

What do models mean?

Ed Seidewitz / IntelliData Technologies Corporation
eseidewitz@intellidata.com

07 March 2003

Over the years since the adoption of the first UML and MOF specifications, there has been a great deal of discussion within the OMG community on modeling and metamodeling and the relationship of modeling and metamodeling languages. The circular nature of such relationships makes them particularly hard to discuss clearly, often hiding many important but subtle issues at the very foundation of what we do when we model. Even so, with the proper alignment of UML and MOF one of the crucial goals of the UML 2.0 and MOF 2.0 efforts, and with the centrality of modeling in OMG's emerging Model-Driven Architecture, this seems to be the right time for careful consideration of these issues.

In order to address this topic as clearly as possible, I asked myself a series of questions that, I think, need to be answered together to understand the fundamental question of "What do models mean?" This paper gives my answers to my own questions. Note that this is not intended as an evaluation of any of the UML 2.0 or MOF 2.0 submissions, as such. But it is necessary background for evaluating whether these submissions meet the foundational goals set out for the efforts.

The reader may also quibble with some of my terminology—I have chosen what seem to be the best terms to me, but there are, no doubt, better choices for some. I do think that coming to consensus on terminology can be important to promote better discussion, but I am more concerned about reaching some consensus on the underlying concepts.

What is a model?

A *model* is a set of statements about some *system under study* (SUS). The term *statement* is used here to mean some expression about the SUS that may be considered true or false (though no truth value has to necessarily be assigned at any particular point in time).

A model may be used to *describe* a SUS. In this case, the model is considered *correct* if all statements made in the model are true for the SUS.

Alternatively, a model may be used as a *specification* for an SUS, or for a class of SUS. In this case, a specific SUS is considered *valid* relative to this specification if no statement in the model is false for the SUS.

Examples

- A Newtonian model of a set of particles may make statements on the positions, velocities and masses of the particles at a given point in time. An alternative model could make statements on the trajectory of each particle over some period of time. In physics, such models are generally descriptive (e.g., the "particles" in the model may be planets), while in engineering such models are often specifications (e.g., the "particles" in the model may be spacecraft).
- A UML model of an object-oriented software system may make statements that the software system contains certain classes related in certain ways. Additional models could make statements on how instances of those classes are expected to interact, resulting in state changes over some period of time. Such models could either be describing the structure and operation of an existing system or they may be specifying how a planned system is to be built.

How is a model interpreted?

An *interpretation* of a model is a mapping of elements of the model to elements of the SUS such that the truth-value of statements in the model can be determined from the SUS, to some level of accuracy. Colloquially, an interpretation of a model can be said to give it "meaning" relative to the SUS. If this mapping can be inverted, so that elements of the SUS can be mapped to model elements, then a model can also be constructed as a *representation* of a given SUS, such that the all the statements in the representation are true for the SUS under the interpretation mapping.

Examples

- The interpretation of a position three-vector in Newtonian physics is that some element of the universe identified as a particle will be found at the coordinates given by the three numbers of the vector, relative to some origin point.
- The interpretation of a UML class model of an object-oriented software system is that there are elements of the SUS that may be identified as *classes* and that they have relationships that may be identified with the relationships given in the model.

How is a model used?

A *theory* is a way to deduce new statements about a SUS from the statements already in some model of the SUS. This can be considered as a way to extend the original model or as a way to determine whether the model *conforms* to the theory. In the latter case, any statements deduced from some subset of the model must be *consistent* with any other statements in the model. Two statements within one model of an SUS are *consistent* if they can both be true for the SUS (that is, the truth of one statement does not necessarily imply the falsity of the other).

Given a model that conforms to a theory, one can use an interpretation of the model to make deductions about a SUS by making deductions within the model using the theory. If one is modeling descriptively, then a theory is considered *correct* if deductions made about the SUS using the theory correspond to what is actually observed of the SUS (to some level of accuracy). On the other hand, for a specification model, the theory is assumed to be correct, so that all statements deducible from the specification also effectively become part of the specification—that is, not only can no statement in the specification model itself be false for a *valid* SUS, but, also, no statements deducible from the specification can be false.

