

Abstract metamodeling, I: How to reason about meta- and metamodels in a formal way

Zinovy Diskin*

Frame Inform Systems, Ltd.
Riga, Latvia
zdiskin@acm.org, diskin@fis.lv

Abstract

The paper suggests a formal framework to reason about metamodeling in a way independent of specifics of one or another metamodel and so encompassing such distinct metamodels as, say, relational, UML and XML. To this end, the notion of specification (schema, model) and its instances is described in an entirely abstract way like mathematical structures are described.

The key novelty of the framework is that mappings between specifications are involved and, moreover, play the key role in formal explicating of metamodeling constructs. Also, the notion of metamodel morphism is defined, which formally explicates such well known transformations as mapping ER-diagrams into relational database schemas or UML-diagrams into code in some OO-programing language. Particularly, the notion makes it possible to describe the UML architecture by precise and formalizable yet comprehensible graphic diagrams.

1 Introduction: AMM vs. UML

Relational data (meta)model is well defined in a precise way, both syntactically and semantically, and reasoning within RDM is, in fact, mathematics. Much less precision one finds in semantics of ER-based models and even less in semantic definitions of various OO-models but it does not prevent reasoning about models within ER or OO frameworks. Of course, the deficiency of precision on the base semantic level may result in poor reasoning about these models but this problem is not in focus of the present paper. Leaving specifics of one or another metamodel aside, we will address the following question: is it possible to reason about models *in general*, beyond specifics of particular metamodels (like RDM, IDEF1X, UML) and even beyond different approaches to business modeling (like ER or OO)? Is it possible to define what is a data or behavior schema and its instances in abstract way? And if so, is it possible to explicate within the framework set up by such a definition the notions of view, query, process or event in some abstract way generalizing instances of these notions in particular metamodels like above?

*Supported by Grants 93.315 and 96.0316 from the Latvian Council of Science

The questions above present a non-trivial mathematical problem but motivation of the paper is not in this challenging exercise in formalization. It seems that these questions are really posed by the current state-of-the-art in business modeling. Indeed, a modern business or software environment may be extreme heterogeneous and so one has an extreme heterogeneous model space: it is populated by models built in so conceptually distinct frameworks as ER, OO and, say, XML, and by many models built on mixed foundations, and by a lot of "this organization"/"this-vendor-tools" models. In addition, one often needs to relate these models between themselves and to reason about their interoperation.

Basically, there are two principally different approaches to manage the issue. One is to avoid or reduce heterogeneity by suggesting one universal metamodel to be used everywhere, the UML is an industrial strength effort on this way. The other approach is to preserve heterogeneity but to view all the models from a single perspective as particular instances of some general construct of "abstract model". Being developed, this latter approach should give rise to a language for reasoning about models in an abstract way, we may call this imaginary language *the AMM – Abstract MetaModeling*. Thus, while the UML approach reduces the universe to a point (or a dense accumulation of points), the AMM preserves diversity but organizes it into a single conceptual whole. Though the AMM approach may seem to be too theoretical, the following passage from practice oriented forum [6] advocates the AMM rather than the UML approach:

... it is clear that the problem of identifying and resolving semantic heterogeneity in a heterogeneous database environment is far from solved, and that the problem itself is still at the stage of being understood. ... Wittgenstein said long ago that if an idea was worth discussing, then it was worth having a language to discuss it. Thus, researchers need to develop languages for handling data interoperation. ... *Perhaps, one of the most difficult problems in developing integrated systems is in figuring out what schemas, data, and application code mean.*

It is remarkable, and characteristic to the problem we discuss, that the "abstract" Wittgenstein's philosophy was invoked at a conference organized by practitioners.

Well, at first glance it seems impossible to speak in a precise way about, say, database schemas and their instances without a specific description of what they are, it would be a kind of substantial speaking about nothing. Nevertheless, a methodological framework and a machinery for such a strange thing are already developed in the mathematical category theory (CT), which was even entitled by *abstract nonsense* during its early days. And indeed, it was demonstrated in the predecessor of the present paper [4], and will be continued in the present one, that the CT-methodology does make it possible to develop AMM.

A more concrete source of motivation for the present paper is in the recent OMG's document set on the UML and MOF (below referred as *OMGdoc*) where general metamodeling and meta-metamodeling considerations form an important (at least, methodologically) fragment. Further we will use terminology close to that in *OMGdoc* and usually referred to as the four-layer OMG architecture for metamodeling¹; meta-metamodeling will be abbreviated by *meta2modeling*. The *OMGdoc*'s metaconsiderations are highly informal and (as it often happens with informal reasoning on logical matters) result in some gaps and incorrectness that become evident within some simple formal framework. For example, even an initial formal explication of the issue shows that the constructs that are called meta-metamodels in the UML and the MOF documentation are related to, respectively, the UML and MOF metamodels in quite different way than these metamodels are related to models. So, if one accepts the definition of meta2model stated in the OMG four-layer architecture, then no UML or MOF meta2models are built in *OMGdoc* (and, in fact, cannot be built on the OMG's level of reasoning because of meta2modeling is a much more sophisticated issue than assumed in *OMGdoc*). Fortunately, our analysis will show that actually the construct of meta2model (in its precise sense) is not necessary for the goals *OMGdoc* is aimed at. What is really needed for making metamodeling considerations in *OMGdoc* precise is a kind of AMM and indeed,

¹before this paragraph we have used the term model for both proper models (data or behavior specifications) and metamodels (model specifications), further we will try to be more accurate

even our initial AMM-based treatment makes it possible to shorten some of valid considerations (few algebraic expressions instead of a few paragraphs), remove some non-correct considerations and fill-in some of the gaps in the presentation. Unfortunately, the AMM-idea is entirely missed in OMGdoc (and it is not surprising because of no suitable language is familiar to the community). The present paper is just aimed at filling the gap and our plan is as follows.

We begin with a very simple abstract and formal notion of modeling system (metamodel) in section 2. Despite its extreme simplicity, it allows to rephrase the main concepts of the four-layer architecture in a formal way and at once clarifies what is done in OMGdoc in this respect. In section 3 the framework is enriched with a key construct of mapping between specifications and the benefits are discussed in section 4. In section 5 morphisms between modeling systems as whole entities are introduced, which organizes the universe of metamodels into an integral structure. In particular, with metamodel morphisms one can specify the UML's architecture by graphic diagrams in a brief and comprehensible yet precise way. In Appendix A a simple example of metamodeling is considered rigorously to illustrate main concepts proposed in the paper. In Appendix B modeling systems are related to a very close to them formal framework of the so called *institutions* invented by Goguen and Burstall for AMM of logical systems that have been studied in mathematics and theoretical computer science.

Finally, some remarks on the technicalities used in the paper. Basically, they are quite elementary and do not go beyond the framework of the following notions (treated in an intuitive way): sets (classes) and their elements, subsets, mappings between sets, composition of mappings, partial order on a set. No any special knowledge is needed to understand definitions and manipulations though, of course, some experience in set-theoretical manipulations would be useful, especially in tracing "higher-order" constructs like a mapping whose values are sets, or a mapping defined on a set of mappings, or compatibility of a mapping with operations defined on the domain of this mapping.

