Modeling Modeling

Pierre-Alain Muller[1], Frédéric Fondement[1], and Benoît Baudry[2]

[1] Université de Haute-Alsace, Mulhouse, France
{pierre-alain.muller,frederic.fondement}@uha.fr
[2] IRISA / INRIA Rennes, Rennes, France
benoit.baudry@irisa.fr

Abstract. Model-driven engineering and model-based approaches have permeated all branches of software engineering; to the point that it seems that we are using models, as Molière's Monsieur Jourdain was using prose, without knowing it. At the heart of modeling, there is a relation that we establish to represent something by something else. In this paper we review various definitions of models and relations between them. Then, we define a canonical set of relations that can be used to express various kinds of representation relations and we propose a graphical concrete syntax to represent these relations. Hence, this paper is a contribution towards a theory of modeling.

1 Introduction

Many articles have already been written about modeling, offering definitions at various levels of abstraction, introducing conceptual frameworks or pragmatic tools, describing languages or environments, discussing practices and processes. It is amazing to observe in many calls for papers how modeling is now permeating all fields of software engineering. It looks like a lot of people are using models, as Monsieur Jourdain [22] was using prose, without knowing it.

While much has already been written on this topic, there is however neither precise description about what we do when we model, nor rigorous description of the relations among modeling artifacts. Therefore we propose to focus on the very heart of modeling, straight on the relation that we establish to represent something by something else, when we say that we model. Interestingly, the nature of these (some)things does not have to be defined for thinking about the relations between them. We will show how we can focus on the nature of relations, or on the patterns of relations that we may discover between these things.

This paper is a contribution towards a theory of modeling. Whilst focused on modeling in software development and model-management, the presented material may apply to models in general, and in other disciplines. We define a canonical set of relations that can be used to ease and structure reasoning about modeling. This canonical set contains 5 representation relations that may be refined with nature (analytical/synthetical) and causality (correctness/validity).

A. Schürr and B. Selic (Eds.): MODELS 2009, LNCS 5795, pp. 2–16, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The paper proceeds as follows: after this introduction, section 2 (related works) summarizes what several authors have said about models, section 3 defines a set of primitive representation relations based on the analysis of these various points of views, section 4 illustrates the use of the notation via several examples excerpted from the software engineering field, and finally section 5 draws some final conclusions and outlines future works.

This paper is the result of numerous informal discussions we have had with so many people that it is almost impossible to enumerate them all here. We would like to especially thank a few of them: including Jean-Marie Favre, Thomas Kuehne, Colin Atkinson, Marc Pantel, and Christophe Gaston. We would also like to acknowledge the invaluable comments of anonymous reviewers of an earlier version of this paper.

Table 1. Summary of model definitions

Bézivin	"*A **model** is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the **actual system**.*" [2]
Brown	"***Models** provide **abstractions** of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on the relevant ones.*" [3]
Jackson	"*Here the word '**Model**' means **a part of the Machine's local storage** or database that it keeps in a more or less synchronised correspondence with **a part of the Problem Domain**. The Model can then act as a **surrogate** for the Problem Domain, providing information to the Machine that can not be conveniently obtained from the Problem Domain itself when it is needed.* " [4]
Kuehne	"*A **model** is an **abstraction** of a (real or language based) **system** allowing predictions or inferences to be made.*" [5]
Ludewig	"***Models** help in developing **artefacts** by providing **information about the consequences of building those artefacts** before they are actually made.*" [1]
OMG	"*A **model** of a system is a **description** or **specification** of that system and its environment for some certain purpose.*" [6]
Seidewitz	"*A **model** is a set of statements about some **system under study** (SUS).*" [7]
Selic	"*Engineering **models** aim to reduce risk by helping us better understand **both a complex problem and its potential solutions** before undertaking the expense and effort of a full implementation.*" [8]
Steinmüller	A model is information: on something (content, meaning), created by someone (sender), for somebody (receiver), for some purpose (usage context). [9]

2 Related Works

Much has already been written on modeling. In this section we will examine related works, and start to classify what authors have said about models. The following table contains a summary of model definitions, even if Jochen Ludewig states in [1] that "*nobody can just define what a model is, and expect that other people will accept this definition; endless discussions have proven that there is no consistent common understanding of models*".

