof of equivalence (e.g. bissimulation [19]) but relies on a formal specification of the DSL semantics. Furthermore, this formal specification should be built atop a general framework providing all formal structures required for the proof of the equivalence. It should also be general enough to support the different views of MDE, for example those of the OMG.

In this context, our contribution consists in defining a formal framework in which both models and modelling language concepts could be mathematically defined. We first introduce some background and related works in Section 2. Then, we present our framework in Section 3. One important point is to clearly separate the instance level, the model, and the type level, the modelling language. Also, we define the (1) conformance criteria which states whether a model is valid according to a modelling language and (2) the promotion operation which consists in mapping a model into a modelling language that in turn is used to build models that conform to it. We then validate our framework in Section 4 by formalising one of the approach of the MDE, that is the pyramid proposed by the OMG in its view of the Model-Driven Architecture. In particular, we formalise a subset of Essential MOF (EMOF) [5]. In this paper, we only address static semantic issues. In the last section we give some concluding remarks and perspectives.

2 Background & Related works

Following the object-oriented (OO) paradigm, the MDE approach relies on two different levels:

- a first one allows the definition of concepts and relations between them (i.e. the language). It will drive the conception of particular systems. Such systems are defined according to these concepts and relations. In an OO view, concepts are represented by *Classes* and relations are represented by *References*.
- a second level allows the definition of systems. It is composed of a set of elements and a set of relations among these elements. In an OO view, these elements are called *Objects* and these relations are called *Links*.

The notion of *systems* used previously is used in a very general view in MDE. The modelling can be made at any abstraction level and in every domain, as soon as the modelling language is defined.

These two different *abstraction* levels are linked by an instantiation relation, also called typing relation in the opposite direction. In the OO view, an object O is an instance of a given class A. One can say that O is of type A. Every link between two objects is such that it exists, in the associated language, a reference between the two types, the two classes, of both considered objects.

After having precisely defined the concept of *model* and its *representationOf* relation with the modelled system [20,21,22], many works in the MDE community have precisely defined the different abstraction levels and relations between them, in the context of one model-driven approach. They allowed to well differentiate the instantiation relation, between one element of a model and one element of the modelling language, from the conformity relation, between one model and its modelling language [23,24,12].

However, these different conceptual frameworks did not define an associated formal coding. This purpose is addressed by the model type definition. It allows to formalise the contents of one model. For instance, the works proposed by Steel et al. in [25] have allowed to define a MOF-based model type-checker for in-place model transformation. Then, he allowed to evaluate the type substitutability for an input model of one transformation. The same issues arose in the ModelBus project. The purpose is to provide a bus of models so that MDE CASE tools, seen as service providers, may be integrated and may interoperate with each other through the exchange (input and output parameters) and sharing (in-out parameters) of models [26]. Parameters sent or received when a service is called have to be checked. They are thus typed according to a MOF conforming meta-model. It means that the model should conform to the metamodel given as its type. Further constraints have to be defined according to the restrictions of the tool implementing the service. For instance, it may not be able to handle more than a certain number of instances of some particular metaclass. Such constraints are defined through a specific metamodel and an ad hoc semantics. These works deal with the nature of model elements in a particular context, that is MOF in both cases. So they do not provide a general framework to define any type of model, and thus of modelling language.

Other works result from the graphs transformations community (e.g., AGG [27]). They are interested in graph representation of one model. The principle is to define a type graph from this model graph, which is its conceptual generalisation. Targeted on the structural graphs transformations, these works do not take into account the definition of semantic properties on the type graph to perform graph validation.

Finally, Jouault and Bézivin have proposed in [28] a formalisation of their KM3 language, a textual language to define metamodels including MOF. They introduce the notion of $ReferenceModel$ as a model that is a language to define models (Figure 2, left). A model must conform to its reference model. They also formalise the MDE concepts of $TerminalModel$, $MetaModel$ and $MetaMetaModel$ all being models following the usual view of the MDE community (Figure 2, right). Metamodel and meta-metamodels are reference models that differ depending on the level of the reference model they conform to. They define a model as a triple (G, ω, μ) where G is a graph, ω another model which is the reference model for the given model and μ is a function that maps elements of G to nodes of the ω graph. As this recursive definition does not allow finite representation of models, the authors propose to define an initial model in a reflexive manner. Such a model is a meta-metamodel and allows the definition of metamodels, that will in turn permit the definition of models (Figure 2). The core of the article is a formalisation in this framework of their meta-metamodel proposal KM3. This model is mainly composed of two classes: class and reference. Class is linked to reference which is itself linked back to class. It allows to model relationships between elements in metamodels.