All formal definitions in the paper consist of two parts: first, a system of sets and mappings between them is defined, and then special *commutativity* conditions of matching mappings between themselves are formulated. Commutativity is of vital importance for enabling the entire machinery to work properly when one manipulates the constructs but for understanding the approach and ideas one may omit tracing these conditions.

In general, the technical framework used in the paper is based on what can be called elementary naive category theory. However, this refers to the spirit rather than machinery: no mathematical results will be used and, the more so, no theorems will be proven. The goal of the paper is just to make precise some informal conceptual framework that is commonly used in meta- and metamodeling considerations in software engineering literature.

Acknowledgement. The author is indebted to Ken Baclawski and Haim Kilov for useful discussion of the material displayed in the paper and its presentation.

2 Specifications and their instances formally: Getting started

Let U be some universe of discourse (say, a view to a business). Normally, U is not static and evolves through a set of states, $SttU$. In the terminology adopted in OMGdoc, a *model* of U is some specification S s.t. U -states can be considered (semantic) instances of S .

To formulate this description more formally we should first define what are specifications and their instances. However, in our general setting it would be hardly possible to define what are specifications and instances in some *analytical* way, that is, by pointing out from what details they consist of. Instead, we will take the *synthetical* approach and define what we would have after the notions are somehow defined.

2.1 Preliminary definition. *A(n abstract) modeling system* consists of the following two components:

- a collection of abstract objects called *specifications*²;
- a mapping that assigns to each specification S a collection of objects called *instances* of S , $Inst(S)$.

For specifications we will also use the term *schema* that matches well the syntactic nature of the notion. Correspondingly, the collection of the specifications will be denoted by *Schema*.

Thus, a modeling system is defined to be a pair $Mod = (Schema, Inst)$ with *Schema* a class and *Inst* a mapping like described above. No further assumptions about the nature of schemas and instances are made. \diamond

A classical example of modeling system is the relational data model (RDM) whose schemas are relational database schemas and instances are database states. Another example is the ER data model whose schemas are ER-diagrams with their intended meaning as instances, and one more example of modeling system is the Petri nets – its schemas are Petri nets (diagrams) as such and instances are behaviors they specify. The UML is also an example of (monstrous) modeling system whose schemas are UML-diagrams and instances are described in the UML-semantic volume.

2.2 Basics of modeling, accurately. Given a universe U with intended set of states $SttU$, a *model* of U is a pair $M = (Mod, S)$ with Mod a modeling system as above and S a Mod -schema, $S \in Schema$, s.t. U -states can be considered as S -instances and a one-one correspondence between $Inst(S)$ and $SttU$ holds.³

Note, it is common to call a model of U just a schema S without mentioning Mod . However, S gets its power of model only in the context of the entire modeling system Mod . Indeed, to specify something one should first set a specification tool, a system Mod in our case, and only then build a schema of that thing. So, speaking accurately, the notion of model differs from the notion of schema and its first component, though often implicit, is essential. \diamond

Despite its extreme simplicity, definition 2.1 above allows to start to reason about specifications formally.

2.3 Ordering specifications. A specification S_1 will be called *more specific* than a specification S_2 or, the same, S_2 is *more general* than S_1 , we write $S_1 \prec S_2$, iff $Inst(S_1) \subset Inst(S_2)$. A typical case when this relation appears is when one adds to a schema S new constraints C , clearly, $(S + C) \prec S$ where we write $S + C$ for the augmented schema. Obviously, relation \prec is a partial order on the set *Schema*.

On the other hand, we can enrich definition 2.1 of modeling system by postulating another partial order on the set *Schema* from the very beginning. This order is to be thought of as the partial order of inclusion of one schema into another: we write $S \subset S'$ and informally understand this as that S is a subschema of S' . For example, if S' is a relational database schema and S is S' from which some relations are removed, then $S \subset S'$. This intuition induces that any instance I of S' , $I \in Inst(S')$, can be also considered as an instance of S if one forgets about the part of I assigned to "the difference" $S' \setminus S$. So, if $S \subset S'$ then there is some reduction mapping $red: Inst(S') \rightarrow Inst(S)$.

Thus, if we postulate some partial order of *inclusion* on the set *Schema*, then we should also postulate a *reduction* mapping $red_{S \subset S'}: Inst(S') \rightarrow Inst(S)$ for each pair $S \subset S'$. These two constructs – partial order and the reduction mappings – explicate formally the intuitive notion of being subschema but, of course, only in part. Further we will see how to manage these considerations in a more general and systematic way. \diamond

2.4 Basics of metamodeling, accurately. How can one set a modeling system $Mod = (Schema, Inst)$? One way is to list all objects of *Schema* and, for any schema S , all objects of the set $Inst(S)$. This way may be good for finitary modeling systems but practically useful modeling systems normally have the sets above infinite (and as for instances, even non-enumerable). In such

²but if one prefers she might call them *cups*, or *cats*, or somehow else as well

³To state this formally we would need to formalize what is U and $SttU$ which is not necessary for our goals.

a case the only way is to present precise definitions (templates) of what is a schema and what are its instances, and then schemas and instances are exactly those objects that satisfy the definitions (match templates). It may seem that this necessary turns us back to the analytical approach but actually we can present the construction in question synthetically. To this end, we consider the very modeling tool, the system *Mod*, as a special universe of discourse that has two different kinds of "states" – schemas and instances, and specify them within some new meta-modeling system. That is, our meta-specification has to consist of two meta-schemas, say, **No** for *Mod*-schemas (*Mod*-notation) and **Se** for *Mod*-instances (*Mod*-semantics), which have to be built within their own modeling systems, say, **Mod**₁ and **Mod**₂.

Formally, for defining *Mod*-notation, a (meta) modeling system **Mod**₁ = (**Schema**₁, **Inst**₁), has to be set and schema **No** ∈ **Schema**₁ is called a *meta-schema* for *Mod*-notation if **Inst**₁(**No**) = *Schema*. In particular, it is possible to take *Mod* for **Mod**₁, and it is indeed often done in metamodeling⁴. In Appendix a simple example is considered to illustrate how it works.

As for specifying semantics of *Mod*, one more modeling system should be set, **Mod**₂ = (**Schema**₂, **Inst**₂), with the peculiarity that its schemas may be parameterized by objects of *Schema*, that is, have the form **S**[*S*] with *S* ∈ *Schema*. Then, a schema **Se**[*S*] ∈ **Schema**₂ is called a *meta-schema* for *Mod*-semantics if **Inst**₂(**Se**[*S*]) = *Inst*(*S*) for any schema *S* ∈ *Schema* (we again refer to Appendix for an example). Of course, formal specifying semantics is much more sophisticated than formalizing syntax. For example, for OO modeling, **Se**[] should specify formally what is a class, object, operation, inheritance and so on, which is still an open research problem.