Features of Models
According to Stachowiak [10] a model needs to posses the following three features:

- **Mapping feature.** A model is based on an original.
- **Reduction feature.** A model only reflects a (relevant) selection of an original's properties
- **Pragmatic feature.** A model needs to be usable in place of an original with respect to some purpose.

According to Bran Selic [8] an engineering model must posses the following five characteristics:

- **Abstraction.** A model is always a reduced rendering of the system that it represents.
- **Understandability.** A model must remain in a form that directly appeals to our intuition.
- **Accuracy.** A model must provide a true-to-life representation of the modeled system's features of interest.
- **Predictiveness.** A model must correctly predict the interesting but nonobvious properties of the modeled system.
- **Inexpensiveness.** A model must be significantly cheaper to construct and analyse than the modeled system.

Different Kinds of Models
Ed Seidewitz classifies models in two categories: descriptions and specifications. "*A model may be used **to describe** a SUS (System Under Study). 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 a 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.*" [7].

Jean-Marie Favre, reminds us that systems have the truth, not models: "*Making the distinction between **specification models** and **descriptive models** is useful to express who, of the model or the system, has the truth*" [11]. Jochen Ludewig further states that in order to make our models more useful we have to compare them with reality: "*The reality is always right and the model is always wrong*" [1]. This is also acknowledged by Michael Jackson: "*The model is not the reality*" [4]. Wolfgang Hesse,

stresses the fact that in software engineering models often play a double role: they may be either *prescriptive* or *descriptive*, depending on whether it is there *earlier* or *later* than its original [12]. He coins this the *Janus View.* This is close to the opinion of Bran Selic, in [8] where he states that the models may be developed as a precursor to implementing the physical system, or they may be derived from an existing system or a system in development as an aid to understanding its behavior.

Kuehne, going back to Peirce's (1839-1914) seminal work about semiotic, also distinguishes between token and type models [5]. He gives the following definitions:

- **Token models.** "*Elements of a token model capture **singular** (as opposed to universal) **aspects** of the original's elements, i.e., they model individual properties of the elements in the system.*"
- **Type models.** "*Most models used in model driven engineering are type models. In contrast to token models, type models capture the **universal** aspects of a system's elements by means of **classification**.*"

Another classification of models is provided by Mellor and his colleagues in [13] taking yet another perspective on models. The distinction is made between three kinds of models, depending on their level of precision. A model can be considered as a *Sketch*, as a *Blueprint*, or as an *Executable*. Fowler suggests in [14] a similar distinction based on three levels of models, namely *Conceptual Models*, *Specification Models* and *Implementation Models*.

Definition of Relations between Models

In [15] Bézivin identifies two fundamental relations coined *RepresentationOf* and *ConformantTo*. Jean-Marie Favre shows in [16] that the *ConformantTo* relation is actually a short-cut for a pattern of *RepresentationOf* and *ElementOf* relations. In Jean-Marie Favre's view (called mega-model), further expressed in [17], all MDE artifacts can be described with 4 (+1 derived) basic relations (*RepresentationOf*, *ElementOf*, *DecomposedIn*, *IsTransformedIn*, and the derived *ConformsTo*).

Ed Seidewitz also identifies two relations [7], named *interpretation* (the relationship of the model to the thing being modeled) and *theory of the modeling language* (the relationship of a given model to other models derivable from it).

3 Towards a Model of Modeling

In this section we will define a model of modeling along with a notation to represent relations between modeling artifacts. By a model of modeling (hence the title of this paper: *modeling modeling*) we designate a representation of what we manipulate when we use modeling techniques. Our target domain is software development; therefore, all our examples will be drawn from the software engineering field.

We will use a very simple language to build this representation, based on "things" and "arrows" between them, such as the "objects" and "morphisms" found in Category Theory [18]. Things can be anything (this includes what other authors have called models and systems), and nothing has to be known about the internal structure of these

things (which therefore do not have to be collections of "elements"). Conversely, arrows do not need to be functions between sets (thus arrows cannot be applied to "elements" but only composed with other arrows).

We do not want to come up with a brand new interpretation of what a model is. In our mind, the model of modeling that we are defining should reflect (or encompass) the various points of views which have already been expressed by the authors cited in the related works. To this end, we will first analyze these points of view, and next use our simple notation to synthesize them all into one single representation.