In our opinion, this approach has some drawbacks. Seeing everything as a model does not allow cleanly to define a reflexive model. The initial formalisation of models is not well founded without assuming the existence of initial models. Introducing this concept of an initial model, as a reflexive model, differentiates it from the other non-

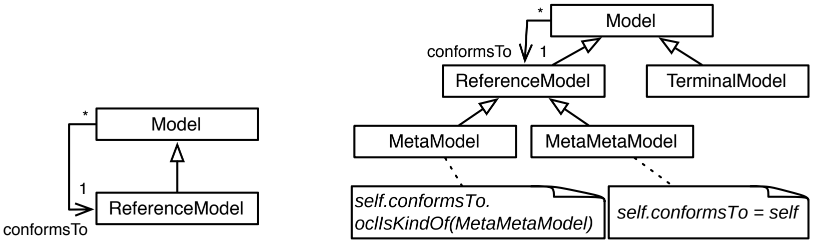

Fig. 2. Formalisation of MDE according to [28]

initial models and therefore breaks the view of "everything is a model" without any distinction.

Another problem is the definition of the μ function in non-initial models. The μ function maps edges and nodes of the model graph to nodes of its reference model. Let us consider a simple model with only one node and a cyclic edge on this node. This model can be well defined as it conforms to KM3 meta-metamodel: the node is mapped to the class concept and the edge is mapped to the reference concept. But this model, instance of the meta-metamodel, is, in their view, a metamodel. Consider now a model with two nodes linked by an edge. This model conforms to the metamodel but it is impossible to map via a μ function its edge to a node of its reference model. In fact, the μ function signature requires that every metamodel, at the level M2, contains a class similar to "reference" that allows to define links between objects at the sub-level. If not the case, edges cannot be typed and therefore the model is not definable. So, the proposed formalisation only applies for meta-metamodels but not to any model in the general case.

A prolog code is given as an appendix which validates the KM3 reflexivity but it does not follow the mathematical formalisation of the paper. For instance, the μ function is not explicit and scattered among prolog rules. The choice of prolog for such an implementation does not provide any typing or structuring mechanism.

Our work differs from this approach and separates models from reference models considering them elements of different types. It allows the definition of the reflexiveness of an initial model in a more satisfying way while not relying on the structure required at a particular level of the OMG pyramid.

3 Formalisation of a Model-Driven Engineering Framework

In this section, we give the insight of our framework. We first define the notions of *model* and *reference model*. We then describe conformity through the *conformsTo* function and also the *promotion* operation that maps a model into a reference model.

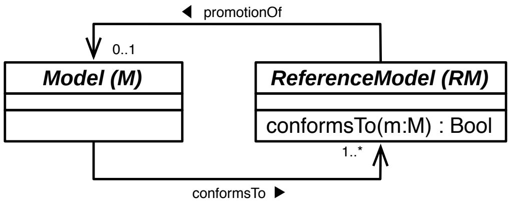

Fig. 3. Model & Reference Model Definition with UML Class Diagram Notation

3.1 Model Vs. Reference Model

Our approach consists in separating the type level from the instance level, describing them with different structures hence different types. Our aim is to provide a solid foundation to the MDE approach, thus allowing further developments in formalisation and validation activities. Following [28], we use the terms *Model* and *Reference Model* in order to distinguish between these two levels. A *Model* (M) is the instance level and a *Reference Model* (RM) is a modelling language from which we can define a family of models (Figure 3). A *RM* also specifies the semantic properties of its models. For instance, in UML, a multiplicity is defined on relations to specify the allowed number of objects that have to be linked. Moreover, OCL is used to define more complex constraints which may not have any specific graphical notation.

Into our framework, the concept of $Reference\ Model$ is not a specialisation of $Model$. They are formally defined in the following way: let us consider two sets: `Classes` represents the set of all possible classes and `References` correspond to the set of reference labels. We also consider instances of such classes, the set `Objects`.

Definition 1 (Model). *Let $\mathscr{C} \subseteq \texttt{Classes}$ be a set of classes. Let $\mathscr{R} \subseteq \{\langle c_1, r, c_2\rangle \mid c_1, c_2 \in \mathscr{C}, r \in \texttt{References}\}$ be the set of references among classes such that $\forall c_1 \in \mathscr{C}, \forall r \in \texttt{References}$, $card\{c_2 \mid \langle c_1, r, c_2\rangle \in \mathscr{R}\} \leq 1$.*

We define a model $\langle MV, ME\rangle \in \texttt{Model}(\mathscr{C}, \mathscr{R})$ as a multigraph built over a finite set MV of typed objects and a finite set ME of labelled edges such that:

$$MV \subseteq \{\langle o, c\rangle \mid o \in \texttt{Objects}, c \in \mathscr{C}\}$$
$$ME \subseteq \{\ \langle\langle o_1, c_1\rangle, r, \langle o_2, c_2\rangle\rangle \mid \langle o_1, c_1\rangle, \langle o_2, c_2\rangle \in MV, \langle c_1, r, c_2\rangle \in \mathscr{R}\ \}$$

Note that in our framework, an object can be duplicated in a model graph, associated to different classes to form different graph nodes. The semantics of this potential duplication is related to inheritance and to the possibility of object upcasting, both notions usually being present in OO languages. This concern will be further discussed in Section 4.1.

