What is a Model?

Thomas Kühne[1]

Darmstadt University of Technology, Darmstadt, Germany
kuehne@informatik.tu-darmstadt.de

Abstract. With the recent trend to model driven development a commonly agreed notion of "model" becomes a pivotal issue. However, currently there is little consensus about what exactly a model is and what it is not. Furthermore, basic terms such as "metamodel" are far from being understood in the same way by all members of the modeling community. This article attempts to start establishing a consensus about generally acceptable terminology. Its main contribution is the distinction between two fundamentally different kinds of models, i.e. "type model" versus "token model". The recognition of the fundamental difference in these two kinds of models is crucial to avoid misunderstandings and unnecessary disputes among members of the modeling community.

1 Introduction

Anytime a new research area gains momentum the task of defining its central notions needs to be addressed. Communities can take surprisingly long time spans to come to an agreement what notions like "object" and "component" should be encompassing. Although such efforts can be tedious and are known to endanger research meetings to stall on the "definition problem" already, they are necessary in order to enable communication among community members. Unless the community arrives at an agreement on its basic terms its communication will be plagued by misunderstandings both noticed and unnoticed. Without a shared conceptualization the different ontologies used by different members may potentially create the illusion of agreement where is none and raise barriers of communication where they are just accidental.

For the latest trend in software development the critical question is:

What is the "M" in "Model Driven Development"?

This question sets the context for our further discussion. We are not going to be concerned what the term "model" means for the fashion business, mathematicians[1], pedagogy, etc.

In software engineering "model" has traditionally been referring to an artifact formulated in a modeling language, such as UML, describing a system through the help of various diagram types. In general, such model descriptions are graph-based and typically rendered visually.

[1] In mathematics a model is an interpretation of a theory.

Dagstuhl Seminar Proceedings 04101
http://drops.dagstuhl.de/opus/volltexte/2005/23

In our context of interest, i.e., model driven development, such a characterization would be too narrow. Other artifacts, such as Java programs, are considered to be models as well.

The challenge is to apply the successful principle of unification[2] as long as it provides benefits—e.g., to understand code generation from a UML model as a "model to model" transformation—but stop before the term becomes meaningless. After all, practically anything could be characterized as an "object" but as a technical term it only provides value in communication if not everything is included by "object". Likewise one should attempt to define a boundary for "model" that is large enough to draw on the powers of unification but small enough to exclude interpretations that make it almost arbitrarily applicable.

The remainder of this article first attempts to home in on a characterization of "model" in the context of model driven development that everyone may subscribe to. Next we will distinguish two fundamentally different kinds of models. Only after the differences between these two kinds have been made explicit, will we be able to further define basic model properties and notions such as "metamodel".

2 What is a Model?

According to Stachowiak a model needs to possess three features [1]:

mapping feature A model is based on an original.
reduction feature A model only reflects a (relevant) selection of the original's properties.
pragmatic feature A model needs to usable in place of the original with respect to some purpose.

The first two features are covered simultaneously if one speaks of a model as a "projection" as this implies both something that is projected (the original) and that some information is lost during the projection. Of course exactly what information is dropped—by an activity called "abstraction"—and what is retained depends on the ultimate purpose the model is going to be used for.

The third feature can be further elaborated by detailing what the pragmatic use of the model is. According to Steinmüller (1993): A model is information

- on something (content, meaning)
- created by someone (sender)
- for somebody (receiver)
- for some purpose (usage context)

The above characterizations are in accordance with the following definitions[3] from Webster's new encyclopedic dictionary [2]:

a) a small but exact copy of something

[2] As proposed by Jean Bézivin in the Dagstuhl seminar 04101.
[3] Only definitions contributing some insight and which are relevant to our subject area have been chosen, excluding connotations such as "role model", etc.

An example for this type of model is a scale car model to examine e.g., aerodynamic properties. Here "exact" refers to the properties one wants to retain but must not be misunderstood to mean "complete".

2.1 A Copy is not a Model

If I build a car according to an original being precise in every minute detail, I have not constructed a model but a copy. If I use the copy in a crash test, I have not performed a model simulation but a real test run. Copies neither offer the advantages of models (typically cost reduction) nor their disadvantages (typically inaccuracy with regard to the original).