Let's start by modeling the fact that we have things which represent others things. As stated by Bran Selic [8], we first have to find a tradeoff between abstraction and understandability; therefore we will depart from the single *System class* view of Jean-Marie Favre [11], and distinguish between a *source* thing (that many authors call the model) and a *target* thing (called *original* by Stachowiak [10]), although we understand that being a source thing or a target thing is relative to a given arrow, and does not imply anything about a given thing. This is represented in Figure 1, where the source is named X, the target Y, and the *RepresentationOf* relation μ.

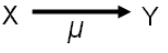


Fig. 1. X is a representation of Y

We are using on purpose a very simple graphic concrete syntax for representing modeling relations. Our notation is based on arrows, and is intended to be easy to draw by hand (on blackboard and napkins).

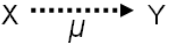
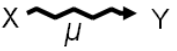
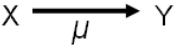
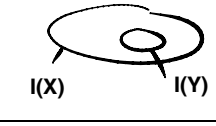
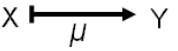
Intention

Neither things nor representations of things are built in isolation. As said by Steinmüller, both exist for a given purpose, exhibit properties, are built for some given stakeholders [9].

We can think about this as the *intention* of a thing. Intentional modeling [19] answers questions such as who and why, not what. The intention of a thing thus represents the reason why someone would be using that thing, in which context, and what are the expectations vs. that thing. It should be seen as a mixture of requirements, behavior, properties, and constraints, either satisfied or maintained by the thing.

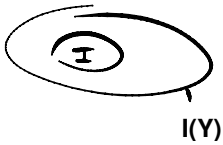
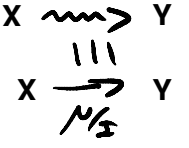
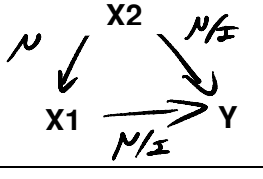
As already said earlier, the "category theory kind" of thinking that we take in this paper does not require a description of the internals of the modeling artifacts (nor their intentions). Hence, it is enough to say that artifacts have an intention. The intentional flavor of models has also been used by Kuehne [23] in his description of metamodeling and by Gasevic et al. in their extension of Favre's megamodel [24]. The consequences of intentional thinking applied to modeling can be understood and represented using Venn diagrams [20]. The following table summarizes how the μ-relation may be specialized:

Table 2. Variations of the μ-relation, and graphical notation

	Intention	Description	Notation
a)		X and Y have totally different intentions. This usually denotes a shift in viewpoints.	
b)		X and Y share some intention. X and Y can be partially represented by each other. The representation is both partial and extended.	
c)		X is a partial representation of Y. Everything which holds for X makes sense in the context of Y. Y can be partially represented by X.	
d)		X and Y share the same intention. They can represent each other. This usually denotes a shift in linguistic conformance.	
e)		X covers the intention of Y; X can represent Y, but X has additional properties. It is an extended representation.	

All authors agree to say that the power of models stems from the fact they can be used in place of what they model, at least for some given purposes. This is what Stachowiak [10] calls the *pragmatic* feature of models. In practice it is convenient to work with a subset of the intention, and to consider that the μ-relation is a complete representation of that given subset: hence the μ/I notation below, which means that X is a representation of Y (for a given subset of the intention). The I sign can then be used elsewhere in a diagram, to show that a given pattern holds for that subset of the intention. If intention is constant throughout the diagram, it can be omitted as a notation shortcut.

Table 3. Notation shortcut. X is a complete representation of Y, for a given subset of the intention (in a given context).

Analytical vs. Synthetical Nature of Representations

As seen earlier, several authors make a distinction between analytical models and synthetical models (respectively descriptive and specification models in the sense of Seidewitz [7] and Favre [11]).

An analytical representation relation states that the source expresses something about the target. We define the analytical representation (represented μ_α) as:

$$X \xrightarrow{\mu_\alpha} Y : \exists T_\alpha \mid x = T_\alpha(y)$$

where T_α is a relation such as X can be derived (or abstracted) from Y. In model-driven parlance T_α could denote a model-transformation. Interestingly, intentions of source and target do not necessarily have to overlap (notice that for convenience we use here a simple arrow as a placeholder for the different kinds of relations that we have defined in table 2). In terms of truth (as coined by Favre), truth is held by the target in case of μ_α representation.