Examples

- Given the positions, velocities and masses of two particles at a particular point in time, Newton's theory of gravity can be used to deduce the trajectories of the particles over time under the influence of their mutual gravitational attraction. Alternatively, given a model of the trajectories of the two particles, one can determine whether the trajectories are consistent with the Newtonian theory of gravity. These models make predictions about how actual particles will move. Newtonian theory is *correct* up to the level of accuracy at which relativistic effects become significant.
- Given a UML class model of an object-oriented system and an initial state of the system, a “theory of UML class modeling” would allow one to make deductions about how instances of those classes will interact and how their states will evolve under a given set of external stimuli. Given, in addition, a UML interaction model for the system, such a theory should allow one to deduce whether this model is consistent with the class model. These models can be used to make predictions about the operation of the software system (objects as instances of classes) based on the models. For descriptive models, UML semantics are *correct* to the extent that, e.g., hardware limitations and failures can be ignored. For specification models, the software system is *valid* if it does not violate any of the predictions.

How is a model expressed?

A *modeling language* is a language used to express the statements in models of some class of SUS.

Examples

- The mathematics of vector calculus is the primary modeling language for Newtonian physics.
- UML is a modeling language for “information-processing” systems. The term *information processing* is used here to encompass any SUS for which it is useful to make statements about the data maintained and/or behavior exhibited by the system relative to its environment.¹

¹ The applicability of UML was originally object-oriented software systems, then all software or “software-intensive” systems and now is often taken to include non-software systems such as businesses and hardware, when considered from the general point of view of “information processing”. Note also that defining the “class of

Digression: Model theory

The concept of *interpretation* as the means for assigning truth-values to linguistic statements is taken from the discipline of mathematical logic called *model theory*. However, the way I have defined the terms *model* and *theory* above are different than the way these same terms are used in model theory.

In model theory, a “theory” is a set of statements in a given language. An interpretation of the theory provides the basis for determining the truth-values of these statements relative to some “model” (or, more generally, a “universe of discourse”).

As you can see, the model theoretic definition of “theory” is essentially the same as the definition I give for *model*. The model theoretic concept of “model” corresponds to a kind of *system under study* as I defined it above.

I chose to use the term *model* because this is consistent with how the term is usually used in software modeling and in other scientific and engineering disciplines. However, there is an intentional correspondence between the concept of *model* as defined here and the concept of a “theory” in model theory. Despite the possibility of further confusion with the terminology of model theory, I chose to use the term *theory* to mean a set of deduction rules over models (as I define it), since this is consistent with how the term is used in other scientific and engineering disciplines.

If the reader is familiar with model theory, then it may be useful to keep the above correspondences in mind, though I will use the terms *model* and *theory* in the following exclusively as I have defined them earlier. If the reader is not familiar with model theory, then it is probably best to just ignore this digression.

What is a metamodel?

A *metamodel* is a specification model for which the SUS being specified are models in a certain modeling language². That is, a metamodel makes statements about what can be expressed in the valid models of a certain modeling language.

Examples

- Mathematics provides the metamodel for vector calculus, and hence for Newtonian physics, usually as expressed in various mathematical textbooks.
- The *UML Specification* document provides a metamodel of UML. That is, it includes a set of statements about UML models that must not be violated by any valid UML model. (Note that, in its entirety, this metamodel includes all of the concrete graphical notation, abstract syntax and semantics for UML.)

How is a metamodel interpreted?

Since a metamodel is a model of a modeling language, an *interpretation of a metamodel* is a mapping of elements of the metamodel to elements of the modeling language, such that the truth-value of statements in the metamodel can be determined for any model expressed in the modeling language. Since a metamodel is a *specification*, a model in the modeling language is *valid* only if none of these statements are false.

If the interpretation mapping for a metamodel is invertible, one can also uniquely map elements of the modeling language back to elements of the metamodeling language. In this case, given any model, we can invert the interpretation mapping to create a *metamodel representation* of the model—that is, a set of true statements about the model in the metamodeling language.

Examples

- The mapping of general mathematical concepts to the elements of vector calculus is provided by the definitions given in the textbooks. (Interestingly enough, the more formal these definitions, the less “meaning” they actually impart—indeed, the term *formal* indicates the importance of “form” over “meaning”.)

SUS” for which UML is intended to be applicable is actually somewhat difficult today—which is something of a problem in itself. The above is my best attempt for now.

² The term *metamodel* may be used more broadly than this. However, this more restrictive definition is sufficient for the purposes of this paper.

