Metamodels: Definitions of Structures or Ontological Commitments?

Ivan Kurtev

Software Engineering Group, University of Twente
7500 AE Enschede, the Netherlands
kurtev@ewi.utwente.nl

Abstract. The concept of metamodel is central in Model Driven Engineering (MDE). It is used to define the conceptual foundation of modeling languages. There exist specialized languages for specifying metamodels known as metalanguages. The most popular of them are object-oriented and support defining structures of the metamodels without considering the semantical underpinnings of the structures. In this paper we study the nature of metamodels from philosophical perspective. We claim that a metamodel is something more than an abstract syntax definition: it is an ontological commitment that guides the modeler in his perception about the real world phenomenon. Therefore, metalanguages should derive their foundation from the study of Ontology. We employ an ontological theory based on the Four-category ontology and the principles of metaphysical realism. We propose a metalanguage called OGML (Ontology Grounded Metalanguage) built upon the basic concepts of this ontology.

1 Introduction

In Model Driven Engineering, the term metamodel is assigned with several meanings most of them related to the concept of modeling language. Often, metamodel is considered as the definition of the abstract syntax of a language. Another definition states that a metamodel models the conceptual foundation of a modeling language. Since MDE opens the possibility for defining multiple modeling languages, metamodeling is an important and repeating activity in MDE. Currently it is supported by metalanguages (i.e. languages that express metamodels) and their accompanying tools. A metalanguage is therefore a domain-specific language (DSL): its domain consists of metamodels.

In the ideal case, the development of a DSL is preceded by a domain analysis. Then the identified concepts in the domain are mapped to first class syntactical constructs in the DSL. Unfortunately, in the case of current metalanguages (e.g. MOF [15], ECore [6], etc.) we can find only few traces of such a domain analysis. This puts a question about the adequacy of these languages with respect to their problem domain and raises the need of a sound analysis of the metamodeling activity.

If a metamodel is perceived only as an abstract syntax definition, that is, a definition of a structure that abstracts from the concrete syntax, then we may employ multiple languages to describe such a structure. In this case, we even do not need to under-

stand deeply the nature of metamodeling. Current metalanguages are designed to satisfy mostly pragmatical needs or are a result of some historical development. For example, MOF is conceived as a subset of UML and is based on object-oriented concepts. This is easy to understand: object-orientation is the dominant development paradigm today. An object-oriented metalanguage is expected to be easily embraced by many users familiar with OO programming. However, often MOF is considered as a rather complex language. Nowadays we observe a successful adoption of ECore and KM3 [10]. These two languages provide a significant simplification of MOF and add another pragmatic motivation apart from the familiarity to the developers community: simplicity. None of these languages is built upon an analysis of the domain of metamodeling.

If we only aim at metamodels as definitions of structures then we can choose arbitrary language for metamodeling. Metamodels may be represented as graphs and therefore we may benefit from the sound mathematical foundation of the graph theory. Depending on the "cultural" background of the metamodeler he/she may also choose first-order logic, grammars, DTDs, XML schemas, ontologies, etc. In other words, the choice of a metalanguage is somehow arbitrary and the factors that matter are not related to the metamodeling at all.

The result of this approach is that many issues related to metamodeling and language definition are not well-understood. These issues are reported in the literature [2, 4, 5]: different instantiation mechanisms, replication of concepts across metalevels, the number of metalevels and criterion for determining them, lack of higher-order types, etc. These issues would have been addressed if the second view on metamodels was taken as a leading one: a metamodel is a model of the conceptual foundation of a modeling language. This view logically leads to the need for studying the nature of this conceptual foundation.

In this paper we study the domain of metamodeling. The main research question is what is in a metamodel. By answering this we will be able to identify commonalities and variabilities in metamodels. A metalanguage should be capable of expressing the solution space for metamodels and to relate its constructs to a domain of concepts with clear semantics. For our study this domain is the philosophical discipline of Ontology. We give our motivation further in the paper.