A synthetical representation relation explains that the target is generated from the source. We define the synthetical representation (represented μ_γ) as:

$$X \xrightarrow{\mu_\gamma} Y : \exists T_\gamma \mid y = T_\gamma(x)$$

where T_γ is a relation such as Y can be derived (or generated) from X. In model-driven parlance T_γ could again denote a model-transformation. In terms of truth, truth is held by the source in case of μ_γ representation. If we talk in terms of intentions, this means that the intention of Y can be generated (synthesized) from the intention of X, or at least be driven by the intention of X, as Y is actually the result of T_γ applied to X. Quantifying the respective contributions of X and T_γ to the synthesis of Y is out of the scope of this paper.

However, if one wants to represent that the transformation significantly contributes to the target's intention, it is possible to use an explicit representation such as in Figure 2. Y is partially generated from X (for the S part of the intention). The complement (the S' part) is provided by T_γ. This could typically be used to represent that X is a PIM (Platform Independent Model), and Y a PSM (Platform Specific Model), with the specifics of the platform being introduced in Y by the T_γ transformation.

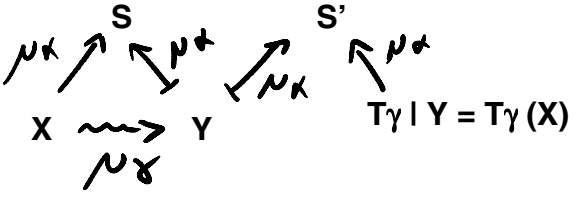


Fig. 2. Explicit representation of the contribution of the transformation used to generate Y from X

Causality

Causality addresses the synchronization concern raised by Michael Jackson [4]; it expresses both *when* the μ-relation is established, and *how* (if ever) it is maintained

over time. Causality is either continuous (the relation is always enforced) or discrete (the relation is enforced at some given points in time). Causality is also tightly coupled with the truth of Favre [11]; actually, causality is a concern about whether a representation is still meaningful when the truth has changed. Going back to the definition of *correctness* and *validity* given by Ed Seidewitz [7], causality states:

- for an analytical representation, when X is *correct* wrt. Y.
- for a synthetical representation, when Y is *valid* wrt. X.

For computer based systems, causality is typically discrete, and making the models meaningful requires adherence to results of information theory such as Nyquist-Shannon sampling theorem [21]. Causality can be used to re-visit the definition given by Wolfgang Hesse, who makes an amalgam between analytical/synthetical representation, and earlier/later existence, when he proposes to distinguish between descriptive and prescriptive "*depending on whether it is (the model) there earlier or later than its original*" [12]. A way to lift this ambiguity is to separate clearly between nature (analytical/synthetical) and causality (correctness/validity) of the representation relation. In Figure 3 the model is a causal analytical representation of the system. If the system changes, the causal μ_α relation implies that the model is updated. In turn, as the model is also a causal μ_γ representation of the program, the program is updated to remain an analytical representation of the system.

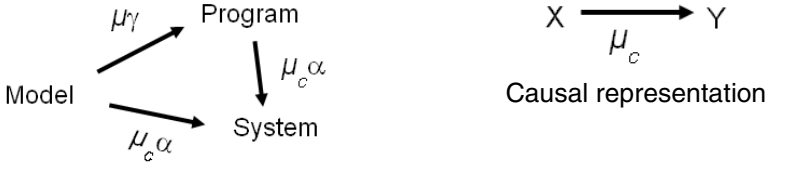


Fig. 3. Causality implies maintaining the representations over time

Transitivity

Transitivity addresses the composition properties of μ-relations of the same nature. Transitivity is realized when the intention of a composed μ-μ-relation contains the intention of a μ-relation. If transitivity holds, then it is possible to use the model of a model of a thing, in place of the model of that thing.

In some cases, there is only one possible result for the composition of relations. For example, on the third line of Table 4, if X is an extended representation of Y and if Y has the same intention as Z, then X is an extended representation of Z. In some cases there are 2 or 3 possible results when composing relations. For example, Figure 4 illustrates the two situations that can occur when X is an extended and partial representation of Y and Y is an extended and partial representation of Z. In case a, the intention that X shares with Y does not overlap at all with the intention that Y shares with Z, this means that X and Z have two completely different intentions. In case b, the intention that X shares with Y overlaps with the intention that Y shares with Z, this means that X is an extended and partial representation of Z.