Anyway, formally, given a modeling system *Mod* = (*Schema*, *Inst*), its *metamodel* is a quadruple **M** = (**Mod**₁, **Mod**₂, **No**, **Se**) with meaning of the components as described above. We can present this quadruple as pair **M** = (**Mod**, **S**) with the first component presenting the modeling tools, **Mod** = (**Mod**₁, **Mod**₂), and the second one is the meta-schema as such, **S** = (**No**, **Se**). The reservation about importance of the first component made above for models is valid for metamodels as well.

Meta-metamodeling layer is built over the metamodeling layer in exactly the same way as meta-modeling is built over modeling. For modeling each of the metasystems **Mod**_{*i*}, *i*=1,2, two new (meta2)modeling systems, say, **Mod**₁ and **Mod**₂ should be set so that the modeling tools of the meta2layer comprise four modeling systems.

Our considerations are summarized in the table 1 below.

◇

2.5 Four-layer meta-architecture in the light of AMM. Let us consider meta- and metamodeling as they are described in OMGdoc, in particular, their four-layer architecture, in the framework displayed in the table.

First of all, the opposition of universe and its schema is fundamental for the modeling idea as such, it presents on each of the modeling levels and it is somewhat incorrect to consider a universe on level 1 while its schema (model) on level 2 as is set in the four-layer architecture. In other words, this opposition is visualized by different columns going across all the rows-layers rather than by different rows.

Another fundamental point pertaining to all modeling levels is that a model is actually a pair **M** = (modeling tools, schema), not only schema as such – this aspect is not clearly recognized in OMGdoc and, in fact, just this gap leads to mistakes in OMGdoc's meta-considerations. Nevertheless, the notions of metamodeling in OMG and in our framework are basically the same but, of course, no modeling system for specifying semantics (**Mod**₂ in our notation) is built in OMGdoc. Actually, OMGdoc's metamodel is a triple (**Mod**₁, **No**, *Sem*) where the first two components are close to those in our framework and *Sem* is some informal description of semantics (like, for example, in the UML semantic volume); we can well consider that *Sem*'s in the UML and MOF documents are some informal description of the mappings *Inst* for the UML and MOF modeling systems.

⁴Of course, to avoid logical circle some delicate conditions must be respected but the accurate treatment of the issue goes beyond the paper framework

Table 1: Layers of metamodeling, accurately

Modeling Level	Universe to be modeled	Model		Condition of schema suitability
		Modeling tools	Schema	
Modeling	$SttU$	$Mod = (Schema, Inst)$	S	$Inst(S) = SttU$
Metamodeling	Mod	(Mod_1, Mod_2) $Mod_i = (Schema_i, Inst_i),$ $i = 1, 2$	$S = (No, Se)$	$Inst_1(No) = Schema,$ $Inst_2(Se[S]) = Inst(S)$
Meta-metamodeling	(Mod_1, Mod_2)	$(Mod_i, i = 1..4)$	$S = (S_1, S_2)$ $S_i = (No_i, Se_i),$ $i = 1, 2$...

As for meta-metamodeling, it is immediately seen from our explication that the notion of meta2model is much more complicated than it is supposed in the OMGdoc: a meta2model has to specify the entire modeling systems Mod_1 and Mod_2 (see the table above) rather than only schemas No and Se as is (intended to be) done in OMGdoc . Clearly, no such a construction is described in OMGdoc (and cannot be described simply because of no metamodeling system for specifying semantic is described there: semantics is described informally in the natural language). Fortunately, a meta2model is not really needed for the OMGdoc goals at all as we will see below. \diamond

And after all, what is the nature of the constructs that are called meta2models in the UML/MOF documentation?

2.6 What is called a meta-metamodel in OMG's UML/MOF documents?

Let us suppose that we have two metamodels, $\mathcal{M}, \mathcal{M}'$, for modeling systems Mod, Mod' respectively. We say that \mathcal{M} is *more specific* than \mathcal{M}' (or \mathcal{M}' is *more general* than \mathcal{M}), $\mathcal{M} \prec \mathcal{M}'$, iff

$$Inst_1(No) = Schema \subset Schema' = Inst_1'(No') \text{ and}$$

$$Inst_2(Se[S]) = Inst(S) \subset Inst'(S) = Inst_2'(Se'[S])$$

for any $S \in Schema$. Clearly, it is in fact relation between modeling systems themselves and we can well write $Mod \prec Mod'$ without referring to metamodels \mathcal{M} 's.

Now, the analysis of the UML/MOF documentation in the framework displayed shows that what is called there a meta2model is a construct of the following type: it is nothing but some very general OO metamodel, say, \mathcal{M}_{OO} , s.t. that $\mathcal{M}_{UML} \prec \mathcal{M}_{OO}$ or, equivalently, $UML \prec OO$. In more detail, No_{OO} is presented as some repository of modeling elements (with their intended semantics explained informally) from which the UML models are to be built. Then, automatically, $Schema_{UML} \subset Schema_{OO}$ and $Inst_{UML}(S) = Inst_{OO}(S)$ for any UML-schema S .

It seems that the MOF metamodel was designed just to play the role of \mathcal{M}_{OO} (rather than meta2model as is stated in OMGdoc). However, some modeling constructs in UML cannot be

neither found nor built from those in MOF (it is explicitly pointed out in the documentation), that is, the subordination $\text{UML} \prec \text{MOF}$ does not hold. Rather, we have two inequalities $\text{UML} \prec \text{OO}$ and $\text{MOF} \prec \text{OO}$, and maybe the least upper bound (in the \prec -ordering of metamodels) of UML and MOF is nothing but that hypothetical most general OO-model OO. Informally, this means that the UML and MOF metamodels together provide all the constructs needed in the OO-modeling. Of course, we cannot check whether this latter statement is true or not because of we have no formal definition of what is OO-modeling system. Incidentally, we can *define* OO by the equality $\text{OO} \stackrel{\text{def}}{=} \text{UML} \vee \text{MOF}$ where \vee denotes the least upper bound in the \prec -ordering of the metamodel space. It seems such a definition matches the spirit of the OMG standardization effort. \diamond

2.7 What do we actually need for reasoning about metastuff in a heterogeneous business/software environment?

Well, what was described above brings the basics of OMGdoc meta-considerations to light. However, practical needs in precise metareasoning go beyond the framework set in OMGdoc, in particular, beyond (meta)modeling of OO-metamodels. As it was said in section 1, what is really needed for reasoning about conceptually and semantically heterogeneous business/software environment is some general abstract notion of metamodel whose instances would be various concrete metamodels like, say, the UML, MOF, XML, ER or RDM.

In fact, the approach was already presented above in our (pre)definition 2.1. However, to make the notion a really working construction, the definition should be developed and, at least, three additional key constructs should be added.

The first one is to consider, along with specifications, their mappings as equally important construct – this will be addressed in the next section.

Also important is that instances of a schema are not independent entities but are mutually related by possible transformations, for example, permutations of domain elements transform database states. Such transformations can be considered as mappings between instances and they play a key role in semantic consideration (few additional words will be said in section 3.4).