We propose a metalanguage whose constructs are derived from the ontological theory of universals and individuals. The language allows expressing different views about the structure of the world, a concept that we refer to as *ontological commitment*. Models are constrained by the adopted ontological commitment. Therefore, metamodeling activity is concerned with defining various commitments. A metalanguage provides the vehicle to express them.

This paper is organized as follows. Section 2 presents our view on metamodeling and its relation to Ontology. Section 3 discusses the philosophical background and gives domain analysis for concepts that a metalanguage should provide. Section 4 describes the major constructs of a metalanguage called OGML (Ontology Grounded Metalanguage). Section 5 discusses key topics in this research. Section 6 concludes the paper.

2 What is in a Metamodel?

We assume that a metamodel is a model of the conceptual foundation of a modeling language [10, 12, 16]. Figure 1 shows the well known metamodeling stack. In the bottom level we have the real world that is represented in several models (*Model 1*, *Model 2*, *Model 3*).

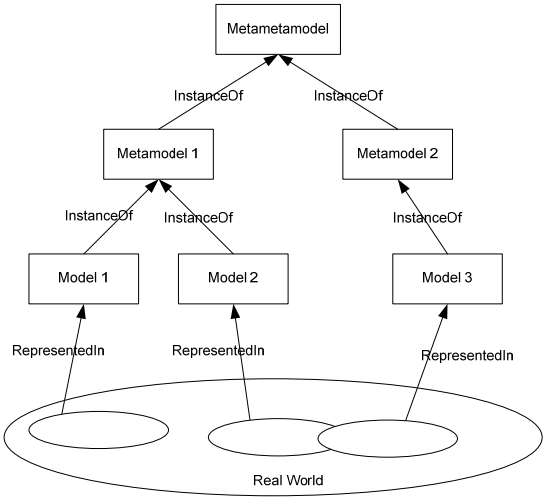

Fig. 1 Metamodeling stack

Models represent different phenomenon from the real world. Some phenomenon may be non-overalaping (e.g. the ones represented by *Model 1* and *2*) and some may be overlapping (e.g. the ones represented by *Model 2* and *3*). We have two modeling languages modeled by *Metamodel 1* and *Metamodel 2*. Models are instances of their metamodels. The features of the real world capturable by a model are determined by the metamodel. A metamodel represents a view on the world. It is an ontological commitment that specifies the things that an observer may see in the world. The existing work in modeling languages, linguistics, formal ontology, and knowledge representation shows that ontological commitments may vary rather broadly. For example, a possible view is that the world consists of objects that must be direct instances of exactly one class. An object cannot be created without a class. In this case the basic ontological commitment is class-centric and can be traced back to the Plato and his world of ideas. Another possible view is that the world consists of objects and some intelligent agents (e.g. human beings) are capable of building conceptualizations upon the objects by grouping them into sets with common features. In this view, object existence is independent of classes and classes are provided posteriori. To continue with the examples we may include the notion of time and space as fundamental parts of the reality and to explicitly represent them in a metamodel.

Regardless the differences among the ontological commitments their reference point is the same: the real world. Only the parts that are perceivable in this world vary. Therefore, a metametamodel (or a metalanguage) should allow expressing onto-

logical commitments by choosing them from a certain solution space. Currently, metametamodels rely on the object-oriented paradigm which is just one possible conceptualization of the world. More precisely, they are class-based and assume the precedence of classes over the objects. Furthermore, their choice is not motivated by an analysis what is possible to be expressed in a model but is a rather pragmatically motivated choice.

But how do we identify a solution space for ontological commitments, that is, metamodels? What is the domain from which the concepts and their variations come from? As we said, the most important invariant in Figure 1 is the real world. Whatever the view upon the world is, it cannot violate and contradict the fundamental structure of the world and its most basic underpinning concepts. The philosophical discipline that studies the nature of being (existence) is the Ontology. Therefore, we analyze the domain of Ontology[1] and choose a set of concepts that will be the building blocks of our metamodeling language.