- The metamodel for the UML abstract syntax is interpreted by mapping instances of metaclasses (where both the metaclasses and their instances are expressed in the metamodeling language) to the syntactic elements of the UML graphical notation (that is, elements of the modeling language). This is an invertible mapping, so that we can also map syntactic elements of the UML graphical notation back to their representations as instances of metaclass instances (which is, indeed, how the mapping is given in the *UML Specification*.)

How is a metamodel used?

A *theory of a metamodel* is a way to deduce new statements about a modeling language from the statements already in a metamodel of the modeling language. Since a metamodel is a specification, a valid model in the modeling language must not violate any statement deducible using the theory from the explicit metamodel statements.

One way to look at this is to consider the statements of the metamodel as *postulates* about the modeling language. Then, given the *metamodel representation* of a model, we can determine, using the theory, whether the representation of the model is *consistent* with the metamodel (in the sense defined in the discussion of *theory* above). If it is consistent then the model is valid, otherwise it is not.

Examples

- Mathematical theories are rigorous systems for making deductions about mathematical statements. (In formal mathematics, the deduction system is actually more important than any “meaning” the statements may have.)
- The syntactic metamodel for UML places constraints on the allowable structure of and relationships between model elements represented as instances of metaclasses. This is the theory of the syntactic metamodel. If the instances in the metamodel representation of a UML model meet these constraints, the UML model is well formed, otherwise it is not.

How is a metamodel expressed?

A metamodel is a model, so it is expressed in some modeling language. Note that a single modeling language may have more than one metamodel, with each expressed in a different modeling language.

Of particular interest for our purposes is when the metamodel for a modeling language uses that same modeling language. That is, the statements in the metamodel are expressed in the same language as is being described by the metamodel. We will call this a *reflexive metamodel*.

A *minimal reflexive metamodel* is a reflexive metamodel that uses the minimum number of elements of the modeling language (for the purposes of that metamodel). That is, every statement about the modeling language that needs to be stated in the metamodel (for its purpose) can be expressed using this minimal set of modeling elements, but there are some statements that need to be stated that could not be expressed if any one of the modeling elements was removed from the set.

Examples

- As Gödel showed, statements of number theory can be used to make statements about number theory itself (e.g., “This proof is false”), given an appropriate encoding of number-theoretic statements as numbers, allowing number theory to express a reflexive metamodel for itself. Since all mathematical statements can, in principle, be reduced to set theory, and mathematical statements can themselves be expressed using mathematics, set theory (in principle) provides a minimal reflexive metamodel for all of mathematics.
- The object-modeling features of UML may be used to make statements about the syntax of UML itself, given an appropriate representation of the surface notation of UML as object structures. This representation is the *abstract syntax* for UML, essentially the abstract parse tree for any concrete surface notation. The metamodel elements of the abstract syntax are then classes and the interpretation mapping is to identify model elements with instances of those classes. The entire UML abstract syntax can be expressed using a minimal subset of the static structure modeling constructs of UML (e.g., for UML 1.x, this subset includes classes, packages, associations, generalizations and dependencies). Other metamodeling languages used in the *UML Specification* are OCL and English.

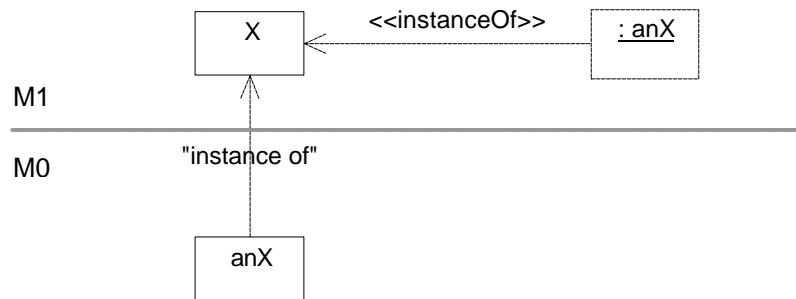
Digression: How does this relate to the four metalevels?

In terms of the OMG metamodeling levels, interpretation can be said to “cross metalevels”. For example, the interpretation mapping for UML maps from model elements, considered to be “at level M1”, to elements of the SUS, considered to be “at level M0”. Similarly, there are interpretation mappings from metamodel elements “at level M2” to model elements “at level M1” and from meta-metamodel elements “at level M3” to metamodel elements “at level M2”.