For an example of an inaccurate model consider an electrician who frequently needs to work out which of several light bulbs in the attic is controlled by a particular switch on story below[4]. The electrician is interested in walking the stairs as few times as possible and figures that bulbs and switches have "on" and "off" states and that all visits to the attic just need to generate a unique "on/off" sequence for each bulb so that it can be matched to the same "on/off" sequence of a switch. For three light bulbs one needs three different sequence which can only be generated with a minimum two visits. While this answer is true for the electrician's model, it is not for reality. In reality the electrician can figure out the correspondence of three bulbs and switches with just one visit to the attic. Hint: The model dropped the fact that light bulbs not only emit light but heat as well and that it takes a while for a light bulb to completely cool down again...

2.2 Motivation for Modeling

Coming back to the discussion of the dictionary entry for "model" above we may observe that in particular in model driven development we do not make use of any physical models. All our models are linguistic in nature; they are expressed in some language. Therefore we might be tempted to adopt the following general characterization:

> A model is a description of something.

However, this neglects the purpose of modeling. Imagine a complex distributed system for which an almost equally complex model exists. If we now create a very small simple model for simulating some of the performance properties of the system, we did not create the model to *describe* the original system. Although even the small and simple model describes some aspects of the original system the intent clearly is a different one.

The next two definitions from the dictionary—

b) a description or analogy used to help visualize something that cannot be directly observed

[4] This logic problem is taken from [3].

c) a pattern or figure of something to be made

—capture two other purposes of modeling: In addition to "simulation" we also have "description" and "construction plan".

In software engineering models created in the analysis phase are used to *describe* the problem whereas quite similar but more detailed versions of these models are used as *construction plans* in the design phase. This last purpose points out a fact made explicit by the last definition from the dictionary:

d) a theoretical projection of a possible or imaginary system

Ergo, the "original" does not have to exist. It might be something yet to be build or it may remain completely imaginary. Only the former possibility seems to be of relevance in our context. Further types of models of course do exists, e.g., "decision models" but they are not of further interest to us here.

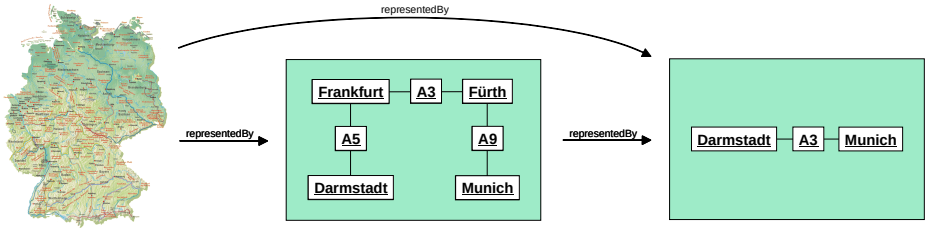

Fig. 1. Model Transitivity

2.3 Is Everything a Model?

This section 2 discussed a number of characteristics of models. While there is certainly value in employing unification to the greatest extent possible—as already alluded to in the introduction—it is equally certain that taking this too far will have detrimental effects.

Is there, for instance, a value in regarding tests or transformations as models? It might be useful to transform transformations and regard this as a "model to model" transformation as well. However, although models can be transformed not everything transformable needs to be a model. The author believes that the characteristic features of technical models as describe before should hold true for "model driven development"-models as well. If something lacks one of these features it should not be considered a model. For instance, what is the "original" modeled by a transformation? What kind of reduction (abstraction) would take place? If no good answers can be given to such questions one should abstain from regarding the subject in question as a model. First, this prevents diluting a term to a degree where it does not signify anything substantial anymore.

Second, it pays off to acknowledge different concepts where they help to give structure to a system using them. For instance, although it is quite possible and

elegant to unify the concepts of class, object, and method into one abstraction called "pattern"—as realized in the programming language BETA [4]—systems written in this style can be harder to understand. A pattern nested within a parent pattern might be a nested class, a local object, or a method. Which of these possibilities applies has to be figured out by reading the pattern definitions and their usages. Whether the unification advantages outweigh such a loss of apparent structure should always be carefully analyzed.