3 Philosophical Background

In this paper we use an ontology called *Four-category ontology* that can be traced back to Aristotle and is also used in several contemporary works on Formal Ontology [9, 7]. In this ontology, the basic distinction is between *individuals* and *universals* as the most fundamental entities of being. The ontological study that claims the existence of universals is known as *metaphysical realism* [1, 13]. Figure 2 depicts the concepts in this ontology.

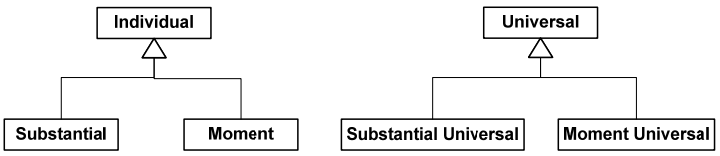

Fig. 2 Four-category ontology

Individuals are classified as *Substantial* and *Moment* individuals. Substantial individual or just *substance* is something that can exist by itself without depending on the existence of other individuals. This existential independence is the core feature of substances and gives the major distinction from moment individuals. Examples of substantial individuals are cars, people, books, etc. In the programming languages and modeling languages substantial individuals are usually represented as objects (e.g. Java object and UML object).

Moments are individuals that exist in other individuals. Moments cannot exist standalone, they are existentially dependent on at least one individual (called *bearer*). Example of a moment is the red color property of a car. In that case the red color moment exists in the substance car. The relation between a moment and its bearer(s)

[1] *Ontology* with capital *O* should be distinguished from the word *ontology* with low *o*. The first one refers to the philosophical discipline and the second one refers to a specification of conceptualization [8] in the context of computer science.

is called *Inherence* relation. Moments may inhere in more than one individual, i.e. their existence depends on more than one individual. For example, the fact that Harry and Sally are married creates a relation between two people. This relation corresponds to a moment that relies on the existence of two individuals: the people being married.

It is philosophically debatable if moments can inhere in other moments. In this paper we assume that moments inhere in individuals. This assumption allows a moment to inhere in another moment.

In programming and modeling languages moments are called in various ways: *slot* and *link* in UML, *field* in Java, etc.

Universals are entities that can be instantiated in individuals. According to Aristotle, universals can only exist via their individuals and not independently from them. The individuals that exemplify a universal have something in common. For example, things that consist of matter have mass. The actual value of the mass varies but the mass is shared among the individuals. In this case mass is a universal.

As we mentioned earlier we employ an ontological view that accepts the existence of universals. It is beyond the scope of this paper to give the philosophical discussion about the theory of universals and their existence (for example, Nominalism objects the existence of universals). There are practical reasons to accept such a view. Some kinds of universals are expressed by classes in programming and modeling languages. The concept of class should be expressible in the various ontological commitments.

Universals are classified into *substantial universals* and *moment universals*. As the names suggest, substantial universals are exemplified (i.e. exist through) by substantial individuals and moment universals are exemplified by moment individuals. *Instantiation relation* is the relation between an individual and a universal that exists in this individual.

Similarly to the concepts presented before, universals have their representatives in the existing computer languages. UML classes correspond to substantial universals. UML attributes and associations roughly correspond to moment universals.

The presented basic ontology is very simple and does not accommodate significant part of the available ontological knowledge. The ontology does not consider three fundamental concepts: time, space, and part-whole relation. For the latter one, there exists well developed theory called *mereology*. It is difficult to decide which concepts to be taken into account when an engineering solution needs to be crafted. We opt for these four basic categories and the relations among them as the first step in our experiment in applying ontological categories in defining metamodels. Missing concepts should be defined per metamodel if needed.

Now we can refine Fig. 1 armed with the knowledge derived from the domain of Ontology and can give answers about the content of the boxes in the diagram. The result is presented in Fig. 3.

The *real world* level consists of individuals and universals.

Models level represents abstractions of individuals and universals following a given notation (textual or visual).