Finally, a principle development of the framework set by definition 2.1 is to enter into consideration operations which, syntactically, augment any given specification with new items denoting derived information and, semantically, extract derived information from any instance (Codd’s relational algebra is a classical example of such operations set). The issue is of vital importance for the modeling idea as such but its value is often underestimated in literature and, unfortunately, in practice of modeling too. For example, no in any way systematic treatment of this key aspect of modeling can be found even in such an all-embracing framework as the UML. Unfortunately, space limits do not allow even a little discussion of the operation aspects, details can be found in [3] (and in the companion paper [2]). \diamond

3 Developing the framework: Mappings between specifications and instances

3.1 Motivation and preliminaries. Let $Mod = (Schema, Inst)$ be a modeling system as defined above. Generally speaking, having schemas (specifications) one would like also have the possibility to relate them between themselves. For example, one schema can be a part of another schema, or a view to it, or refinement or interpretation of it, and so on. So, we need some machinery entered in our AMM framework, which would make it possible to relate schemas. The most general way of setting such a machinery is to assume that together with schemas there are given *schema mappings* or *morphisms*. Intuitively, if schemas can be thought of as relational database schemas or, say, the UML class diagrams, their morphisms can be thought of as mappings between schemas sending items of the source schema into similar items of the target schema so that basic structural

relationships between items are respected (for example, the image of an association between two classes is an association between the images of the classes).

Formally, we assume that together with abstract objects called *schemas* are given other abstract objects called *schema morphisms* or *mappings* s.t. with each schema mapping f there are connected its *source* schema, $\square f$, and its *target* schema, $f\square$. This is written as $f: S \rightarrow S'$ if $S = \square f$ and $S' = f\square$;

The data above can be presented as a directed graph whose nodes are schemas and arrows are mappings (there can be many arrows between two nodes and actually the graph is a multigraph). In addition, there is a binary operation of composing arrows which satisfies certain conditions (associativity and presence of the unit). Such graphs are called *categories*.

So, our first step in developing the metaframework is to constitute a category *Schema* of abstract objects called *schemas* and *schema mappings*, and this is all that we need to know about schemas.

◇

3.2 Examples. A simple example of *Schema* is the category $REL(\mathcal{D})$ of relational schemas over some predefined collection of value domains $\mathcal{D} = \{D_1, \dots, D_k\}$. In more detail, an object of $REL(\mathcal{D})$ is defined to be a finite set of relation schemes coupled with functional dependencies. Given two such objects, *ie*, relational schemas S, S' , a morphism $f: S \rightarrow S'$ is a mapping which assigns to each relation scheme $R[A_1, \dots, A_n]$ in S some relation scheme $R'[A'_1, \dots, A'_n]$ in S' so that domains of A_i and A'_i coincide. In addition, if a pair $(R[A_1, \dots, A_n], X \rightarrow Y)$ with X, Y subsets of $\{A_1, \dots, A_n\}$ is declared to be a functional dependency in S then its f -image, the pair $(R'[A'_1, \dots, A'_n], X' \rightarrow Y')$, must be a functional dependency in S' .

For another example of *Schema* we would like to have some graph-based metamodel like ER or some closed fragment of the UML, say, the static one. So, let objects of *Schema* will be all *class diagrams* in the UML. For a morphism of such diagrams, $f: S_1 \rightarrow S_2$, we take a mapping that has the following properties.

- Classes are mapped into classes so that if C is a class (name) in S_1 with attributes a_1, \dots, a_k and operations o_1, \dots, o_m then its image in S_2 is some class $B = f(C)$ and the sets of C -attributes and C -operations are mapped into the sets of B -attributes and B -operations respectively, in addition, attributes a_i and $f(a_i)$ must have the same domain and operations o_j and $f(o_j)$ be of the same arity shape.
- If some class G is declared to be a generalization of classes C_1, \dots, C_n in S_1 then the class $f(G)$ must be a generalization of $f(C_1), \dots, f(C_n)$ in S_2 .
- Associations are mapped into associations and role names into role names so that if

$$\boxed{C_1} \xrightarrow[r_2]{r_1} \boxed{C_2}$$

is an association in S_1 , its image in S_2 is an association

$$\boxed{f(C_1)} \xrightarrow[f(r_2)]{f(r_1)} \boxed{f(C_2)}$$

- Constraints declared for associations must be respected in the following way. If a constraint c is declared for the association end (role) r in S_1 , then the same, or stronger, constraint must be declared for the association end $f(r)$ in schema S_2 . For example, if r is stated to be an aggregation end, $f(r)$ has to be also declared as an aggregation or composition end. Or, if some multiplicity range $m1..m2$ is specified for r , then a range $n1..n2$ with $m1 \leq n1 \leq n2 \leq m2$ must be specified for $f(r)$.
- And so on.

Here "so on" means that the list must be continued (along the guidelines just displayed) for other items that may appear in the UML class diagrams. The general idea is that

- items of some kind are mapped into items of the same kind so that incidence between them is respected (like it was described above for mapping binary associations);
- if a constraint c is declared for a configuration D of items in the schema S_1 , then the same or stronger constraint c' that logically entails c , $c' \Rightarrow c$, must be declared for the image configuration $f(c)$ in schema S_2 .

On a whole, these conditions must guarantee that if $f: S_1 \rightarrow S_2$ is a schema morphism and so S_1 is somehow interpreted in S_2 in a correct way, then any instance of S_2 can be considered as an instance of S_1 via the interpretation f . This consideration becomes much clearer if we treat instances as special mappings which send items of schemas to their semantic meanings: class names to classes, binary association lines to sets of binary links *etc* so that all the constraints specified are respected. Then an instance of, say, schema S_2 is a mapping $I: S_2 \rightarrow U$ where U denotes special schema whose items are semantic constructs themselves, for example, class names in U are classes as such, associations are sets of links (whose multiplicities are real multiplicities rather than constraint specifications), operations are operations as such – actions rather than their specifications and so on. Then, because of f is a schema morphism as they were specified above, the composition of mappings f and I ,

$$f \triangleright I: S_1 \rightarrow S_2 \rightarrow U,$$

will be also a mapping that assigns semantic meanings to S_1 -items in such a way that all the constraints specified in S_1 are respected. Thus, if schema mapping $f: S_1 \rightarrow S_2$ is a morphism, then there is defined a mapping $f^*: Inst(S_2) \rightarrow Inst(S_1)$; note the reverse of the direction. \diamond

What we have just described in the context of the UML class diagrams is a quite general consideration applicable for any metamodel: for example, other fragments of the UML, or ER and its various extensions, or the relational data model (see also Appendix A for a simple yet rigor, and somewhat generic, example of how to treat instances as mappings). A lot of similar examples can be also found in the institution theory mentioned in introduction.