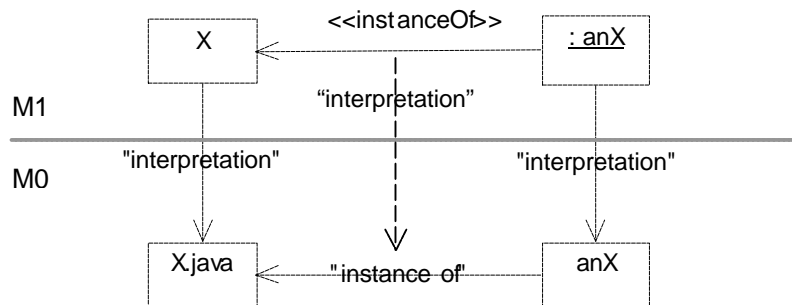
On the other hand, a theory is “within a single metalevel.” For example, a theory of UML allows some models to be deduced from other models (e.g., instance models from class models), entirely at level M1. Similarly, a theory of the UML abstract syntax allows the validity of a UML model to be determined entirely at level M2, after mapping the model to its metamodel representation.

Note that this view of the metalevels does *not* consider elements at one metalevel to necessarily be “instances of” elements at the level above it. This happens to be the interpretation mapping used between levels M3 to M2 and levels M2 to M1. But if this is also considered to be the relationship used between levels M1 and M0, it is difficult to understand how, for instance, component models at level M1 can be used to simply specify the structure of a software system, rather than the execution behavior.

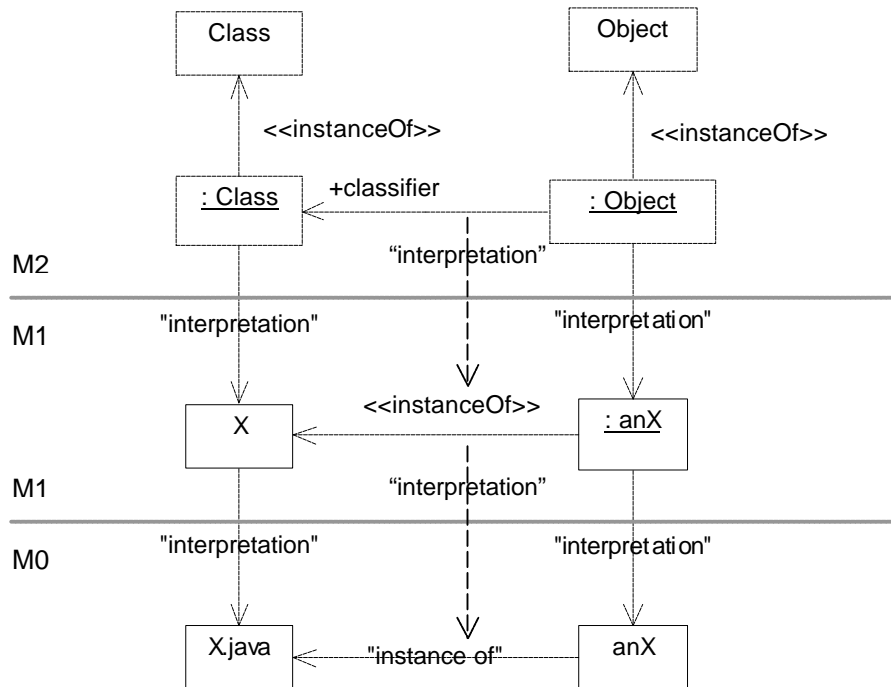
For example, consider the particularly simple case of the SUS at level M0 being an object-oriented software system. Typically, the relationship of the model of a class at level M1 to level M0 is considered to be something of the following sort.



The view taken here is that the concept of interpretation provides the general relationship between one metalevel and the next. Thus, the above situation would be considered as follows. (Note that I am using the concrete example of a Java class here at M0, but the argument applies equally well to other more abstract SUS domains, such as workers and the conceptual classes of their positions in a company.)



If we now add level M2 to this diagram, the interpretation mapping is between instances of metaclasses at M2 and the model elements at M1. This is shown below.