Metamodels level builds ontological theories about individuals and universals. Since a modeling language is usually a compromise between various factors we do not require a faithful repetition of the concepts just explained. If we want a full com-

mitment to the achievements of the modern formal ontology we will end up with a single modeling language that will be useful for conceptual modeling only! Furthermore, we would like to accommodate existing languages that often violate the "ideal" ontological commitment. For example, an object-based language like Self recognizes only object as a fundamental modeling concept. In a hypothetical metamodel of Self, the part concerning universals will correspond to the concept of prototypical object. Java assumes that no object can exist without a class. However, one view about the universals claims the opposite: no universal can exist without at least one individual. To continue with the violations we can consider a language that has only binary relations and therefore cannot express the concept of moment that exists in exactly one individual (such moments are known as *qualities*).

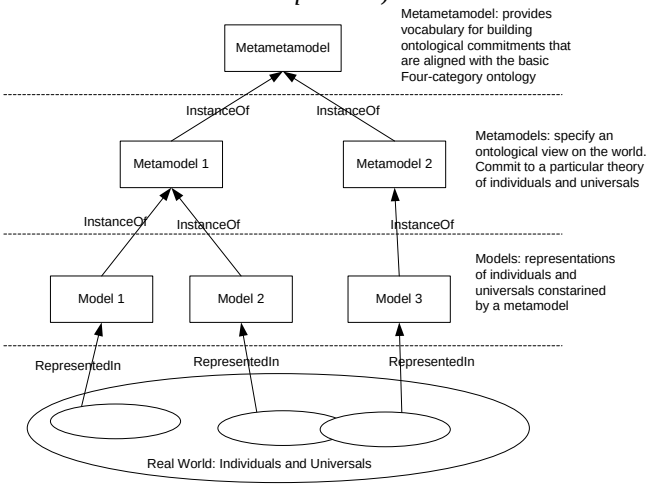

Fig. 3 Metamodeling stack and the Four-category ontology

Regardless the violations, however, a metamodel should be expressed in terms of the four basic concepts eventually restricting them. Thus, the purpose of the *metametamodel* should be to provide a vocabulary for expressing theories about universals, individuals, and the relations among them. The philosophical theory presented in this section gives us a stable and well-founded set of concepts to start with building our metalanguage.

4 Ontology Grounded Metalanguage

Ontology Grounded Metalanguage (OGML) is an experimental language for studying the definition of modeling languages based on ontological principles. It derives its constructs from the analysis presented in the previous section. The language is self-reflective like many of the current metalanguages (M3 level languages in the terminology of OMG MOF).

4.1 Basic Taxonomy: Definitions, Individuals, and Universals

For each category in the ontology we provide a construct that allows creating definitions of that category. The core hierarchy is shown in Fig. 4. *Definition* is the root of the hierarchy. It is specialized into definitions for individuals and universals (constructs *IndividualDefinition* and *UniversalDefinition* respectively). *IndividualDefinition* is specialized into *ObjectDefinition* and *PropertyDefinition*. We changed the terminology established so far by replacing the term substantial individual with the more common term *object* and moment with *property*. Similarly, *UniversalDefinition* is specialized into *SubstantialDefinition* and *MomentDefinition*.

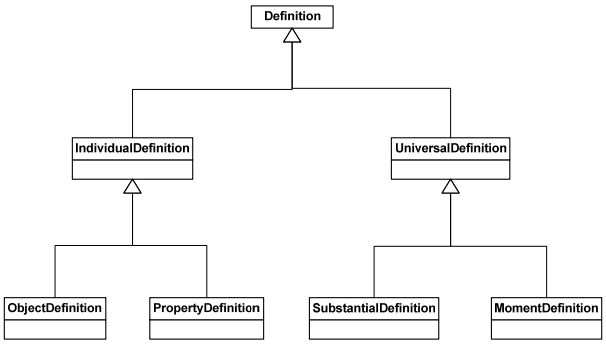

Fig. 4 The basic taxonomy in OGML

The leaves of the taxonomy are universals whose instances are other universals, that is, they are *higher-order* universals. The instances of the leaves are definitions of the corresponding concepts from the Four-category ontology. Thus, we have the means to express various versions of the basic ontology adapted to the problem at hand.