Definition 2 (Reference Model). *A Reference Model is a multigraph representing classes and references as well as semantic properties over instantiation of classes and references. It is represented as a pair composed of a multigraph (RV, RE) built over a finite set RV of classes, a finite set RE of labelled edges, with a property over models represented as a predicate function.*

We define a reference model as a triple $\langle (RV, RE), conformsTo \rangle \in \texttt{refModel}$ *such that:*

$$\begin{aligned}&RV \subseteq \texttt{Classes}\\ &RE \subseteq \{\langle c_1, r, c_2 \rangle \mid c_1, c_2 \in RV, r \in \texttt{References}\}\\ &conformsTo : \texttt{Model}(RV, RE) \rightarrow Bool\end{aligned}$$

such that $\forall c_1 \in RV, \forall r \in \texttt{References}, card\{c_2 \mid \langle c_1, r, c_2 \rangle \in RE\} \leq 1$

3.2 Conformity

Given one model *M* and one reference model *RM*, we can check conformance. The *conformsTo* function embedded into the *RM* achieves this goal. It identifies the set of valid models with respect to a reference model.

In our framework, the conformance checks on the model *M* that:

- every object *o* in *M* is the instance of a class *C* in *RM*.
- every link between two objects is such that it exists, in *RM*, a reference between the two classes typing the two elements. In the following we will say that these links are instances of the reference between classes in *RM*.
- finally, every semantic property defined in *RM* is satisfied in *M*. For instance, the multiplicity defined on references between concepts denotes a range of possible links between objects of these classes (i.e. concepts). It is treated in Section 4.1.

This notion of conformity can be found in the framework depicted in Figure 3 by a dependency between a *M* and a *RM* it conforms to.

In fact, the semantic properties associated to the reference model are encoded into the *conformsTo* function. These semantic properties are not to be given a syntax. Instead, in order to express our properties, we assume an underlying logic that should encompass OCL in terms of expressive power. As such, the problem of representing logical sentences that we may promote to properties afterwards is a secondary concern. As shown below, we can prove the MOF to be meta-circular without any OCL formula.

3.3 *Promotion*: from a MODEL to a REFERENCE MODEL

In this framework, a Reference Model can be either directly defined or built as the promotion of a given model. Despite it is represented as a predicate function *promotionOf* on Figure 3, *promotion* is an operation that allows to map a model to a reference model. The resulting Reference Model can then be used to define new models.

Promotion operation must then build the Reference Model multigraph composed of concepts, represented by nodes, and relations between these concepts, as well as the semantic properties encoded into the *conformsTo* function.

The different steps of a promotion operation are the following:

1. Identify the different concepts among the model elements, inside the model graph. These elements will be mapped at a concept level. They will define types in the Reference Model.

2. Identify relations between the previous concepts. The model graph is again used to identify paths between elements promoted to the concept level. These paths are then promoted to the reference level in the Reference Model graph.
3. The last step defines the different semantic properties that must be satisfied by models conforming to the Reference Model. The conjunction of these properties will then define the semantic properties of the $conformsTo$ function in the resulting reference model.

4 Application to EMOF and OMG pyramid issues

As a proof of concepts, we apply our approach to the formalisation of the OMG pyramid. We choose to formalise the main subset of the OMG EMOF [5, §12], Core EMOF, as the initial reference model. It is presented in Section 4.1 with the formalisation of the semantic properties that are used to define the *conformsTo* function. In Section 4.2, we show meta-circularity of Core EMOF by first defining the model of Core EMOF that conforms to the reference model of Section 4.1 and then by promoting it to a reference model that is equivalent to the initial reference model.

4.1 Core EMOF as an initial Reference Model

The reference model of this EMOF subset is defined in Figure 4 using the usual UML class diagram. It is discussed hereafter.

We informally describe the EMOF reference model and also define a set of generic properties representing its core concepts. Each property, once instantiated, is meant to yield (a conjunct of) an overall $conformsTo$ function.

The EMOF reference model, as can be guessed from Figure 4, is defined with:

$$\begin{aligned} RV \triangleq\ & \{\ NamedElement, Type, TypedElement, DataType, Boolean, \\ & String, Natural^{\top}, Class, Property\ \} \\ RE \triangleq\ & \{\ \langle Class, \texttt{ownedAttribute}, Property\rangle, \langle Class, \texttt{isAbstract}, Boolean\rangle, \\ & \langle Class, \mathrm{inh}, Type\rangle, \ldots\ \} \end{aligned}$$

where “inh” stands for the standard inheritance relation. Due to lack of space, we circumvent the formal definition of the $conformsTo$ function of EMOF and expose it in section 4.2 only as the result of a model promotion (of one of its conforming models), that instantiates our generic properties.

Classes & Properties The class (*Class* in Figure 4) is the conceptual entity that allows to define a family of objects. It can be instantiated to build *instances*. An instance is a particular *object* that has been created following constraints related to its class. Every instance is linked to exactly one class through the *instantiation relation*.

A class gathers an ordered set of *properties* (*Property*) that are duplicated to build the structure of each instance. Each of these properties is defined between a class and a type (*Type*) that can be:

- a datatype (*DataType*), associated to a specific value semantics. It is then called an *attribute*.
- another class, following the aforementioned semantics. It is then called a *reference*.