Now, it is common mental shorthand to identify a model element directly with its metamodel representation (e.g., the class X with its representation as an instance of the metaclass Class) and loosely refer to the model element as being directly "an instance of" the metaclass (e.g., class X "is an instance of" the metaclass Class). However, strictly speaking, the concept of "instance of" only has meaning within the theory of the metamodeling language. The fact that this concept is in the metamodeling language at all is merely consequence of the use in OMG of an object-oriented modeling language for metamodeling, which is not the only possible approach, and is not really fundamental to the relationship between the metalevels.

How is a reflexive metamodel interpreted?

Since a reflexive metamodel is expressed in the same modeling language as it is describing, its interpretation provides a mapping of the modeling language onto itself. Generally, this mapping will be from the entire modeling language to a subset of it. One can then iterate this mapping, each time producing a smaller subset, until one reaches the minimal reflexive metamodel that maps completely onto itself, rather than a subset.

An interpretation of a *minimal* reflexive metamodel maps the metamodel onto itself. This means that any statement in the minimal reflexive metamodel can be represented in terms of elements of the minimal reflexive metamodel. However, the interpretation of this representation is itself expressed reflexively as a mapping to yet another representation in terms of the minimal reflexive metamodel. This circularity means that, for a minimal reflexive metamodel, the interpretation mapping really provides no useful expression of the "meaning" of the metamodel itself.

The problem here is that, even though the interpretation mapping maps the set of modeling elements in a minimal reflexive metamodel into that same set, it does not map any given model into itself. Instead, on each iteration of the mapping, it simply generates increasingly complicated representations of the original model. Now, as noted above, the fundamental thing that is needed is a *theory* that allows us to make well-formedness determinations in terms of metamodel representations of models. In order to break the circularity, however, this theory cannot depend on any reflexive interpretation mapping of the metamodel.

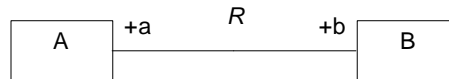
Extended example: Implications for UML

The general intent in UML 2 is, basically, for the MOF metamodel to be a subset of the UML metamodel and to provide the minimal reflexive metamodel for the abstract syntax of UML. Further, the entire UML abstract syntax should map into the MOF metamodel in one iteration. In this way, the MOF metamodel provides the *meta-*

metamodel ("level M3") for the UML abstract syntax, even though it is expressed in the same modeling language as UML.

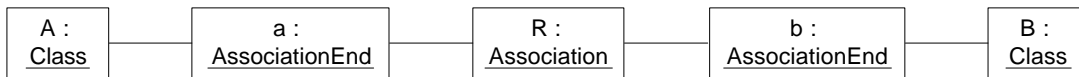
When using the UML language to express a model (at level M1), the *theory of UML* should provide the means for us to deduce whether, say, an instance model of an SUS is consistent with a class model for that SUS. But, as noted earlier, this is exactly the deduction we need to make in order to determine whether UML models are well formed, i.e., consistent with the abstract syntax. Thus, when using the MOF subset of UML (at level M3) to model UML itself (at level M2), the theory of UML also provides the basis for determining if a UML model is well formed in terms of the UML abstract syntax.

Consider the following simple UML class model.

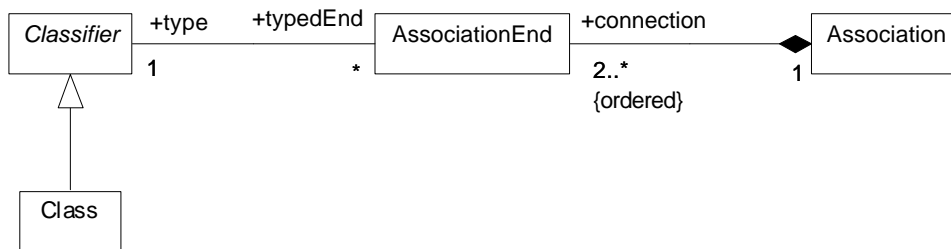


Is this a well-formed model? It is if one can show that the metamodel representation of the model, represented in the metamodeling language for UML, is consistent with the abstract syntax of UML, also represented in the metamodeling language.

In the absence of an accepted UML 2.0 specification, I will continue this example based on the abstract syntax metamodel from the current *UML 1.5 Specification*.³ The *Notation Guide* chapter of the *UML 1.5 Specification* defines how each of the graphical elements in the diagram above maps into metamodel instances (see, in particular, Section 3.22.6 for Class, Section 3.42.7 for Binary Association and Section 3.43.6 for Association End). Since the reflexive metamodeling language for UML is UML, this simple model may be represented as an instance model, as shown below.