So, irrespectively to specifics of any given metamodel, syntactic conditions a schema mapping must satisfy in order to be a schema morphisms should guarantee that any morphism $f: S_1 \rightarrow S_2$ generates an instance mapping $f^*: Inst(S_2) \rightarrow Inst(S_1)$. This motivates the following formal

3.3 Main definition (still incomplete). *A(n abstract) modeling system* is a pair $Mod = (Schema, Inst)$ with

- *Schema* a category of *schemas* and *schema morphisms*,
- *Inst* a mapping that assigns

- to each object $S \in Schema$ a collection of objects called *instances* of S , $Inst(S)$, and

- to each morphism $f: S_1 \rightarrow S_2$ a mapping $Inst(f): Inst(S_2) \rightarrow Inst(S_1)$; note the reverse of the direction. It is convenient to designate $Inst(f)$ simpler by, say, $f^*: Inst(S_2) \rightarrow Inst(S_1)$.

In addition, the mapping *Inst* is compatible with composition of schema mappings: for any arrows $f: S_1 \rightarrow S_2$, $g: S_2 \rightarrow S_3$ in *Schema*, equality $(f \triangleright g)^* = g^* \triangleright f^*$ holds, that is, for any instance $I \in Inst(S_3)$, $I.g^*.f^* = I.(f \triangleright g)^*$.

Categorically, we can formulate the last two conditions shorter by saying that *Inst* is a *functor* $Inst: Schema \rightarrow \mathbf{Set}^{\text{op}}$ from the category *Schema* into the category **Set** of sets and functions, the index **op** points that arrows are reversed.

Note, the case when one schema is part of another, $S_1 \subset S_2$, amounts to a special *inclusion* mapping between them, $i: S_1 \rightarrow S_2$, which is, in fact, the identity mapping of S_1 , and then $i^*: Inst(S_2) \rightarrow Inst(S_1)$ is that reduction mapping we discussed in section 2.2. \diamond

3.4 Mappings between instances. More careful analysis shows that for any schema $S \in Schema$ there are morphisms between instances of S , for example, in the RDM, permutations of domain elements transform database states. So, the collections $Inst(S), S \in Schema$, are also categories rather than plain sets and mappings $Inst(f)$ are functors (mappings between categories)

rather than functions. Instance morphisms are extremely important in semantic considerations, in particular, their role is well known in the relational database theory. Of course, they are also important in other data and behavior models but their value often remains implicit because of poor semantic foundations of these models (compare with RDM which was well semantically formulated from the very beginning). Unfortunately, considering this issue goes far beyond the present paper framework.

Also, instances normally have some complex internal structure which however can be externally captured by instance morphisms. This is done in a standard categorical way somewhat similar OO's encapsulation when object's internal structure is set via interfaces; in a sense, category theory is based on arrow interfaces to objects. So, instance morphisms can be considered as arrow interfaces to instances, which make it possible to explicate their internal structure.

Finally, in metamodeling, instances of a metaschema are schemas and, as we have just seen, schema mappings, *ie*, instance morphisms, are important. Thus, we should include instance morphisms into our general framework. \diamond

3.5 Main definition, completed. A (*n abstract*) modeling system is a pair $Mod = (Schema, Inst)$ with

- *Schema* a category of *schemas* and *schema morphisms*,
- *Inst* a functor that assigns to each object $S \in Schema$ a category of objects called *instances* of S , $Inst(S)$, and each morphism $f: S_1 \rightarrow S_2$ is assigned with a functor $Inst(f): Inst(S_2) \rightarrow Inst(S_1)$.

In the arrow notation, this can be written as $Inst: Schema \rightarrow \mathbf{Cat}^{op}$ where \mathbf{Cat} denotes the category of categories and functors and the index *op* points that arrows are reversed. \diamond

4 Discussion: Specification mappings – what do we get with them?

In the definition of modeling system, passing from considering the collection *Schema* merely a class to considering it a category has far reaching consequences.

First of all, mappings between schemas organize and connect them into an integral universe. Particularly, from this view point the UML is not a single modeling system but rather a collection of modeling systems. In more detail, each kind of the UML's diagrams – class diagrams, statechart diagrams, use-cases and so on – should be organized into a category $Schema_i$ and their intended semanticses determine the mappings $Inst_i$ (by [10, p.24], $i = 8$). Then the UML is the set $\{Mod_1, \dots, Mod_8\}$ with $Mod_i = (Schema_i, Inst_i)$ a modeling system as described above.

Second. The experience of category theory shows that if objects of some kind are organized into a category, the latter forms a natural universe for precise treatment of various relations between these objects and operations over them. For example, one should be able to consider subschemas of a given schema, their intersection, union and difference, their images and coimages under a given mapping from/into the schema *etc*. It should be also desirable to have the possibility to consider Cartesian products of schemas, relations between schemas, integration of schemas modulo some correspondence between them and so on. In other words, it is desirable to perform with *Schema*-objects and their mappings all operations which make sense for sets and functions (or for graphs and graph mappings, or for sets and relations between them and the like).

A surprising discovery made in category theory is that it is possible to specify all these and others relations and operations on objects entirely in terms of arrows between them without any referring to object's internal structure. In more detail, it was developed the notion of category called *topos* that provides very rich specification facilities and allows to emulate all set-theoretical constructs: it was proven that any object or manipulation with objects which can be specified formally can be specified within the topos formalism as well or, speaking semantically, can be performed within any topos. So, as soon as the category *Schema* in the definition of modeling system is taken to be

a topos, one can well suppose that any formalizable manipulation with schemas is possible within *Schema*.

Third. It was shown in [4] that mappings between schemas make it possible to explicate in a formal and quite general way the two fundamental relations between specifications: (i) a schema is a view to another schema, and (ii) a schema is a refinement of another schema. However, these mappings are mappings between schemas augmented with derived items (denoting derived information, see 2.7) and so considering them goes beyond the present paper framework.

Forth. Considering morphisms between some class of objects to be defined often helps to build a correct definition. Category theory teaches us that to define properly some universe one is going to deal with, one must define mappings between objects of the universe. So, if you have difficulties in defining mappings between objects then, most probably, something is wrong in the definition of these objects. A characteristic example in this respect is the famous ER-diagrams. There are several precise definitions of ER-diagrams of one or another kind but no correct definitions of their mappings can be find in the literature, and this is not accidental. Indeed, a natural ER-diagram mapping should send rectangle nodes (entities) to rectangle nodes, and diamond nodes (relationships) to diamond nodes. However, in such a setting the well known, and practically important, phenomenon of semantic relativism is ruled out. If, on the other hand, one allows interpreting rectangles by diamonds and vice versa, then the crucial for ER data modeling distinction between entities and relationships will disappear. It seems that a similar story can well appear with mappings between the UML-diagrams (of some fixed kind). In other words, an extremely useful test on correctness of the UML diagrams definitions would be an attempt to define mappings between them, particularly, in such a way that allows semantic relativism.

And more generally, reformulating some subject of reasoning in the arrow framework reveals or essentially rearrange its structure and so clarifies many aspects of the subject and makes them explicit: in a sense, arrowless treatment is somewhat similar to a one-dimensional (only objects) view on a two-dimensional (objects and arrows) figure. In addition, the arrow formulation makes it possible to apply a powerful graph-based algebraic machinery – the famous diagram chasing – which facilitates greatly manipulation with constructs (see Goguen’s ”Manifesto” [7] for details and examples).