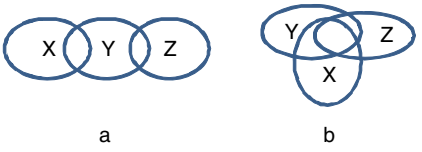
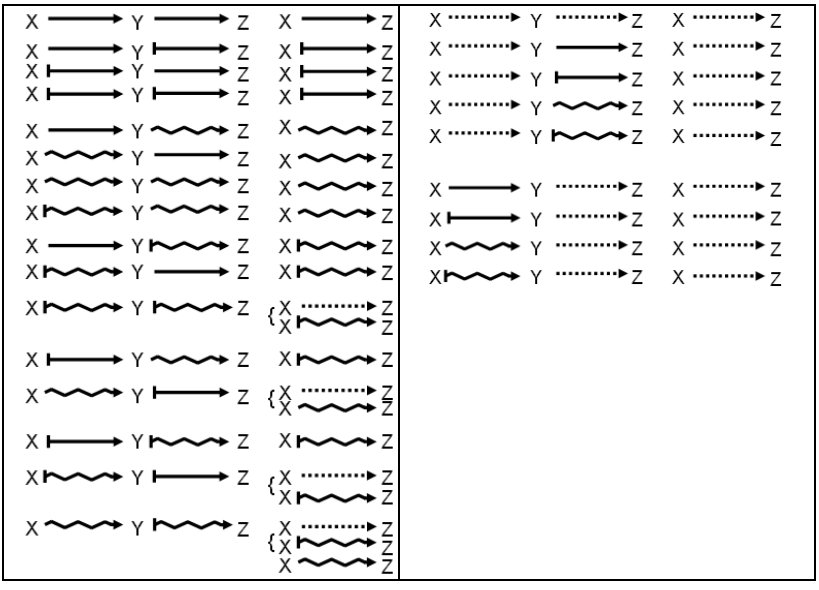


Fig. 4. Intention overlapping when composing partial extended relations

Table 4. Composition law for representations



4 Examples

4.1 This Is Not a Pipe

Let's examine the already classic example inspired from Magritte's painting. The picture is a μ_α representation of the pipe. The picture and the pipe share some intention. In addition, the real pipe could be used to smoke, while the picture could be used to show the pipe remotely. This is represented by an extended partial μ_α representation.

In the following example, the distribution of colors plays the role of an analytical model, providing information about the picture from which it is generated. It does not share intention either with the picture or with the pipe (this is modeled by the dashed arrow); however it may be used to have an idea of the color of the real world pipe (transitively following the μ_α relations).

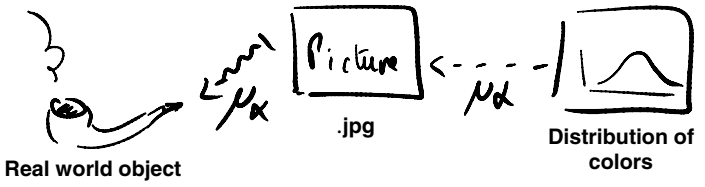


Fig. 5. Example of μ_α relations

4.2 Jackson'sProblem Domain and Machine

In table 2, the c) case represents the fact that the target (in our case generated) thing contains the intention of the source thing. This is especially interesting in case the source was itself in a μ_α relation with a third thing. Figure 6 shows such situation. M stands for model, S for system, and R for representation (with the idea that R is a computerized representation, in other words a program which implements S).

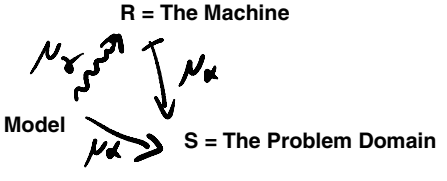


Fig. 6. Generated machine implementing a μ_α representation

This is the typical case for modeling, such as described for instance by Michael Jackson. S is the problem domain. R is what Jackson calls the machine. The μ_α relation from R to S is what Jackson calls the "*'model' which is part of the local storage or database that it keeps in a more or less synchronized correspondence with a part of the problem domain*" [4]. This view is also in line with Bran Selic, who states: "*the model eventually becomes the system that it was modeling*" [8].