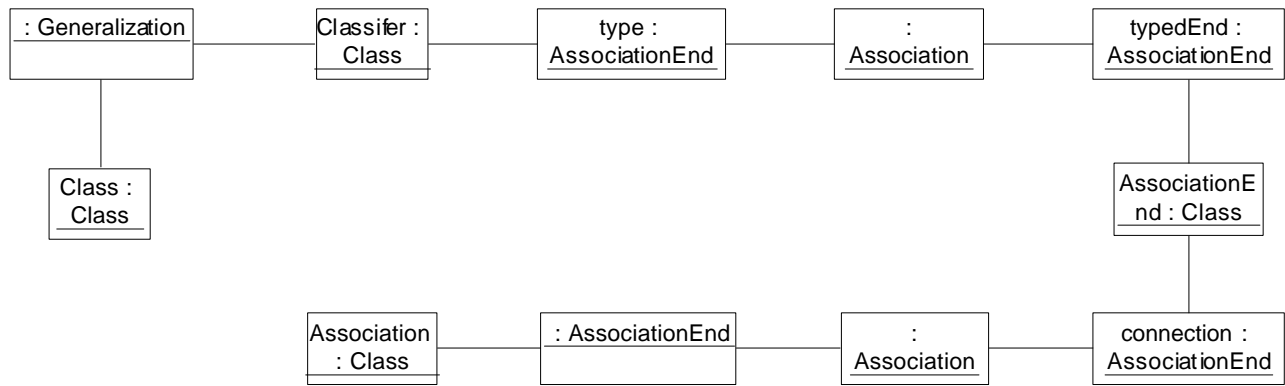
Now, the *Semantics* chapter of the *UML 1.5 Specification* includes the following class model defining the abstract syntax for associations (this is a fragment of Figure 2-3 in Section 2.5.2).



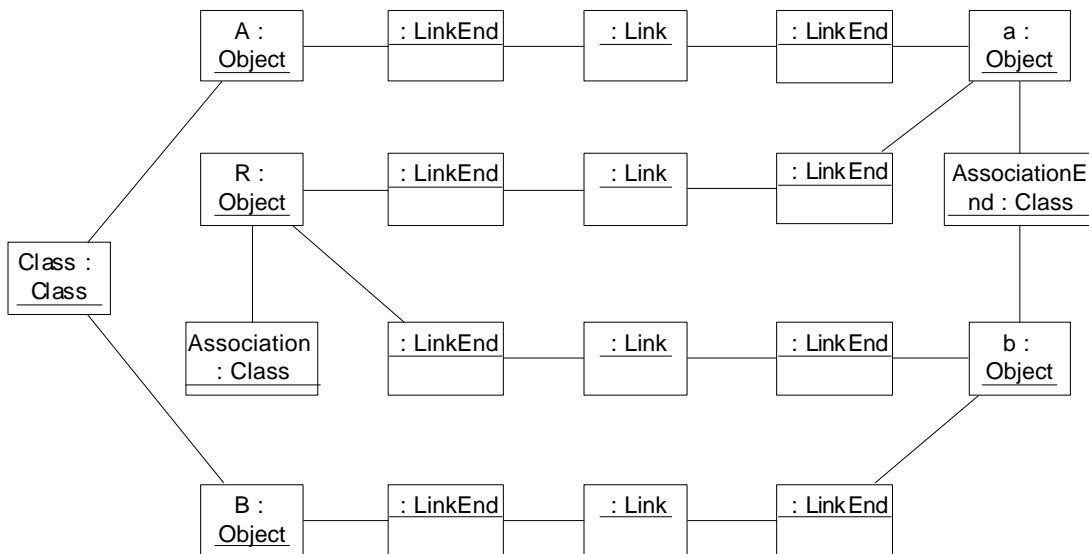
If it is possible (by the theory of UML class models) to deduce the instance model representing the UML model as a legal "instantiation" of this class model, then the UML model is well formed.