5 Universe of metamodels: Morphisms between modeling system.

Up to here we were discussing individual metamodels/modeling systems (section 2.6 is the only exclusion). However, modeling systems appearing in practice are not independent and can be related in many ways. For example, ER-diagrams can be transformed into relational database schemas, and UML-diagrams into pieces of code in, say, Java. These relations between models reflect certain relations between metamodels: ER and RDM, UML and Java. Similarly to that we were doing above for relations between schemas within the same modeling system, we are now going to define morphisms between modeling systems as whole units. It will organize the space of metamodels into some integral universe.

5.1 Definition. Let $Mod_i = (Schema_i, Inst_i), i = 1, 2$ be two modeling system (metamodels). Their *morphism* $F: Mod_1 \rightarrow Mod_2$ is a pair of mappings (F_σ, F_ι) where

- $F_\sigma: Schema_1 \rightarrow Schema_2$ is a functor between categories, that is, a mapping sending nodes to nodes and arrows to arrows in such a way that a schema mapping $m: S \rightarrow S'$ in $Schema_1$ is sent into mapping $F_\sigma(m): F_\sigma(S) \rightarrow F_\sigma(S')$ in $Schema_2$. In addition, arrow composition must be preserved: $F_\sigma(m \triangleright m') = F_\sigma(m) \triangleright F_\sigma(m')$.
- F_ι is an assignment to any schema $S \in Schema_1$ a function $F_{\iota,S}: Inst_1(S) \rightarrow Inst_2[F_\sigma(S)]$ so that given some schema $S \in Schema_1$, any its instance $I \in Inst_1(S)$ is mapped into some instance $F_{\iota,S}(I)$ of $F_\sigma(S)$ in the system Mod_2 .

In addition, for any morphism $m: S \rightarrow S'$ in $Schema_1$, the following diagram holds commutative:

$$\begin{array}{ccc}
 Inst_1(S) & \xrightarrow{F_{\iota S}} & Inst_2[F_{\sigma}(S)] \\
 m^* \uparrow & & \uparrow [F_{\sigma}(m)]^* \\
 Inst_1(S') & \xrightarrow{F_{\iota S'}} & Inst_2[F_{\sigma}(S')]
 \end{array}$$

where commutativity means that $I.m^*.F_{\iota S} = I.F_{\iota S'}.[F_{\sigma}(m)]^*$ for any instance $I \in Inst_1(S')$.
 \diamond

5.2 Examples. It can be checked that with careful defining of the ER and RDM metamodels, the familiar transformation of ER-diagrams into relational database schemas is a metamodel mapping

$$rel = (rel_{\sigma}, rel_{\iota}): ER \rightarrow RDM.$$

The first component maps any ER-diagram $D \in Schema_{ER}$ into the corresponding relational schema $S = rel_{\sigma}(D) \in Schema_{RDM}$ and the second component maps any instance of the real world structured according to the diagram $W \in Inst_{ER}(D)$, into the corresponding database state $DB = rel_{\iota D}(W) \in Inst_{RDM}(S)$.

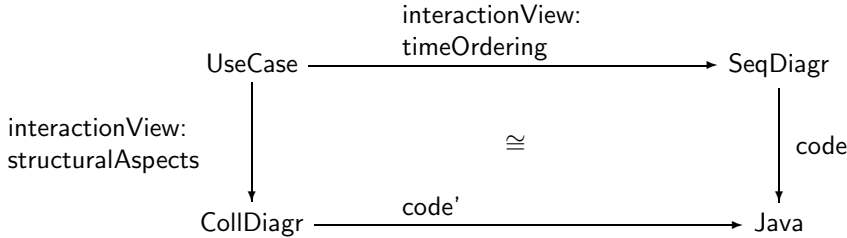
For another example, transformation of the UML's sequence diagrams into, say, Java-code is presented by a metamodel mapping

$$code = (code_{\sigma}, code_{\iota}): SeqD \rightarrow Java$$

where the first component maps any UML sequence diagram $D \in Schema_{SeqD}$ into the corresponding piece of code (program) $P = code_{\sigma}(D) \in Schema_{Java}$ and the second component maps any world behavior instance $W \in Inst_{SeqD}(D)$, into the corresponding state of the program during its execution, $PS = code_{\iota D}(W) \in Inst_{Java}(P)$.
 \diamond

5.3 What is the UML? At the beginning of section 4 we have described the UML as a collection of modeling system (metamodels). Now we can refine this description by enriching it with the metamodel morphisms. Of course, specifying UML's components as modeling systems in our sense and specifying UML's architecture in precise terms of mappings between modeling systems is an open research task. It needs careful checking a lot details (particularly, commutativity conditions), building some missed constructs and rebuilding some of the existing constructs in the UML definitions, and so is a task of the (nearest?) future. Below we will only illustrate how the arrow language can work in specifying the UML's architecture.

Let us consider the UML's fragment treating the use-case diagrams and both kinds of interaction diagrams, that is, sequence diagrams and collaboration diagrams. Description of their semantics and relations between them is highly informal in the UML definition but, probably, the following arrow picture explicates the intention in a more precise way:



Nodes of the graph denote modeling systems, arrows are their mappings as defined above, that is, transformation of not only syntactical schemas but, simultaneously, their semantic instances. Note

that the two different paths leading to Java-programs show that, in general, a use-case diagram D can be implemented in the two different ways by programs (pieces of codes) P_1 and P_2 . However, it is stated in the UML definition that sequence diagrams and collaboration diagrams are "semantically equivalent" [1, p.249], hence, the programs $P_1(D)$ and $P_2(D)$ should be also equivalent in a certain sense (that their execution leads to semantically the same result). A way of explicating this equivalence is to state that sets $Inst_{Java}(P_1)$ and $Inst_{Java}(P_2)$ are isomorphic in some precise sense (that, of course, should and could be defined but this goes far beyond the frame of the present paper). Thus, $P_1(D)$ and $P_2(D)$ are not equal yet isomorphic in a certain sense and so, what is called semantic equivalence of sequence and collaboration diagrams in the UML documents can be explicated by stating that the diagram above is commutative in a certain weakened yet precise sense, this is denoted by the marker \cong hung on the diagram. \diamond

6 Conclusion

A certain sculptor once said that anatomy must be learned and then may be forgotten. Probably, the same receipt is also good for reasoning (sculpture) on logically complex matters (human body): while formalizing it entirely is not necessary or useful, being aware and keeping in mind some basic formal framework (anatomy) underlying the subject can help in making reasoning correct and effective – without conceptual gaps and redundant constructs, and helps to present results in a compact and precise way.

It seems that samples of reasoning one can find in the current literature on metamodeling, in particular, in the recent OMG documents on UML and MOF, well demonstrate difficulties of sculpturing without knowledge of anatomy. The goal of the paper is thus not in formal constructs as such, and not in the intention to put the entire metastuff on the formal foundations; rather, the goal is to make reasoning on the subject more precise, consistent and hence more useful. Among practical recommendations our formal analysis suggests are the following ones:

- when defining a metamodel, consider schema mappings along with schemas, in particular, a good definition of schema should allow a natural definition of schema mapping;
- compare and manipulate metamodels through metamodel morphisms, the latter should involve not only syntactical but also semantic considerations as well;
- it is possible and useful to present a complex metamodel architecture as an arrow diagram of component metamodels and their morphisms.