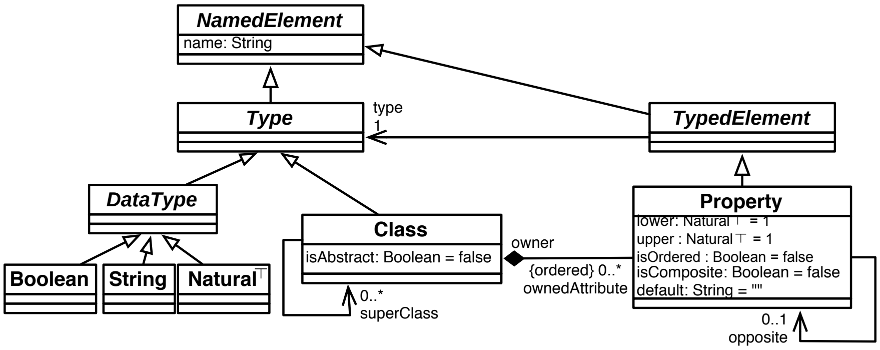

Fig. 4. The EMOF Core with UML Class Diagram Notation

Class description

Inheritance is a way to form new classes (i.e. sub-classes) using other classes (i.e. parent-classes) that have already been defined. A sub-class is linked to its parent-classes by a link of kind *superClass*. The new classes take over (or inherit) properties of the pre-existing parent classes.

In our framework, we have many open directions, chosen whether our reference model should include a notion of inheritance or not. For instance:

- No inheritance: in this context, no object may ever be duplicated and we can assume the following property.

$$noInheritance \triangleq \langle MV, ME\rangle \mapsto \forall \langle o_1, c_1\rangle, \langle o_2, c_2\rangle \in MV, o_1 = o_2 \Rightarrow c_1 = c_2$$

- Standard Inheritance: objects can be duplicated, provided these duplicates may be downcasted to a common base object. The following properties are parametrised by *inh*, taken as the standard inheritance relation. First, we define an auxiliary predicate stating that an object o of type c_1 has a downcast duplicate of type c_2.

$$\begin{aligned}&hasSub(inh \in RE, o \in \texttt{Objects}, c_1, c_2 \in \texttt{Classes}, \langle MV, ME\rangle) \triangleq c_1 = c_2\\ &\quad \vee \exists c_3 \in \texttt{Classes}, \langle\langle o, c_2\rangle, inh, \langle o, c_3\rangle\rangle \in ME \wedge hasSub(inh, o, c_1, c_3, \langle MV, ME\rangle)\end{aligned}$$

Then, we define the notion of standard inheritance. The first conjunct states that the inheritance relation only conveys duplicate objects. The second one states that every set of duplicates has a base element.

$$\begin{aligned}&standardInheritance(inh \in RE) \triangleq \langle MV, ME\rangle \mapsto\\ &\qquad \forall \langle\langle o_1, c_1\rangle, inh, \langle o_2, c_2\rangle\rangle \in ME, o_1 = o_2\\ &\qquad \wedge \forall \langle o_1, c_1\rangle, \langle o_2, c_2\rangle \in MV, o_1 = o_2 \Rightarrow \exists c \in \texttt{Classes},\\ &\qquad\qquad hasSub(inh, o_1, c_1, c, \langle MV, ME\rangle) \wedge hasSub(r, o_2, c_2, c, \langle MV, ME\rangle)\end{aligned}$$

Finally, the following property states that c_2 is a direct subclass of c_1.

$$\begin{aligned}&subClass(inh \in RE, c_1, c_2 \in RV) \triangleq \langle MV, ME\rangle \mapsto\\ &\qquad \forall \langle o, c\rangle \in MV, c = c_2 \Rightarrow \langle\langle o, c_2\rangle, r, \langle o, c_1\rangle\rangle \in ME\end{aligned}$$

Abstract Classes are identified with the attribute *isAbstract* set to true. These classes serve as parent classes and child classes are derived from them. They are not themselves suitable for instantiation. Abstract classes are often used to represent abstract concepts or entities. Features of an abstract class are then shared by a group of sibling sub-classes which may add new properties.

Therefore, a model *M* does not conform to a *RM* if it contains objects that are instances of abstract classes of *RM* without having instances of concrete derived classes as duplicates.

$$\begin{aligned}&isAsbstract(inh \in RE, c_1 \in RV) \triangleq \langle MV, ME\rangle \mapsto \\ &\quad \forall \langle o, c\rangle \in MV, c = c_1 \Rightarrow \exists c_2 \in RV, \langle\langle o, c_2\rangle, r, \langle o, c_1\rangle\rangle \in ME\end{aligned}$$

Property description

Lower & Upper Either for attributes or for references, a minimum and maximum number of instances of target concept can be defined using the *lower* and *upper* attributes. This pair is usually named *multiplicity*. In order to ease the manipulation of this datatype, we introduce the type $\mathrm{Natural}^{\top} = \mathbb{N} \cup \{\top\}$. Using both attributes, it is used to represent a range of possible numbers of instances. Unbounded ranges can be modelled using the $\top$ value for the *upper* attribute.

$$\begin{aligned}&lower(c_1 \in RV, r_1 \in RE, n \in Natural^{\top}) \triangleq \langle MV, ME\rangle \mapsto \\ &\quad \forall \langle o, c\rangle \in MV, c = c_1 \Rightarrow card(\{m_2 \in MV \mid \langle\langle o, c_1\rangle, r_1, m_2\rangle \in ME\}) \geq n\end{aligned}$$

An analogous formalisation is defined for the upper property replacing $\geq$ by $\leq$.