The following example shows a limited part of the definition of the UML metamodel expressed in OGML:

```
Language UML {
     SubstantialDefinition Class {}
     MomentDefinition Attribute{}
     MomentDefinition Association{}
     ObjectDefinition Object {}
     PropertyDefinition Slot {}
     PropertyDefinition Link {}
}
```

This example is refined further in this section when more OGML constructs are introduced. The UML metamodel recognizes three kinds of universals: *Class*, *Attribute*, and *Association*. The last two are moment universals. Their counterparts in the set of individual definitions are *Object*, *Slot*, and *Link* respectively.

4.2 Moment Definitions in OGML

Moment is an individual that exists in other individuals. Moment universal is a classifier for moments (in OGML we call them properties). In order to define moment universals in a modeling language, we need to specify the existential dependency relation. Therefore, the definition of *MomentDefinition* in Fig. 4 is not complete. The construct used to define existential dependency relation is called *DependencyRelation*. It is shown in Fig. 5.

The rationale behind it is that when defining a moment universal we have to enumerate the universal definitions on which it depends and to specify the properties of the dependencies. *DependencyRelation* construct by itself is a moment definition. It has three attributes: *multiplicity*, *momentDefinitionRole*, and *universalDefinitionRole*.

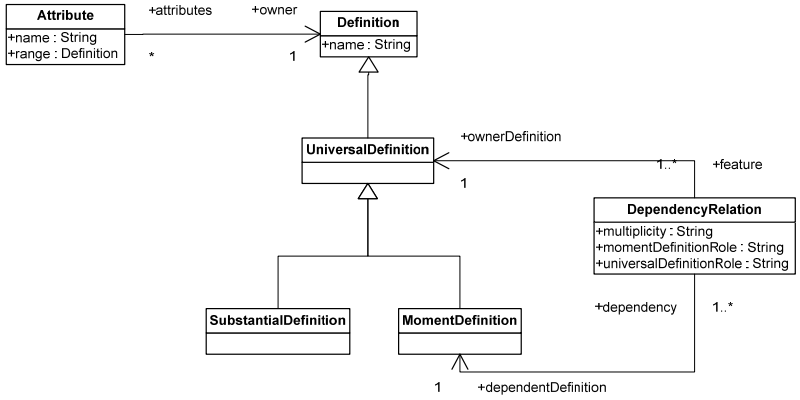

Fig. 5 Attribute and DependencyRelation moment definitions

Instances of the *DependencyRelation* are concrete moment universals that classify the instances of *PropertyDefinition*. An instance of *DependencyRelation* relates an instance of *UniversalDefinition* and an instance of *MomentDefinition*. The role attributes give names of the roles these two parties play in such a relation. The *multiplicity* attribute indicates how many moment definitions may depend on a given universal definition. For example, in Fig.5 the relations between *UniversalDefinition* and *DependencyRelation* and between *MomentDefinition* and *DependencyRelation* (denoted as UML associations) are instances of *DependencyRelation*. The direction of the associations goes from the dependent concept to the concept it depends upon. The dependency between *DependencyRelation* and *UniversalDefinition* states that at least one moment definition is depending on a given universal definition. There is a fundamental ontological principle behind that: there is no entity without properties. Furthermore, the fact that *DependencyRelation* depends on *UniversalDefinition* and not on *SubstantialDefinition* allows us to define moment universals that depend on other moment universals. Ultimately this allows properties to have properties.

OGML defines another moment definition: *Attribute*. Every definition may have attributes. Attributes depend on only one definition. Below we illustrate the self-

reflective nature of OGML by giving the *instanceOf* relations among the constructs in Fig. 5.

```
Definition : SubstantialDefinition
UniversalDefinition : SubstantialDefinition
MomentDefinition : SubstantialDefinition
SubstantialDefinition : SubstantialDefinition
DependencyRelation : MomentDefinition
Attribute : MomentDefinition
```

All the concrete attributes are instances of *Attribute* and the dependency relations (shown as UML associations) are instances of *DependencyRelation*.