The usefulness of these suggestions was practically checked on familiar data metamodels: RDM, ER-based and (static) OO. As for process/behaviour modeling, practical adaptation of the framework displayed in the paper needs further research but it definitely can go along the same lines as in data modeling. Particularly, the recent OMG document set on UML and MOF would benefit from adoption of the general suggestions above.

References

- [1] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language user guide*. Addison-Wesley, 1999.
- [2] Z. Diskin. Abstract metamodeling, II: How to specify derived information. In preparation.
- [3] Z. Diskin. Formalizing schemas for federal database environment architecture. Technical Report 9701, Frame Inform Systems, Riga, Latvia, 1997. (<ftp://ftp.cs.chalmers.se/pub/users/diskin/REPORTS/tr9701.ps>).
- [4] Z. Diskin. The arrow logic of meta-specifications: a formalized graph-based framework for structuring schema repositories. In B. Rumpe, H. Kilov and I. Simmonds, editors, *7th OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*, TUM-I9820, Technische Universitaet Muenchen, 1998.

- [5] Z. Diskin, B. Kadish, and F. Piessens. What vs. how of visual modeling: The arrow logic of graphic notations. In *Behavioral specifications in businesses and systems*. Kluwer, 1999.
- [6] P. Drew, R. King, D. McLeod, M. Rusinkiewicz, and A. Silberschatz. Report on the workshop on semantic heterogeneity and interoperation in multidatabase systems. *SIGMOD Record*, 22(3):47–56, 1993.
- [7] J.A. Goguen. A categorical manifesto. *Mathematical structures in computer science*, 1(1):49–67, 1991.
- [8] J.A. Goguen and R.M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of ACM*, 39(1):95–146, 1992.
- [9] N. Mart’Loliet and J. Meseguer. General logics and logical frameworks. In D. Gabbay, editor, *What is a logical system?*, pages 100–137. Oxford University Press, 1994.
- [10] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

A Appendix. Example of building metamodel of modeling system.

We begin with some necessary definitions. A *directed graph* is a system of nodes and arrows between them, there may be multiple arrows between the same pair of nodes. A *mapping* between graphs $f: G_1 \rightarrow G_2$ maps nodes and arrows of G_1 into, respectively, nodes and arrows of G_2 so that incidence between nodes and arrows is respected. That is, an arrow $a: N \rightarrow N'$ in G_1 is mapped into an arrow $f(a): f(N) \rightarrow f(N')$ in G_2 .

A.1 Modeling system. Suppose that our specifications are directed graphs whose nodes may be labeled by non-negative integers (further we will call them just integers). Their instances are supposed to be systems of sets (built over some predefined universe of objects) and functions between them: sets are assigned to nodes and functions to arrows so that incidence between nodes and arrows is respected. That is, if we have an arrow $a: N \rightarrow N'$ in a specification graph G , its instance is a function $\llbracket a \rrbracket: \llbracket N \rrbracket \rightarrow \llbracket N' \rrbracket$ between sets $\llbracket N \rrbracket$ and $\llbracket N' \rrbracket$. In addition, if a node N is labeled by an integer k , the set $\llbracket N \rrbracket$ must have exactly k elements in any any instance $\llbracket \cdot \rrbracket$ of G . This gives a modeling system

$$GMod = (Lab_{int}Graph, SetFun)$$

where $Lab_{int}Graph$ denotes the set of all directed graphs labeled by integers and $SetFun$ is their semantics as described above. \diamond

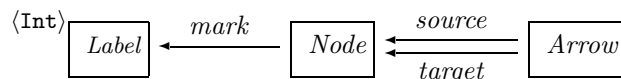
Now we turn to building (meta)specification (metamodel) of the modeling system $GMod$.

A.2 Specifying notation. Consider the following modeling system

$$M_1 = (Lab_{set}Graph, SetFun).$$

Its schemas are directed graphs whose nodes may be labeled by (names of) sets, for example, **Empty** or **Int**. Instances of such set-labeled graphs are systems of sets and functions like for $GMod$ and, in addition, if a node N is labeled by a set X then its only possible instance is the set X , $\llbracket N \rrbracket = X$.

Now, the following graph Gra_1 can serve as a meta-specification of the $GMod$ -notation:



where **Int** names the set of all (non-negative) integers enriched with an additional element $\{*\}$ and $\langle \mathbf{Int} \rangle$ is the corresponding label hung on the node *Label*.

Indeed, any instance of this graph consists of three sets, $\llbracket Node \rrbracket$, $\llbracket Arrow \rrbracket$ and $\llbracket Label \rrbracket$, and three functions between them, $\llbracket source \rrbracket, \llbracket target \rrbracket: \llbracket Arrow \rrbracket \rightarrow \llbracket Node \rrbracket$ and $\llbracket mark \rrbracket: \llbracket Node \rrbracket \rightarrow \llbracket Label \rrbracket$. In addition, because of the label $\langle \mathbf{Int} \rangle$, the set $\llbracket Label \rrbracket$ has to be the set \mathbf{Int} in all instances of \mathbf{Gra}_1 and we can interpret the value $*$ of the function $\llbracket mark \rrbracket$ as "undefined", that is, no number is assigned. It is easy to see that any such an instance can be represented by a directed graph with nodes labeled by integers and, conversely, any directed graph so labeled is an instance of the schema \mathbf{Gra}_1 , that is, $Lab_{int} Graph = SetFun(\mathbf{Gra}_1)$. \diamond

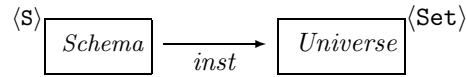
A.3 Specifying semantics. Consider the following modeling system

$$M_2 = (Lab_{gra} Graph, GraMap).$$

Its schemas are again directed graphs but now their nodes are optionally labeled by (names of) \mathbf{Int} -labeled graphs, for example, by " $\bullet \longrightarrow \bullet^3$ " or " \mathbf{Set} ". The latter is the name of the graph of all sets and functions built over the universe used for building $GMod$ -instances. In addition, each node in \mathbf{Set} , that is, some set over the universe, is labeled by its cardinality.

Instances of such graph-labeled graphs are systems of \mathbf{Int} -labeled graph and their mappings (the latter are graph mappings $f: G_1 \rightarrow G_2$ compatible with \mathbf{Int} -labels: if a node N in an \mathbf{Int} -graph G_1 is labeled by k then $f(N)$ in G_2 has to be also labeled by k). In addition, if a node \mathbf{N} in a graph-labeled graph $\mathbf{G} \in Lab_{gra} Graph$ is labeled by a graph X then $\llbracket \mathbf{N} \rrbracket = X$ in any instance $\llbracket \cdot \rrbracket$ of \mathbf{G} .