3 Kinds of Models

Intriguingly the discussion so far did not have to appreciate the existence of two fundamentally different kinds of models. If in personal communication one expert thought of the one kind and another expert thought of the other kind, so far they would have always been in agreement. However, as soon as issues such as "transitivity of the 'modelOf' relationship" or "what is a metamodel?" are touched upon the experts would start disagreeing and may only consolidate their views again when discovering their different mindsets.

Obviously there are many ways in which one can distinguish models, such as "product versus process models" or "static versus dynamic models" but for the issues at interest here these differences are irrelevant. The two kinds of models which are able to create a communication chasm between two people talking about basic notions such as "metamodel", if they are not aware of their different mindsets, are token and type models.

3.1 Token Models

A typical example for a token model is a map (see middle part of Fig. 1). Elements of a token model capture singular aspects of the original's elements. When using UML, one would use an object diagram to create a token model as the original's elements one is interested in are captured in a one-to-one mapping and are shown with their individual attributes.

In UML parlance such models are sometimes referred to as "snapshot models" since they capture a single configuration of a typically highly dynamic system. Other possible names are "representation model" (due to the direct representation character) or "instance model" (since the model elements are instances as opposed to types).

As shown in Fig. 1 with the rightmost model, the original for a token model may be a model (instead of an original from the non-linguistic world). Here, we might be talking about a map that uses a larger scale or just provides less information than the original map. Since the rightmost model of Fig. 1 is a model of a model we might be tempted to call it a "metamodel". After all, we use the prefix "meta" to indicate that some operation has been performed twice. For instance, when classifying twice, we prefer to refer to the result as a "metaclass" instead of "class class".

However, using the "meta" prefix in this way only makes sense if the relation established by the repeatedly applied operation is non-transitive. For instance, when generalizing twice we still call the result "superclass" instead of "super superclass" or let alone "metaclass". Any operation such as generalization which creates a transitive relationship among its elements is not suited to have its repeated application indicated with the "meta" prefix.

Now the "representedBy" relationship—induced by the activity of modeling—is transitive for token models. The courser map (derived from the finer map) is also a valid model of the original (the country). Hence, a token model of a token model is not its metamodel.

Token models have not been extensively used in model driven development yet but are useful for capturing system configurations or as the basis for simulations (e.g., regarding performance). Be that as it may, token models are often what people have in their minds when talking about models. The often used example of building plans for houses, so called blueprints in general, are token models. That is why it is important to be explicit about this fact and recognize the differences to the complementary model kind: The type model.

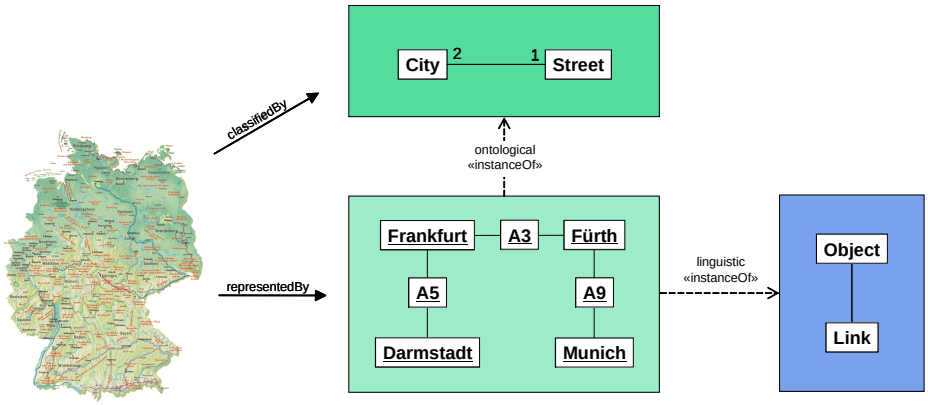

Fig. 2. Kinds of Models

3.2 Type Models

The human mind exploits the power of type models by using object properties (e.g., "four legged, fury, sharp teeth, and stereovision") to classify objects (e.g., as "predator") and drawing conclusions. This way the human mind does not need to memorize all particular observations and arrive at decisions afresh but just collects concepts and their universal properties [5].