The example of defining a simplified UML may be now refined by defining the moment definitions *Attribute* and *Association*. The dependency relations are represented by the *dependsOn* keyword:

```
MomentDefinition Attribute{
              attribute name : String;
              attribute type : Class;
              dependsOn Class universalDefinitionRole="owner"
                              momentDefinitionRole=”attributes”
                              multiplicity = "*";
}

MomentDefinition Association{
              attribute name : String;
              dependsOn Class universalDefinitionRole="source"
                              momentDefinitionRole=”outgoing”
                              multiplicity = "*";
              dependsOn Class universalDefinitionRole="target"
                              momentDefinitionRole=”incoming”
                              multiplicity = "*";
}
```

4.3 Generalization and Instantiation in OGML

Generalization and *Instantiation* are fundamental notions in conceptual modeling. From ontological perspective they have a clear meaning. However, different languages used in the practice show variations of the meaning. For example, in Java we have two different generalization relations (known as *extension*): one for classes and one for interfaces. *InstanceOf* relations expose slight differences likewise. Therefore, in OGML we provide constructs that allow language modelers to define their own generalization and instantiation mechanisms. These constructs are shown in Fig. 6.

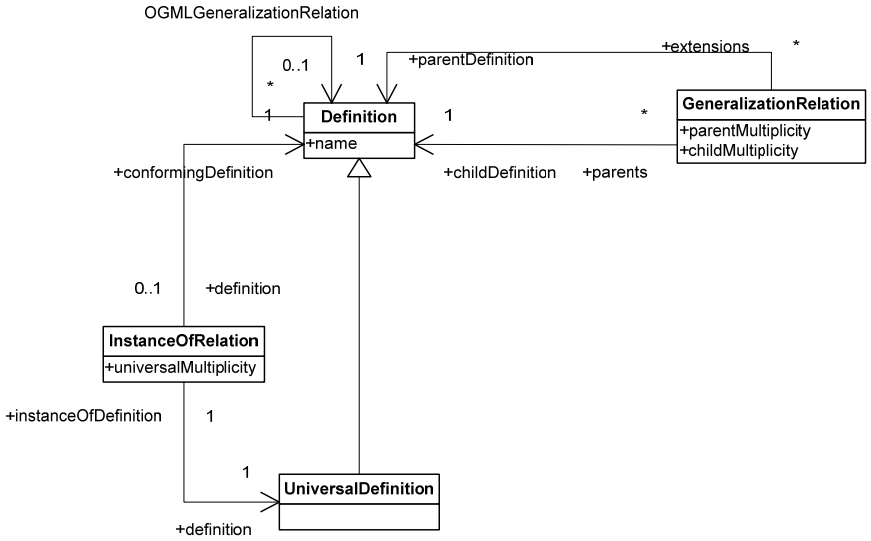

Fig. 6 Generalization and Instantiation relations

GeneralizationRelation is a moment definition. It has attributes *parentMultiplicity* and *childMultiplicity* that allow constraining a particular generalization relation regarding the allowed number of general and specialized concepts. OGML by itself has a concrete generalization relation that allows a definition to inherit from not more than one other definition (single inheritance). It is labeled *OGMLGeneralizationRelation*.

InstanceOfRelation construct relates definitions and universal definitions. In this way we can define higher-order universals. *InstanceOfRelation* has attribute *universalMultiplicity* that indicates the number of universals an object may be instantiated from. In contrary to most of the object-oriented languages that allow only one class to be used for the creation of an object, there are languages that allow more than one class to be used (e.g. RDF Schema [3] and OWL [17]). Furthermore, we should allow languages that recognize only individuals in the world. Such languages will not use universals and therefore instantiation relation is not usable in that case.

