HAL
open science

Modeling Modeling Modeling

Pierre-Alain Muller, Frédéric Fondement, Benoit Baudry, Benoit Combemale

To cite this version:

Pierre-Alain Muller, Frédéric Fondement, Benoit Baudry, Benoit Combemale. Modeling Modeling Modeling. Software and Systems Modeling, 2012, 11 (3), pp.347-359. 10.1007/s10270-010-0172-x . inria-00538459

HAL Id: inria-00538459

https://inria.hal.science/inria-00538459

Submitted on 22 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Modeling Modeling Modeling

Pierre-Alain Muller[1], Frédéric Fondement[1], Benoît Baudry[2], Benoît Combemale[3]

[1] Université de Haute-Alsace
Mulhouse, France
{pierre-alain.muller, frederic.fondement}@uha.fr
[2] INRIA Rennes Bretagne-Atlantique
Rennes, France
benoit.baudry@inria.fr
[3] Université de Rennes 1
Rennes, France
benoit.combemale@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. We also define a structural definition for this language in the form of a metamodel and a formal interpretation using Prolog. 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 [1] 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 explicit description of the relations among modeling artifacts. Therefore we propose to focus on the very heart of modeling, particularly 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 between these things.

In this paper, we emphasize the major importance of the intentional nature of modeling and introduce intention as a first-class property of the representation

relation between two things used in a modeling process. In this, we relate to Rabelais' famous words about scientific knowledge: "science without conscience is the soul's perdition" [2]. A similar remark is made more recently in the IT domain, by Rothenberg [3]: "It is widely recognized that the purpose of a model must be understood before the model can be discussed". These quotes emphasize the paramount importance to distinguish between information contained by models (to be considered as absolute facts) and the intention of models (to be considered as relative to the purpose models were made for). This means for instance, that information that was created with some intention in mind can later be reused for some other intention.

This work 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 represent different intentions when representing a thing with another thing in a modeling process. This canonical set contains 5 kinds of representation relations that may be refined with nature (analytical / synthetical) and causality (correctness / validity). The initial purpose for the definition of these relations and an associated graphical syntax is to ease and structure reasoning about modeling. In particular it is essentially meant as a language for exchanging ideas when talking around a modeling task. Since we would like this language to be mostly used for communicating and debating about the nature of modeling, it has to be flexible but it also needs a precise definition. This paper extends our work published in MODELS'09 [4] with a formal interpretation of the composition laws between representation relations and a metamodel for the modeling modeling language.

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 introduces a metamodel for our language and section 5 discusses possible uses of this metamodel. Section 6 illustrates the use of the notation via several examples excerpted from the software engineering field. Section 7 draws some final conclusions and outlines future works.

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 [5] 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*".

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**.*" [6]
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.*" [7]

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."* [8]
Kuehne	"*A **model** is an **abstraction** of a (real or language based) **system** allowing predictions or inferences to be made*." [9]
Ludewig	"***Models** help in developing **artefacts** by providing **information about the consequences of building those artefacts** before they are actually made*." [5]
OMG	"*A **model** of a system is a **description** or **specification** of that system and its environment for some certain purpose*." [10]
Seidewitz	"*A **model** is a set of statements about some **system under study** (SUS)*." [11]
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."* [12]
Steinmüller	A model is information: on something (content, meaning), created by someone (sender), for somebody (receiver), for some purpose (usage context). [13]

Table 1: Summary of model definitions

Features of models

According to Stachowiak [14] 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 [12] 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.*" [11].

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*" [15]. 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*" [5]. This is also acknowledged by Michael Jackson: "*The model is not the reality*" [8]. 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 [16]. He coins this the *Janus View*. This is close to the opinion of Bran Selic, in [12] 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 [9]. 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 [17], 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 [18] suggests a similar distinction based on three levels of models, namely *Conceptual Models*, *Specification Models* and *Implementation Models*.

Rothenberg [3] distinguishes between two purposes for models: descriptive models that are build to describe or explain the world and prescriptive models built to discover and prescribe optimal solutions. The author identifies a number of possible usages for models: projection, prediction, allocation and derivation, as well as hypothesis testing, experimentation, and explanation.

Definition of relations between models

Bézivin identifies two fundamental relations coined *RepresentationOf* and *ConformantTo* [19]. Jean-Marie Favre shows in [20] 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 [21], 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 [11], 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 we designate a representation of what we manipulate when we use modeling techniques. When modeling, we essentially build things that represent other things with a particular intention. The following proposal for modeling modeling thus proposes to capture different relations between things that we manipulate when modeling. In particular it focuses on modeling artifacts and does not deal with the definition of these artifacts. The relation between models and metamodels is thus outside the scope of this work. 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 [22]. 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 view, 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 [12], 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 [23], and distinguish between a *source* thing (that many authors call the model) and a *target* thing (called *original* by Stachowiak [14]), 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.