In order to make this well-formedness determination, I need to be sure that my metamodel representations are themselves well formed before applying the theory needed for the determination. However, since these representations are already within the minimal reflexive metamodel, their meaning is given by a mapping back to that metamodel. That is, the abstract syntax model is represented as the instance model below.

³ Even though the MOF 1.X metamodeling language is not actually a true subset of the UML 1.X modeling language, the discrepancies between the two languages are not important for this discussion (and are to be resolved in UML 2 and MOF 2). Further, the logical abstract syntax models for UML 1.X have always actually been presented in the UML 1.X modeling language (rather than being strictly in the MOF metamodeling language), and so are truly reflexive.



Even more awkwardly, the metamodel representation of the original model becomes the following. (There also should be an Association instance associated with each Link and an AssociationEnd instance associated with each LinkEnd, but these have been suppressed in order not to make the diagram completely intractable!)



But in order to determine if these models are well formed, I need to represent them again, and so on, indefinitely.

For UML, what we need is a theory that allows us to make deductions directly in the metamodeling language, which is to be a subset of the UML modeling language itself. And, clearly, the heart of this theory must be to provide an elaboration of the "instance of" relationship between classes and instances. Given such an elaboration, we can directly determine the conformance of the metamodel representation of a model as an instance model with the metamodel representation of the abstract syntax as a class model.

Conclusion: So, what do models mean?

As I have discussed, there are actually two aspects to the meaning of a model.

First is the relationship of the model to the thing being modeled. This is *meaning* in the sense of "This class model means that the Java program must contain these classes." I have called this an *interpretation* of the model.

Second is the relationship of a given model to other models derivable from it. This is *meaning* in the sense of "This class models means that instances of these classes are related in this way." I have called this a *theory* of the modeling language.

Both of these aspects of meaning are important for the very reasons we do modeling.

If we are modeling descriptively, then we will generally create a model representation from a given SUS. The interpretation of the resulting model is then exactly as a description of the SUS. We create such a model so that we can analyze the SUS by reasoning about the model. In order to reason about the model, we need a theory, and, in order to relate the results of this reasoning back to the SUS, we need to use the interpretation mapping again.

If we are modeling as a specification, then we will generally create one or more SUS intended to meet that specification. The interpretation of the specification determines the constraints on how a valid SUS may be constructed. We then want to be able to deduce from the specification observable properties of the SUS. This allows us to test that the SUS is correctly built by observing whether it has these properties. In order to make these deductions, we need a theory, and, in order to relate the deduced properties back to the as-built SUS, we need to use the interpretation mapping again.

Now, when we are metamodeling, the metamodel is a specification and the SUS is a modeling language. In the case of UML, we gain conceptual economy by using UML reflexively as the metamodeling language for its own abstract syntax. However, this reflexivity also tends to lead, in many discussions, to a conflation of the two aspects of meaning, which should strictly be understood to be distinct, even in reflexive use.

In particular, as pointed out earlier, the “instance of” relationship, which is formally defined only within the theory of the UML modeling language, tends to get identified with the interpretation mapping from metaclass instances to UML model elements. But simply providing the interpretation mapping (“meaning” in the first sense above) does *not* provide a formal definition of what “instance of” means in the metamodel (“meaning” in the second sense above). In order to provide an appropriate grounding for any deductions based on our metamodel, we must define the meaning of “instance of” within the theory of our metamodeling language, independently of the interpretation mapping we happen to use between the metamodel and modeling elements.

Suppose we take the MOF 2 metamodeling language to be the minimal reflexive metamodeling language for UML 2. It is for this language that one needs to provide the foundational theory of classes and instances, the grounding for all the rest of UML semantics. One would expect this theory to be effectively equivalent to some subset of the UML 2 superstructure semantics, though it could potentially be more than the core semantics of the UML 2 infrastructure.

In any case, without this well-grounded foundation, our models are, in the end, just pictures that don't really mean anything at all.

Acknowledgements

I would like to greatly thank Ed Barkmeyer, Steve Cook, Andy Evans, William Frank, Dave Frankel, Steve Mellor, Joaquin Miller, Jim Rumbaugh and Bran Selic for reviewing earlier versions of this work and participating in various lively discussions of the topics it addresses. The current version has benefited greatly from this very enjoyable interaction, though I would not say we have all yet come to a complete consensus. Thus, the responsibility for how their suggestions were interpreted (or misinterpreted) in this version remains, of course, mine.