Most models used in model driven development are type models. In contrast to the *singular* aspects captured by token models, type models capture the *universal* aspects of an original's elements. Fig. 2 shows (at the top) a type

model for the modeled country, using UML's natural diagram type for type models: The class diagram. Instead of showing all the particular elements and their relationships the type model shows the types of interest only. Thus "schema model" or "classification model" are further appropriate names.

Of course the type model of Fig. 2 may not only be used as a type model for the original but also for the token model produced from the original. Obviously, although the type model is a model of a model it is again inappropriate to call it a metamodel since both are models of the original. If we repeat classification twice, i.e., produce a type model of the type model in Fig. 2 containing for instance elements like "ConnectorType"—with instances "Street", "CarFerry", etc.—then we have a true metamodel. The (proposed) litmus test for metamodels—Are the instances of their models *not* instances of them?—holds true, as e.g., the motorway "A3" is a "Street" but *not* a "ConnectorType".

When dealing with a true metamodel, created from applying classification twice, the issue of "deep characterization" arises. One may wish to be able to control from the level of "ConnectorType" that instances of its instances will have certain properties, e.g., the feature "length". Only for such true metamodels, concepts to achieve "deep characterization" are required [6]. In the case of token models each individual model in a chain of models always directly characterizes the original because in this case the "modelOf" relationship is transitive.

Another way to make a model of the token model in Fig. 2, which is not a model of the original, is to model properties of the token model itself, instead of its content. The rightmost model in Fig. 2 models the language used to create the token model, in this case a tiny portion of the UML. This way, it is a model of a model but not a model of the original and thus may be called a metamodel. Indeed the UML language definition is typically referred to as the UML metamodel [7].

In order to make explicit that the rightmost model of Fig. 2 is a classification model, which constitutes a metamodel with respect to the language used in the token model, and the topmost model is a classification model in terms of the token model's content, Atkinson and Kühne proposed to distinguish between "linguistic" (for the former) and "ontological" (for the latter) classification [8].

In fact, just as the rightmost model defines the representation format for the token model, the topmost model defines the well-formedness of the token model. Checking the well-formedness is analog to checking whether the token model is a valid "sentence" of the language defined by the ontological type model. Hence, one could regard the two type models as "content language" and "representation language" definitions respectively.

Before we proceed to discuss the relationship between "metamodels" and "language definitions" we should point out that a model containing types still may be used as a token model for an original. Fig. 3 shows that although a UML class diagram can be regarded as a type model for the execution of a Java program, it is also a token model for the Java classes. The class diagram does not capture the universal aspects of the Java classes but directly represents them (the relevant ones) in a one-to-one mapping. Thus, when dealing with modeling

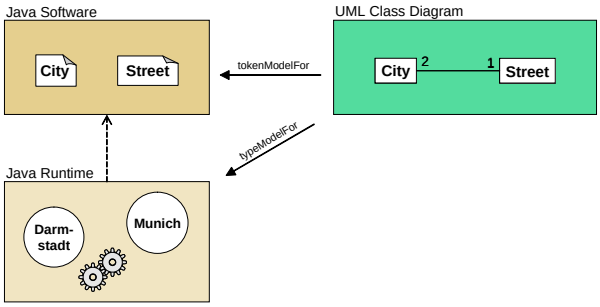

Fig. 3. Software Models

domains which support instantiation—here we regard a Java program execution as an instantiation of its program—in characterizing a model as being either a token or a type model, one must always specify with respect to which original.

4 Metamodel = Language Definition?

In the previous section we have seen that metamodels can be regarded (either linguistically or ontologically) as language definitions. Fig. 4[5] illustrates the relationships between a system, its models and the respective languages used to write the models. The author prefers to characterize the relationship between a

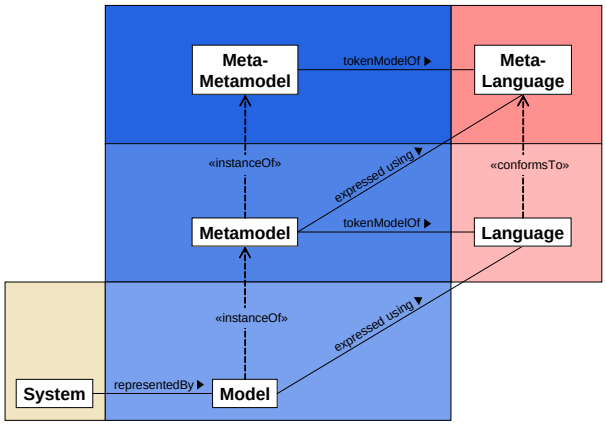

Fig. 4. Metamodels & Language Defintions

[5] Also compare to [9] (p. 24) using a different distribution of elements to levels and different terminology though.