Opposite A reference can be associated to an *opposite* reference. It implies that, in valid models, for each such a link between two object instances of this reference, there must exists a link in the opposite direction between these two same objects.

$$\begin{aligned}&isOpposite(r_1, r_2 \in RE) \triangleq \langle MV, ME\rangle \mapsto \\ &\quad \forall m_1, m_2 \in MV, \langle m_1, r_1, m_2\rangle \in ME \Leftrightarrow \langle m_2, r_2, m_1\rangle \in ME\end{aligned}$$

Composite A reference can be *composite* and as a matter of fact, defining a set of references considered as a whole to be composite, instead of a single one, appears closer to the intended meaning. In such a case, instances of the target concept belong to a single instance of source concepts. Another constraint, a temporal one, considers the lifetimes of source and target instances to be entangled. But this constraint can not be modelled yet in our framework.

$$\begin{aligned}&areComposite(c_1 \in RV, R \subseteq RE) \triangleq \langle MV, ME\rangle \mapsto \\ &\quad \forall \langle o, c\rangle \in MV, c = c_1 \Rightarrow card(\{m_1 \in MV \mid \langle m_1, r, \langle o, c\rangle\rangle \in ME, r \in R\}) \leq 1\end{aligned}$$

Other properties, which are not treated here, are firstly the impossibility to have two opposite references which are both composite. A second one is the *ordered* property. This semantic property is only temporal and is not treated in this article.

Indeed, we choose not to model the notion of operation on classes. This notion would allow the definition of behaviours through an execution semantics. It is out of the scope of this paper.

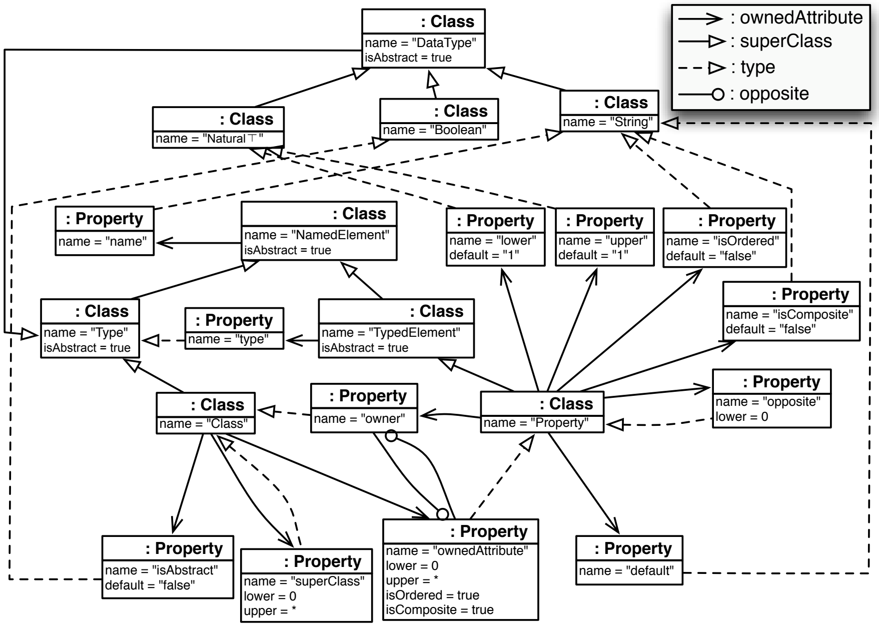

Fig. 5. The EMOF Core with UML Object Diagram Notation

4.2 Validation of the Core EMOF Meta-circularity

We now describe, in our framework, how the MOF is correctly self-defined. The usual view of the MOF presented in Figure 4 is its reference model view without the *conformsTo* function. Only a small part of the semantic properties are represented in such diagrams, like multiplicities over references. We will first introduce a model for the MOF reference model. Then we will build a promotion function, mapping this MOF model to the MOF reference model.

From a MOF Reference Model to the MOF Model The Figure 5 presents a MOF model, with the UML Object Diagram notation [3]. In this diagram, attributes with default values are omitted and links have specific graphical notations according to their types. All model elements are instances of some concepts or references of the MOF reference model presented in Figure 4 and this model satisfies the reference model semantic properties.

Each concept is instantiated as an object of type Class and each reference or attribute as an object of type Property. A property of a concept, either a reference or an attribute, is represented by a link from the object instantiating the concept to the object instantiating the property. There is also a link from the object instantiating the property to an object instantiating the type of the property. For instance, the instantiation of an attribute will point to an object instantiating the concept Boolean, String or Natural$^\top$, when the instantiation of a reference will point to an object instantiating the referenced concept.