Below we indicate how these constructs are instantiated from the constructs in OGML:

```
MomentDefinition GeneralizationRelation {
  attribute parentMultiplicity : String;
  attribute childMultiplicity : String;
  dependsOn Definition universalDefinitionRole = "parentDefinition"
                       momentDefinitionRole = "extensions"
                       multiplicity = "*";
  dependsOn Definition universalDefinitionRole = "childDefinition"
                       momentDefinitionRole = "parents"
                       multiplicity = "*";
}

GeneralizationRelation OGMLGeneralization {
  parentDefinition = Definition;
```

```
  childDefinition = Definition;
  parentMultiplicity = "0..1";
  childMultiplicity = "*";
}

MomentDefinition InstanceOfRelation {
  attribute universalMultiplicity : String;
  dependsOn Definition universalDefinitionRole = "conformingDefinition"
                       momentDefinitionRole = "definition"
                       multiplicity = "0..1";
  dependsOn UniversalDefinition universalDefinitionRole = "definition"
                       momentDefinitionRole = "instanceOfDefinition"
                       multiplicity = "0..1";
}
```

We do not show the definition of *Inherence* relation due to its similarity to *DependencyRelation*. *Inherence* relates *IndividualDefinition* and *PropertyDefinition*.

We presented OGML language in an informal manner without giving a formal semantics in a mathematical notation. Currently, a prototype of the language is implemented on top of the AMMA platform [12]. The KM3 definition of OGML defines its abstract syntax. A TCS specification [11] defines its concrete syntax used in the examples shown in this paper. We defined the self-reflective description of OGML in the OGML syntax. These may be obtained from [14].

5 Discussion

An important benefit of the approach taken in OGML is that a modeling language definition has a better semantical description of some of its constructs compared to the descriptions provided by MOF, ECore and similar languages. Consider a metamodel expressed in MOF. The modeler may define the *instanceOf* relation and the generalization relation but they will be typically expressed as associations not semantically distinguishable from other associations in the metamodel. In OGML, instantiation and generalization are instances of first class constructs that capture the corresponding meaning.

OGML narrows the gap between the domain of metamodeling and the notion of *real world semantics*. In general, semantics requires a relation from the subject being defined to a domain of concepts with clear meaning. Such a domain is usually a mathematical structure as observed in the existing approaches for defining language semantics. However, it may be a set of real world entities. The concepts of individuals and universals refer to entities found in the real world and thus give the name of this type of semantics. At every model level we may identify the relation between a language construct and its counterpart: individual, universal, universal definition, etc.

The real world semantics is not intended to replace the semantics of a language in the traditional sense. For a given metamodel (modeling language) we still need to specify its semantics in a formal way. The two types of semantics are complimentary and may exist together.

An important issue in the design of OGML was which ontological concepts to be included in and left aside. The notions of time and space, for example, are not in-

cluded. Building OGML on a full ontological theory will result in a heavy language with meaning defined by philosophy thus putting a question about its usability in practice. OGML is created for experimental purposes. We decided to choose a simple set of ontological concepts that are found in the existing computer languages. In Section 3, we mentioned constructs equivalent to the elements of the Four-category ontology found in the well established languages. We believe that this improves the understandability of the language. In case somebody needs more advanced ontological concepts, e.g. part-whole taxonomy, he/she should define it in the concrete modeling language being specified.

It should be mentioned that the presented language constructs only form a vocabulary for defining metamodels. OGML has also a part that defines its own view on universals and individuals, that is, OGML defines a full ontological commitment. This part is not included in the paper for the sake of brevity. We will report on it in a future paper. The interested reader may consult [14] for the full definition of OGML.

6 Conclusions and Future Work

We presented a metamodeling language based on an analysis of the nature of metamodels. In our view, metamodels are ontological commitments that specify what can be observed in real world phenomenon. The possibilities to construct such commitments are derived from the philosophical study of Ontology. We based the proposed OGML language on the Four-category ontology that is based on the notions of individuals and universals. Current metamodeling languages employed in MDE are based on the heritage of object-orientation and are not derived from a systematic analysis of the problem domain of metamodeling.