Now, the following graph $\mathbf{Gra}_2[S]$, parameterized by $GMod$ -schemas $S \in Schema$, can serve as a meta-specification of the $GMod$ -semantics:



where $\langle S \rangle$ is a label-parameter hung on the node $Schema$. Indeed, any instance of this graph consists of two \mathbf{Int} -labeled graphs, $\llbracket Schema \rrbracket$ and $\llbracket Universe \rrbracket$ and a mapping between them $\llbracket inst \rrbracket: \llbracket Schema \rrbracket \rightarrow \llbracket Universe \rrbracket$. Note however that for each $GMod$ -schema $S \in Lab_{int} Graph$, the label $\langle S \rangle$ bounds all instances of node $Schema$ to be the \mathbf{Int} -graph S while the set $\llbracket Universe \rrbracket$ is bounded to be our graph-universe \mathbf{Set} . Then, by the definition of \mathbf{Int} -graph mapping, the mapping $\llbracket inst \rrbracket$ is nothing but an assignment of sets to S -nodes and functions to S -arrows so that incidence is preserved and labels are respected, that is, $\llbracket inst \rrbracket$ is nothing but a $GMod$ -instance of S . Conversely, any $GMod$ -instance of S can be presented as such a mapping and, thus, $GraMap(\mathbf{Gra}_2[S]) = SetFun(S)$.

By definition, this means that \mathbf{Gra}_2 is a (meta)specification of $GMod$ -semantics and the quadruple $\mathcal{M} = (M_1, M_2, \mathbf{Gra}_1, \mathbf{Gra}_2)$ is a metamodel of modeling system $GMod$. \diamond

Our considerations above are not absolutely correct because of the possibility of logical circle when we specify graphs and their set-function instances by graphs and set-function instances. However, they could be made perfect by careful separating and restricting universes from which sets and functions involved are to be taken. Such a type-theoretical arrangement is standard (and even not too difficult) but we omit it because of it would make our description too bulky and overly rigor in comparison with the rest of the paper. So, building a rigor metamodel of our sample modeling system is not a big problem. In contrast, building a rigor model of metamodel is much more complicated and is quite another story. The point is that in a meta2model we should specify precisely what labels like \mathbf{Int} or \mathbf{Set} mean, that is, in fact, we need formal arithmetic and formal set theory in one or another form. The issue is a must for formalized mathematics but business/software modeling hardly need formal meta2modeling and informal yet precise description of metamodels like that displayed above is quite sufficient. So, among the three layers of modeling mentioned in OMGdoc, the subject really needs only two – those of modeling and metamodeling.

B Appendix. Modeling systems vs. institutions

The notion of modeling system is designed in the spirit of the so called institution theory already mentioned above. *Institutions* were invented by Goguen and Burstall (see [8] for a survey) for abstract (meta)reasoning about different logical calculi that have been studied in mathematics and theoretical computer science (TCS). Roughly, an institution is an abstract construct whose concrete instances are logical systems like equational logic or first-order logic including both syntax and semantics. Immediately after invention, the framework gained some popularity in TCS and, in particular, institutions were heavily used in the vast body of works on (algebraic treatment of) abstract data types, and for explaining semantics of logic programming [9]. However, it seems that now institutions is not very fashionable academic field.

As for more practical applications, one might expect that institutions should find a wide use in software engineering because of its real needs in managing different specification systems (logics) coexisting and interacting within a single heterogeneous space. However, up to now the institution framework was not applied here (except maybe some marginal attempts) and probably the reasons are the following.

Concrete examples of logical systems the inventors of institutions had in mind were logics like equational logic over some signature of operations symbols, or first order logic over some signature of predicate and operation symbols, or implicational logic of partially defined operations, and so on. In all these logical systems, a specification is a pair $S = (\Sigma, Th)$ with Σ a signature and Th a set of legitimate (in the logic in question) sentences (statements, constraints) in the signature Σ ; such a specification (or just its second component, if the first one is clear from the context) is usually called a theory. An instance of S is then an instance of Σ that satisfies all the statements (constraints) in Th ⁵:

$$Inst(S) \stackrel{\text{def}}{=} \{I \in Inst(\Sigma) : I \models \phi \text{ for any } \phi \in Th\}.$$

Thus, given an institution \mathcal{I} , one can build a specification (modeling) systems $mod(\mathcal{I})$ whose specifications (models in our sense) are theories and instances (models in the institution terminology) are instances of theories. However, still signatures are basic ingredients of these systems and so their metatheory is necessarily centered around the satisfaction relation \models between Σ -instances and Σ -sentences.

A peculiarity of specification systems used in software engineering (ER-diagrams, the UML-class diagrams etc) is that here one deals with integral specifications where it is not natural, and hardly useful, to extract pure signatures as such. Consider, for example, some object class diagram with an association R between classes A, B . It is clear that multiplicity constraints on the R 's ends are pure constraints (sentences) having nothing to do with the signature but how to classify R itself? In the standard first order logic approach, the signature of this fragment is formed by A and B as basic sorts while R is a relation of arity $A \times B$. However, it may well happen that one needs to consider R as a class itself and then R should be also treated as basic sort (with two projection functions $p: R \rightarrow A$ and $q: R \rightarrow B$ to be also entered into the signature). This latter method of extracting signatures is universal and then any specification is convertible into a theory in some signature of unary operations. However, in this way all the advantages of graphic specification languages used in software engineering disappear (compare, for example, directed graphs and their bulky and difficult to comprehend analytical presentations). In addition, the situation becomes even more complicated because of semantic relativism when within the same universe of discourse one often needs to deal with the two views on R : as a pure relation of arity $A \times B$ and as a class on its own.

A quite natural yet equally mathematically correct way of treating the issue is to consider graphic specifications in the categorical logic framework as graph-based logic theories or sketches ([5]). Our modeling systems is nothing but a variant of the institution approach suitable for categorical, that is, graph-based logics. By the way, a fundamental disadvantage of the OCL language as a means

⁵The reader should be warned that in mathematical logic and in the institution theory instances are normally called *models* so that model is a semantic construct – just the opposite to using this term in software engineering where a model means a specification

of specifying semantics of the UML is that OCL is a string-based language similar to first-order logic while the UML-schemas are graph-based; this results in a principle gap between the semantic intuition underlying the UML diagrams and their OCL-based formal semantics.

Another feature of the standard institution framework that prevents its adoption in software engineering is that institutions do not offer any machinery for derived information – a fundamental aspect of reasonable specification languages in software engineering (e.g., in data modeling it is the issue of query languages, see also 2.7).

The last remark. Actually, signatures can be recovered in modeling systems in the following way. Given a modeling system $Mod = (Schema, Inst)$, consider in the set $Schema$ maximal elements w.r.t. the partial order \prec defined in 2.3. The \prec -maximality of some $S_0 \in Schema$ means that it is the most general specification in the following fragment of $Schema$:

$$(S_0] \stackrel{\text{def}}{=} \{S \in Schema : S \prec S_0\}.$$

So, one can only reduce the number of S_0 instances by adding constraints and thus, we may call schema S_0 a *signature* and any schema $S \prec S_0$ is then a *theory* in the signature S_0 .