A promotion from the MOF model to the MOF reference model We now describe a promotion function which maps the model of Figure 5, represented as a typed multigraph, to a reference model, represented by a pair composed of a multigraph and a (conjunction of) semantic properties over the multigraph elements. This function satisfies the correctness assumptions of Section 3.3. This function is defined in three parts:

1. we first build the nodes of the reference model graph,
2. we define the references between nodes,
3. we finally enrich the resulting graph with semantic properties expressed on the graph elements.

Building nodes: reference model concepts This first step is straightforward. We have to identify the types of nodes in our model that will represent concepts of our reference model.

In the MOF model, we identify objects that are of type *Class* as concepts (i.e., classes) in the target reference model. The field *name* of these objects will identify these classes in the resulting reference model. We thus have the following concepts: DataType, Natural$^{\top}$. . . .

More formally, let (MV, ME) be the graph of the model. We build the set RV of reference model vertices:

$$RV \triangleq \left\{\, o.name \,\middle|\, \langle o, Class\rangle \in MV \,\right\}$$

Building edges: reference model references Once classes have been built, the second step aims at enriching the graph with edges between nodes, and modelling references between classes.

We describe the edge building as a mapping from a sequence of edges and objects in the MOF model graph to one edge in the MOF reference model graph. Each pattern must be applied in every possible way. Considering semantic properties of the MOF reference model would be necessary if we were to prove that these pattern-matching rules are well formed.

We have only one such rule. This rule maps paths from one object o_1 of type *Class* to one object o_2 of type *Property* by an edge of type *ownedAttribute*, followed by an edge of type *type* to an object o_3 of type *Class*. Such paths are mapped to a reference from the class associated to the object o_1 to the class associated to the object o_3. They are labelled by the attribute *name* of the object o_2. These references will later be associated to semantic properties.

Let *Rule* be the predicate used in this matching rule.

$$\begin{aligned} Rule(o_1, o_2, o_3) \triangleq\ & \langle\langle o_1, Class\rangle, \texttt{ownedAttribute}, \langle o_2, Property\rangle\rangle \in ME \\ & \wedge \langle\langle o_2, Property\rangle, \texttt{type}, \langle o_3, Class\rangle\rangle \in ME \end{aligned}$$

RE is the set of edges in the reference model graph built with this rule.

$$RE \triangleq \left\{\, \langle o_1.name, o_2.name, o_3.name\rangle \,\middle|\, Rule(o_1, o_2, o_3) \,\right\}$$

Enriching the reference model with semantic properties The last step enriches the resulting reference model graph by specifying semantic properties over models. This global semantic property (seen as a $conformsTo$ function) is the conjunction of the different single properties attached to the promoted reference graph elements. Each of these single properties is defined as an instantiation of a generic property as defined in Section 4.1.

inheritance Every link labelled *superClass* between two objects of type *Class* is mapped to an inheritance relation between the two associated classes.

$$\bigwedge \left\{ subClass(inh, o_1.name, o_2.name) \,\middle|\, \langle\langle o_1, Class\rangle, \texttt{superClass}, \langle o_2, Class\rangle\rangle \in ME \right\}$$

abstract classes Every class c resulting from the promotion of some object o with an attribute `isAbstract` which value is `true` has the property $isAbstract(c)$. It denotes abstract classes that could not be instantiated.

$$\bigwedge \left\{ isAbstract(inh, o.name) \,\middle|\, \langle\langle o, Class\rangle, \texttt{isAbstract}, \langle \texttt{true}, Boolean\rangle\rangle \in ME \right\}$$

lower Every reference $o_2.name$ between classes $o_1.name$ and $o_3.name$, for every triple (o_1, o_2, o_3) matching the reference promotion rule, has a minimum number of $o_2.\texttt{lower}$ instantiations in valid models.

$$\bigwedge \left\{ lower(o_1.name, o_2.name, o_2.\texttt{lower}) \,\middle|\, Rule(o_1, o_2, o_3) \right\}$$

upper This semantic property is defined similarly.

$$\bigwedge \left\{ upper(o_1.name, o_2.name, o_2.\texttt{upper}) \,\middle|\, Rule(o_1, o_2, o_3) \right\}$$

ordered Every reference between classes that has been built using the reference matching rule and for which the object o_2 has an attribute named *isOrdered* with value $true$ satisfies the ordered property. This property is related to the order in which target objects of a reference may be accessed and therefore implies that some notions of time and execution are defined. This property cannot not be treated here, due to the lack of some operational semantics.

opposite Every reference promoted from an object o_1 with a link typed `opposite` to another object o_2 also promoted to a reference defines these two references as opposite.

$$\bigwedge \left\{ isOpposite(o_1.name, o_2.name) \,\middle|\, \langle\langle o_1, Property\rangle, \texttt{opposite}, \langle o_2, Property\rangle\rangle \in ME \right\}$$

composite Every reference between classes that has been built using the reference matching rule and for which the object o_2 has an attribute named *isComposite* with value $true$ satisfies the composite property. We first define the set of global set of composite relations.

$$compRelations \triangleq \{o_2.name \mid \langle o_2, Property\rangle, \texttt{isComposite}, \langle \texttt{true}, Boolean\rangle\rangle \in ME\}$$

Then we state the promotion itself.

$$\bigwedge \left\{ areComposite(o_1.name, compRelations) \,\middle|\, \langle o_1, Class\rangle \in MV \right\}$$

default value Every reference between classes that has been built using the reference matching rule and for which the object o_2 has an attribute named *default* with value v has a default value v when instantiated. This property is only valid for the initial state of objects, so we cannot treat it here, due to the lack of some operational semantics.