model and its metamodel as "instanceOf" rather than "conformsTo", as for instance proposed by Jean Bèzivin, but such naming issues are rather secondary in comparison to the question whether it is appropriate to characterize a language definition as a (meta-)model.

After all, a language definition ought to be precise and complete so the reduction feature required of a true model seems to be missing. Do we just refer to the UML language definition as a "metamodel" because a modeling language was used to describe it?

One possible answer to justify the use of "metamodel" in this case is the fact that just the abstract syntax of the language is defined. Other aspects, such as concrete syntax and semantics are left out. But would we want to stop using the term "metamodel" if these aspects were included in the future? Probably not, so we might argue that there are aspects of the abstract syntax we do not want to specify and claim that as the reduction feature of our model.

However, we do not need to resort to such argumentation wich tries to establish that a metamodel truly is a model with a reduction feature with respect to the language it models as a *token model*. To justify the model nature of a metamodel it is sufficient to recognize that it universally captures all models that may be expressed with it, i.e, are instances of it. Hence, it is capacity as a *type model* for all the models expressible with it, that qualifies it as a model.

5 Conclusion

In order to establish a commonly agreed terminology it is inevitable for the model driven development community to define when the notions "model" and "metamodel" are applicable and when not. This article argued for maintaining the required features already well established for technical models and not extent "model" to be applicable to practically everything.

Defining the notion "metamodel" proved to be even more involved. To this end it was necessary to distinguish between token models and type models. Without this distinction an agreement about statements like "a model of a model is a metamodel" can not be settled systematically.

Only if one carefully keeps token models and type models apart are sensible statements possible. Discussing, e.g., deep characterization issues with a token model is completely beside the point just as the transitivity of the "modelOf" relationship never applies for type models.

Finally, it was argued that true metamodels can always be understood as language definitions either ontologically or linguistically and that it is justified to still regard them as true models with a reduction feature.

6 Acknowledgements

The author would like to thank the following participants of the Dagstuhl seminar 04101 on "Model-Driven Language Engineering"—(in alphabetical order)

Pieter van Gorp, Martin Grosse-Rhode, Reiko Heckel, and Tom Mens—who send emails to the author with their views on what a model is and is not.

References

[1] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, Wien and New York, 1973.

[2] Michael Harkavy and et al., editors. *Webster's new encyclopedic dictionary*. Black Dog & Leventhal publishers Inc., 151 West 19^{th} Street, New York 10011, 1994.

[3] Bernd Oestereich. *Objektorientierte Softwareentwicklung mit der Unified Modeling Language*. Oldenbourg, 1997.

[4] Ole L. Madsen, Kristen Nygaard, and Birger Möller-Pedersen. *Object-Oriented Programming in the BETA Programming Language*. Addison-Wesley and ACM Press, 1993.

[5] Jochen Ludewig. Models in software engineering—an introduction. *Journal on Software and Systems Modeling*, 2(1):5–14, March 2003.

[6] Colin Atkinson and Thomas Kühne. The essence of multilevel metamodeling. In Martin Gogolla and Cris Kobryn, editors, *Proceedings of the 4^{th} International Conference on the UML 2000, Toronto, Canada*, LNCS 2185, pages 19–33. Springer Verlag, October 2001.

[7] OMG. *Unified Modeling Language Specification, Version 1.4*, 2000. Version 1.4, OMG document ad00-11-01.

[8] Colin Atkinson and Thomas Kühne. Rearchitecting the UML infrastructure. *ACM Transactions on Modeling and Computer Simulation*, October 2003.

[9] Susanne Strahringer. *Metamodellierung Als Instrument Des Methodenvergleichs*. Shaker Verlag, Aachen, 1996.