The partial μ_γ and the extended μ_α relations express the fact that R is "richer" than M (and thus S) in terms of intention, because R contains additional information required for execution. The intention of the model can also be seen as the intersection of the intensions of the machine and the problem domain. The grayed part represents the additional intension required to "implement" the intention of the problem domain. This is what we name *platform dependence* in Figure 7.

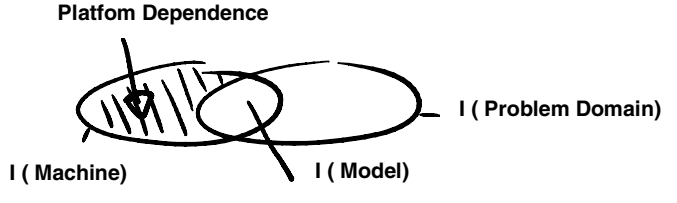


Fig. 7. The machine implements the subset of intention of the problem domain, represented by the model

4.3 PIM, PSM and PDM

A PSM (Platform Specific Model) is a refinement of a PIM (Platform Independent Model), which contains additional platform information as given by a PDM (Platform Description Model). The Venn diagram in Figure 8 shows how all successive levels of refinement extend the intention of the System, with platform dependent information required for implementation.

We also see here how the previous example (the triad System-Model-Representation) may be used as a meta-modeling pattern, by replacing M (the model) by PIM and R (the representation) by PSM (PDM was left unexpressed in the pattern).

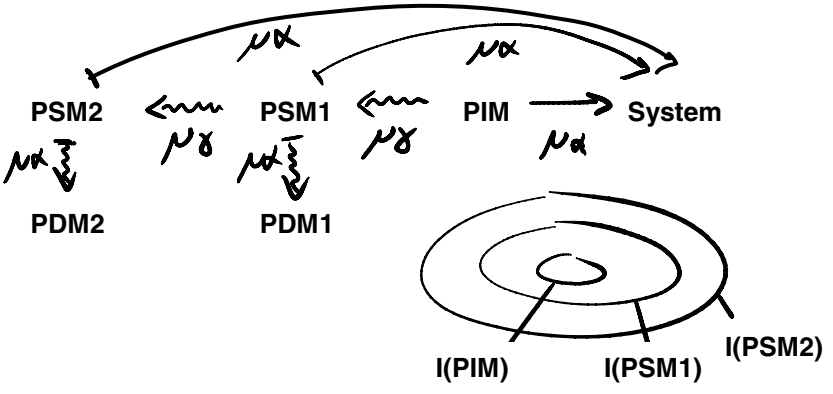


Fig. 8. Refinement of PIM into PSM, with platform specific information

4.4 Host-Target Development

In host-target development, the same program (here the model) is compiled both for a host machine (typically a workstation) and a target machine (typically some embedded computer). This allows early problem detection, even before the final hardware machine is available. Therefore, the host implementation can be considered as a partial analytical model of the target implementation (it may also be extended by host specific concerns).

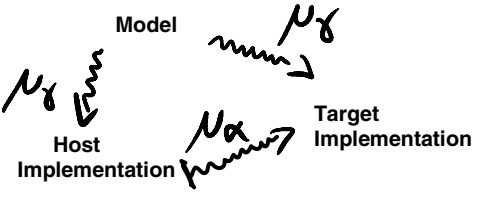


Fig. 9. The host implementation provides information about the target implementation

4.5 Round-Trip Engineering

Code skeletons are generated from UML class diagrams (represented by the μ_γ). Then, developers extend the skeletons by hand. If developers change the structure of the final program (and therefore also the structure of the skeletons which get updated at the same time as they live in the same file), then the class diagram has to be changed accordingly. We model this with a causal μ_α relation between class diagrams and Java skeletons. The causal nature of the relation implies that the model is always up-to-date.