4.3 Definition of the OMG Pyramid

The framework defined allows the formalisation of the pyramid proposed by the OMG (Figure 6). Indeed, the *Model* and *Reference Model* levels are necessary and sufficient to define in a clear and formal way *model*, *metamodel* and *meta-metamodel* differentiating the various abstraction levels of the OMG pyramid.

The MOF metamodelling language (i.e., meta-metamodel) allows to standardise the concepts used to describe the various modelling languages (i.e., metamodels). As introduced by the OMG, "the MOF has the reflection property that extends the MOF with the ability to be self describing". This property makes it possible to limit to four the number of abstraction levels of the OMG pyramid.

However, this property means that classes (i.e., types) defined in the MOF are instances of classes of this same MOF. From a formal point of view, it is not possible that a type is an instance of a type!

Within our framework, we define the MOF (Figure 6, M3) like a *Reference Model* ($MOF : RM$) built such that there is a model ($MOF : M$) in conformity with the *Reference Model* and we construct a *promotion* that makes possible to obtain exactly the $MOF : RM$ *Reference Model*.

The description of all other levels is similar. A modelling language (i.e., metamodel) describes the concepts (i.e., types) that can be instantiated using this language. Thus, a metamodel (e.g. Figure 6, M2) is defined like a *Reference Model* (e.g., $UML : RM$ for the UML modelling language) obtained through the promotion of a *Model* ($UML : M$) that conforms to the $MOF : RM$.

In the same way, a *model* (abstraction of the real world, e.g., $System : M$ at the level M1 in Figure 6) is defined from, and conforms to, a *Reference Model* (e.g. $UML : RM$) of the M2 level.

Note that the framework which is defined in this article supports the formalisation of the OMG pyramid but, with the advantage of being more general, does not bound the number of abstraction levels (as described by OMG). Nonetheless, the classification of the OMG is still meaningful.

5 Conclusion

In this paper, we have proposed a mathematical formalisation of the MDE framework. Our work, inspired by [28], consists in defining the notions of model and reference model based on typed multigraphs. We also formalise the two operations that are fundamental to the MDE approach, *conformsTo* that indicates whether a model is valid with respect to a reference model and *promotion* which builds a reference model from a

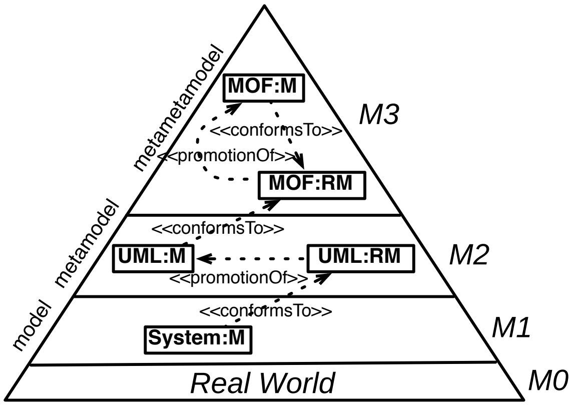

Fig. 6. OMG Pyramid with our Framework

model. It is then possible to prove that EMOF is self-defined and to derive the MOF reference model from the MOF model. The OMG pyramid but also any tower of reference model/model with an arbitrary number of levels may be built using this process: starting from a reference model, a model is defined that conforms to it and is then promoted to a new reference model. In this paper, the formalisation is presented using set theory. This formal framework for MDE is being implemented using the COQ proof assistant [29] with the aim of being able to formally reason on metamodels (in fact reference model) and models.

Short-term perspectives include formalising common operations on models, such as the *merge*, *import* and *fusion* operators as well as models parametrised by models. Also, reflecting the OCL logic into our formalism seems to be a rather simple but useful task, as it would allow standard presentation of reference models to be embedded in our framework in a straightforward way.

In this paper, we have only addressed the formalisation of static semantics, applied on EMOF. A further more complex task would be to deal with operational semantics (including semantics of so-called operations) in the same spirit, our wish being to handle as homogeneously as possible both structural relations between two concept instances and next-time relations between concept instances and their images (new states represented by new models) as a side-effect of an operation calling. It would allow to define a reference semantics for a given DSL. That could be use to check the behavioural equivalence with different translations from this DSL to other technical spaces.

Finally, from a theoretical standpoint, when a complete framework exists endowed with its final characteristics, it will remain to find out its relative pros and cons with the much more well-behaved and structured category theory. For the time being, our ideas have led to a typed framework, but with quite a loose structure. In other words, is there a specific place to be taken between untyped set theory and category theory or is one of the aforementioned mathematical languages able to fully handle MDE activities on its own in a smart way.