From that point of view, the existing metamodeling languages are more or less ad-hoc solutions. The same is valid for approaches that base metamodeling on the graph theory, grammars, and other formalisms. We do not want to put a question on their merits. We would like to stress the importance of a systematic analysis of the metamodeling from philosophical point of view.

Often, a requirement for simplicity is imposed on a metamodeling language. In our opinion this requirement is vague and is difficult to judge when is achieved. What is simplicity in the context of metamodeling? Is it the number of the constructs in the metamodeling language? If so, then Lisp is the perfect metamodeling language because it defines only the concepts of list and atom for building structures. However, Lisp is not used for metamodeling!

Instead of simplicity we prefer the principle of *parsimony*. It states that if we have several theories for describing a solution we should opt for the one with the smallest number of concepts. Since current metamodeling languages do not provide a real problem analysis of the metamodeling activity we cannot apply the principle of parsimony to them. The lack of clear problem prevents evaluating the adequacy of these languages.

This paper outlines an approach that needs a further elaboration in many directions. We plan to extend OGML by providing a formal semantics to it. A formal definition of the term ontological commitment is also required. Furthermore, performing

case studies of expressing real modeling languages in terms of OGML is a must. We plan to experiment with expressing the OWL language [17] as an OGML metamodel.

The views on metamodels as definitions of structures or ontological commitments are not mutually exclusive. Clearly, the latter implies the former. The main statement of this paper is that a metamodeling language should not define arbitrary structures. The structures should be systematically derived based on a sound knowledge about metamodeling.

7 Acknowledgments

I would like to thank Klaas van den Berg for the useful feedback that helped me to improve the paper. Many of the ideas were conceived as a result of my discussions with Jean Bézivin and Frédéric Jouault on the nature of metamodeling.

References

1. Armstrong, D.M. Universals: an Opinionated Introduction. Westview Press, 1989
2. Atkinson, C., Kühne, T. Model-driven development: a metamodeling foundation. IEEE Software, 20(5), pp. 36-41, 2003
3. Beckett, D. RDF/XML syntax specification. W3C Document, 2003
4. Bézivin, J., Lemesle, R. Ontology-based layered semantics for precise OA&D modeling, In ECOOP'97 Workshop Reader, Finland, 1997
5. Bowers, S., Delcambre, L. On modeling conformance for flexible transformation over data models. In Proceedings of the Workshop on Knowledge Transformation for the Semantic Web at the 15th European Conference on Artificial Intelligence (KTSW-2002), Lyon, France, 2002
6. Eclipse Modeling Framework http://www.eclipse.org/emf
7. Guizzardi, G. Ontological Foundations for Structural Conceptual Models. PhD thesis. University of Twente, 2005. ISBN 90-75176-81-3
8. Gruber, T. R. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human and Computer Studies, 43(5/6): 907-928, 1995
9. Heller, B., Herre, H. Ontological Categories in GOL. Axiomathes 14: 71-90, Kluwer Academic Publishers, 2004
10. Jouault, F., Bézivin, J. KM3: a DSL for Metamodel Specification. FMOODS 2006, Bologna, Italy, 14-16 June 2006
11. Jouault, F., Bezivin, J, Kurtev, I. TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering. GPCE2006, Portland, Oregon, USA. October 2006
12. Kurtev, I., Bezivin, J., Jouault, F., Valduriez, P. Model-based DSL Frameworks. OOPSLA 2006 Companion Proceedings. 2006
13. Loux, M.J. The problem of universals. In Metaphysics: contemporary readings. M.J. Loux (ed.). Routledge, 2001
14. OGML definitions web site. http://wwwhome.cs.utwente.nl/~kurtev/OGML/
15. OMG/MOF Meta Object Facility (MOF) Specification. OMG Document AD/97-08-14, September 1997. Available from www.omg.org
16. Seidewitz, E. What Models Mean. IEEE Software, 20(5), 2003
17. W3C. OWL Web Ontology Language Overview. 2004