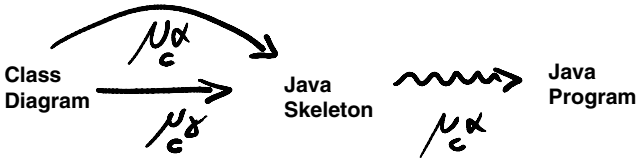


Fig. 10. Using causality to model round-trip engineering

4.6 Model-Based Testing

Model-based testing is performed by generating test cases that can be used to test the program. As represented in Figure 11, the model and the program are developed on one side while the test cases are developed separately. Then, testing consists in checking the consistency between these two views on the system. When an inconsistency is detected, an error has been found.

The test model is a partial representation of the system, with an additional intention of testing (looking for errors) that is not present in the system. The test model is also a partial representation of the model that shares intentions with the model (the concepts manipulated by these representations are the same), but again the test model has this additional test intention. Symmetrically, the model is a representation of the system. The model is then used to generate parts of the program.

When the test model is rich enough, test cases can be automatically synthesized from this model, according to a test adequacy criterion. Thus there exists a μ_γ relation

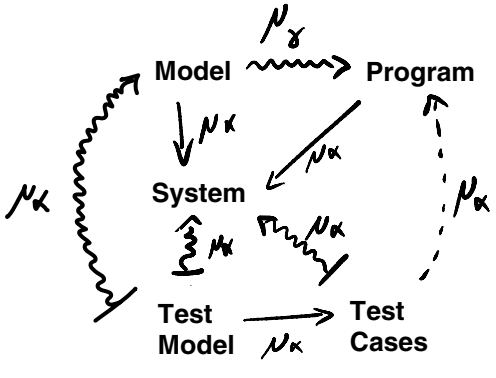


Fig. 11. Model-based testing

between these things. This particular relation also implies that the μ_α relation between the test model and the system is propagated to the test cases that are thus also representations of the system.

The last interesting relationship that appears on the figure is that test cases are representations of the program since they can provide information to analyze the presence of errors in the program. However, these two things do not share any intention since test cases aim at detecting errors in the program while the program aims at providing functionalities to some user.

4.7 Eclipse EMF

This example is drawn from the tutorial T38 "*Introduction to the Eclipse Modeling Framework*" delivered at OOPSLA'06. The tutorial includes generating a working graphical editor to create and manipulate instances of a UML model. The editor is made of three generated Java projects (respectively Model, Edit, and Editor).

The process starts with an XML file that contains a schema which represents a purchase order system. The various modeling artifacts are represented in Figure 12.

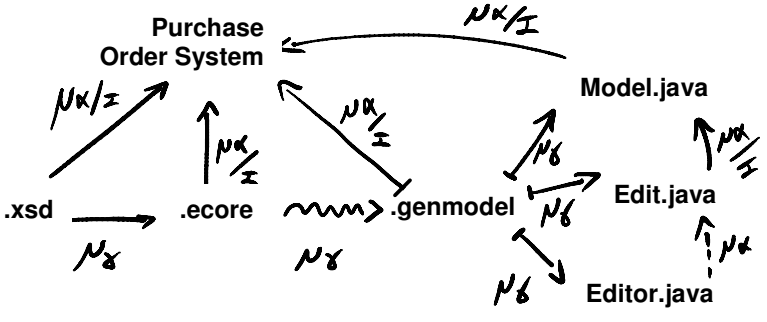


Fig. 12. Purchase order Eclipse EMF tutorial

The XML schema (.xsd file) is a μ_α representation of the system (wrt. a given intention I). The schema is used to generate an EMF model (.ecore file). The model and the schema share the same intention I, as shown by μ_α/I relations. The model is then used to generate a generation model (.genmodel) which is also in a μ_α relation with the system. The .genmodel contains additional information (wrt. the model) to drive the code generation process; therefore it is the target of a partial μ_γ relation. Three Java projects are generated from the generation model: model, edit, and editor. Edit.java is a Java projection of the model, thus it is a μ_α/I representation of the system as well. Edit.java contains general editing mechanisms (not dependent on the graphical user interface) and uses the java projection of the model (represented with another μ_α relation). Finally, Editor.java provides end-user editing facilities to visualize models, using a tree-based explorator.

5 Conclusion

In this paper we have analysed various definitions of models, as found in the related works, and we have proposed a modeling language which can be used as a foundation

to represent the various representation relations between models, metamodels and languages.