References

1. Bézivin, J.: In Search of a Basic Principle for Model Driven Engineering. CEPIS, UPGRADE, The European Journal for the Informatics Professional **V**(2) (2004) 21–24
2. Object Management Group, Inc.: Unified Modeling Language (UML) 2.1.1 Infrastructure. (February 2007) Final Adopted Specification.
3. Object Management Group, Inc.: Unified Modeling Language (UML) 2.1.1 Superstructure. (February 2007) Final Adopted Specification.
4. Object Management Group, Inc.: Software Process Engineering Metamodel (SPEM) 2.0 RFP. (novembre 2004) ad/04-11-04.
5. Object Management Group, Inc.: Meta Object Facility (MOF) 2.0 Core Specification. (January 2006) Final Adopted Specification.
6. Miller, J., Mukerji, J.: Model Driven Architecture (MDA) 1.0.1 Guide. Object Management Group, Inc. (June 2003)
7. Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"?. IEEE Computer **37**(10) (2004) 64–72
8. Object Management Group, Inc.: UML Object Constraint Language (OCL) 2.0 Specification. (October 2003) Final Adopted Specification.
9. Clark, T., Evans, A., Sammut, P., Willans, J.: Applied Metamodelling - A Foundation for Language Driven Development. version 0.1 (2004)
10. Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving Executability into Object-Oriented Meta-Languages. In: MODELS/UML'2005. LNCS, Montego Bay, Jamaica, Springer Verlag (October 2005)
11. Combemale, B., Rougemaille, S., Crégut, X., Migeon, F., Pantel, M., Maurel, C., Coulette, B.: Towards a Rigorous Metamodeling. In: 2nd International Workshop on Model-Driven Enterprise Information Systems (MDEIS), Paphos, Cyprus, INSTICC (May 2006)
12. Bézivin, J., Jouault, F., Kurtev, I., Valduriez, P.: Model-based DSL Frameworks. In: Onwards! track of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, Oc tober 22-26, 2006, Portland, OR, USA, ACM (2006)
13. Chen, K., Sztipanovits, J., Abdelwalhed, S., Jackson, E.: Semantic Anchoring with Model Transformations. In: Model Driven Architecture, Foundations and Applications (MDA-FA). Volume 3748 of LNCS., Springer (2005) 115–129
14. Gurevich, Y.: The Abstract State Machine Paradigm: What Is in and What Is out. In: Ershov Memorial Conference. (2001)
15. Agrawal, A., Karsai, G., Kalmar, Z., Neema, S., Shi, F., Vizhanyo, A.: The Design of a Language for Model Transformations. Technical report, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37235, USA. (2005)
16. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Satellite Events at the MoDELS 2005 Conference, LNCS 3844, Springer (2006) 128–138
17. Ruscio, D.D., Jouault, F., Kurtev, I., Bézivin, J., Pierantonio, A.: Extending AMMA for Supporting Dynamic Semantics Specifications of DSLs. Technical Report 06.02, LINA (2006)
18. Combemale, B., Garoche, P.L., Crégut, X., Thirioux, X.: Towards a Formal Verification of Process Model's Properties – SimplePDL and TOCL case study. In: 9th International Conference on Enterprise Information Systems (ICEIS), Portugal, INSTICC (June 2007)
19. Milner, R.: Communication and concurrency. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1995)
20. Atkinson, C., Kuhne, T.: Model-Driven Development: A Metamodeling Foundation. IEEE Softw. **20**(5) (2003) 36–41
21. Seidewitz, E.: What models mean. Software, IEEE **20**(5) (2003) 26–32

22. Bézivin, J.: In search of a Basic Principle for Model-Driven Engineering. Novatica – Special Issue on UML (Unified Modeling Language) **5**(2) (2004) 21–24
23. Favres, J.M.: Towards a Basic Theory to Model Model-Driven Engineering. In: 3rd Workshop in Software Model Engineering (WiSME@UML), Lisbon, Portugal (October 2004)
24. Kuhne, T.: Matters of (meta-) modeling. Software and Systems Modeling (SoSyM) **5**(4) (December 2006) 369–385
25. Steel, J., Jézéquel, J.M.: On model typing. Software and Systems Modeling (SoSyM) (2007)
26. Blanc, X., Gervais, M.P., Sriplakich, P.: Model Bus: Towards the Interoperability of Modelling Tools. In: Model Driven Architecture, Foundations and Applications (MDAFA). Volume 3599 of LNCS., Twente, The Netherlands, Springer (June 2004) 17–32
27. Taentzer, G.: AGG : A Graph Transformation Environment for Modeling and Validation of Software. In Springer-Verlag, ed.: AGTIVE'03. Volume 3062 of LNCS. (2003) 446–453
28. Jouault, F., Bézivin, J.: KM3: a DSL for Metamodel Specification. In: IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS). Volume 4037 of LNCS., Springer (2006) 171–185
29. The Coq Development Team: The Coq Proof Assistant Reference Manual – Version V8.1. (2006) http://coq.inria.fr.