Our language focuses on representation relations between modeling artifacts, without actually trying to understand the nature of these artifacts. Ignoring the details of their internal structure appears to be very effective because it magnifies the fact that modeling is a matter of relations and roles, and not intrinsic to the artifacts.

We have identified 5 variations of the representation relation (based on their intention), two natures (analytical and synthetical), and taken causal dependencies and transitivity into account. We have illustrated our approach with several simple examples, drawn from the software engineering domain.

From a practical point of view, we hope that this step toward a better understanding of representation relations will serve as a basis for rigorous metamodeling tools, in the same way as relational algebra triggered the development of efficient databases.

References

[1] Ludewig, J.: Models in software engineering - an introduction. SoSyM 2(3), 5–14 (2003)
[2] Bézivin, J., Gerbé, O.: Towards a Precise Definition of the OMG/MDA Frame-work. Presented at ASE, Automated Software Engineering (November 2001)
[3] Brown, A.W.: Model driven architecture: Principles and practice. SoSyM 3(3), 314–327 (2004)
[4] Jackson, M.: Some Basic Tenets of Description. Software and Systems Modeling 1(1), 5–9 (2002)
[5] Kuehne, T.: Matters of (meta-) modeling. SoSyM 5(4) (2006)
[6] OMG, Model Driven Architecture, Electronic Source: Object Management Group, `http://www.omg.org/mda/`
[7] Seidewitz, E.: What models means. IEEE Software 20(5), 26–32 (2003)
[8] Selic, B.: The pragmatics of Model-Driven Development. IEEE Software 20(5), 19–25 (2003)
[9] Steinmüller, W.: Informationstechnologie und Gesellschaft: Einführung in dieAngewandte Informatik, Wissenschaftliche Buchgesellschaft, Darmstadt (1993)
[10] Stachowiak, H.: Allgemeine Modelltheorie. Springer, Wien (1973)
[11] Favre, J.-M.: Foundations of Model (Driven) (Reverse) Engineering: Models - Epi-sode I: Stories of The Fidus Papyrus and of The Solarus. Presented at Dagstuhl Seminar 04101 on Language Engineering for Model-Driven Software Development, Dagsthul, Germany, February 29-March 5 (2004)
[12] Hesse, W.: More matters on (meta-)modeling: remarks on Kuehne's "matters". SoSyM 5(4), 387–394 (2006)
[13] Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principle of Model Driven Architecture. Addison Wesley, Reading (2004)
[14] Fowler, M., Scott, K., Booch, G.: UML distilled, Object Oriented series, 179 p. Addison-Wesley, Reading (1999)
[15] Bézivin, J.: In Search of a Basic Principle for Model-Driven Engineering. Novatica Journal, vol. Special Issue March-April 2004 (2004)
[16] Favre, J.-M.: Foundations of the Meta-pyramids: Languages and Metamodels - Epi-sode II, Story of Thotus the Baboon. Presented at Dagstuhl Seminar 04101 on Lan-guage Engineering for Model-Driven Software Development, Dagsthul, Germany, February 29-March 5 (2004)

[17] Favre, J.-M.: Towards a Megamodel to Model Software Evolution Through Software Transformation. In: Proceedings of the Workshop on Software Evolution through Transformation, SETRA 2004, Rome, Italy, October 2, vol. 127 (2004)
[18] Fokkinga, M.M.: A Gentle Introduction to Category Theory - The calculational approach, University of Twente (1994)
[19] Yu, E., Mylopoulos, J.: Understanding “Why” in Software Process Modelling, Analysis, and Design". In: Proceedings of the 16th International Conference on Soft-ware Engineering (ICSE), Sorrento, Italy, May 16-21, pp. 159–168 (1994)
[20] Venn, J.: On the Diagrammatic and Mechanical Representation of Propositions and Reasonings. Dublin Philosophical Magazine and Journal of Science 9(59), 1–18 (1880)
[21] Shannon, C.E.: Communication in the presence of noise. Proc. Institute of Radio Engineers 37(1), 10–21 (1949)
[22] Molière. Le Bourgeois gentilhomme (1607)
[23] Kuehne, T.: Matters of (Meta-) Modeling. Software and Systems Modeling 5(4), 369–385 (2006)
[24] Gasevic, D., Kaviani, N., Hatala, M.: On Metamodeling in Megamodels. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 91–105. Springer, Heidelberg (2007)