We are using on purpose a very simple graphical 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). We also follow a naming convention: we use upper-case roman letters for the name of "things" and Greek letters for relations.

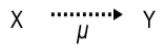
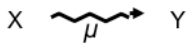
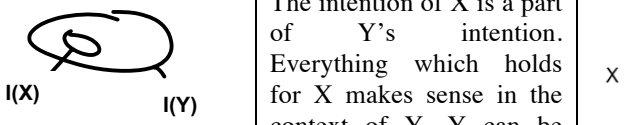
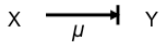
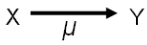
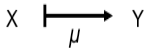
$$X \xrightarrow[\mu]{} Y$$

Figure 1: X is a representation of Y

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 [13].

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

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

We can think about this as the *intention* of a thing. Intentional modeling [24] 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.

It is important to notice that intentional modeling must not be confused with modeling in intension[1]. Intention (with a 't') refers to the reason why a thing is made

[1] http://www.cse.buffalo.edu/~rapaport/intensional.html

or the mental purpose a modeler wants to achieve when building a representation of a thing. On the other hand, modeling in intension (with an ‘s’) aims at capturing the set of properties that are shared by all elements contained in a representation of a thing. For example, in set theory, a set is defined in intension if it is defined as the properties satisfied by all elements in the set. This can be opposed to an extensional definition of the set, which consists in enumerating all elements in the set.

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. Hence, it is enough to say that artifacts have an intention. The intentional flavor of models has also been used by Kuehne [25] in his description of metamodeling and by Gasevic et al. in their extension of Favre's megamodel [26]. The consequences of intentional thinking applied to modeling can be understood and represented using Venn diagrams [27]. Table 2 summarizes 5 kinds of μ-relations and associated notation.

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 [24] 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.

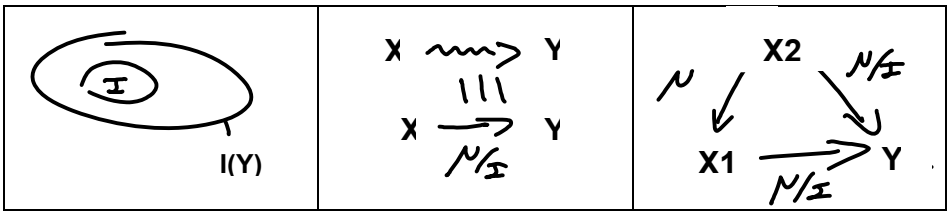


Table 3: Notation shortcut. X is a complete representation of Y, for a given subset of the intention (in a given context)

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 [11] and Favre [23]).

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

$$x \xrightarrow[\mu_\alpha]{} y \;:\; \exists\, T_\alpha \mid x = T_\alpha(R(y))$$

where T_α is a relation such as X can be derived (or abstracted) from R(Y), with R being a representation of X (including the Identity). 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 representation 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:

$$X \xrightarrow[\mu_\gamma]{} Y : \exists T_\gamma \mid Y = T_\gamma(R(X))$$

where T_γ is a relation such as Y can be derived (or generated) from R(X), with R being a representation of X (including the Identity). 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 derived (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.

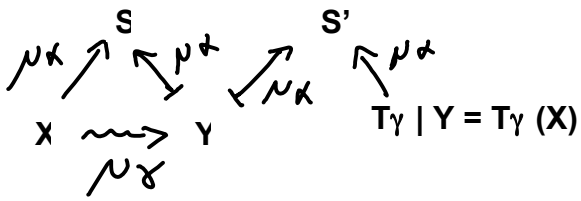


Figure 2: Explicit representation of the contribution of the transformation used to generate Y from X

Causality

Causality addresses the synchronization concern raised by Michael Jackson [8]; 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 [15]; 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 [11], 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 [28]. 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*" [16]. 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.

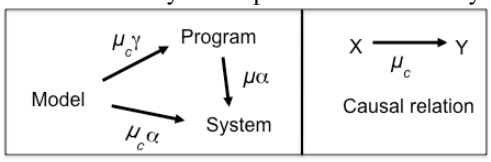


Figure 3: Causality implies maintaining the representations over time

Transitivity

Transitivity addresses the composition properties of similar μ-relations. Transitivity is realized when the intention of a composed 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.

We used Prolog in order to reason about the recursive transitivity between μ relations, and benefit from its inference abilities. In our implementation, we distinguish two predicates for the representation relation: a direct one (called `directrepresentation`) that specifies an explicit and direct representation between two things (i.e., a μ-relation), and a potentially transitive one (called `represents`) between two things (i.e., a μ *-relation). The representation kind of the latter can be inferred by transitivity from a sequence of μ -relation using the first one.

We have formally specified transitivity rules between representation kinds as Prolog clauses in order to infer the representation kind between two things that are related through n direct representations.

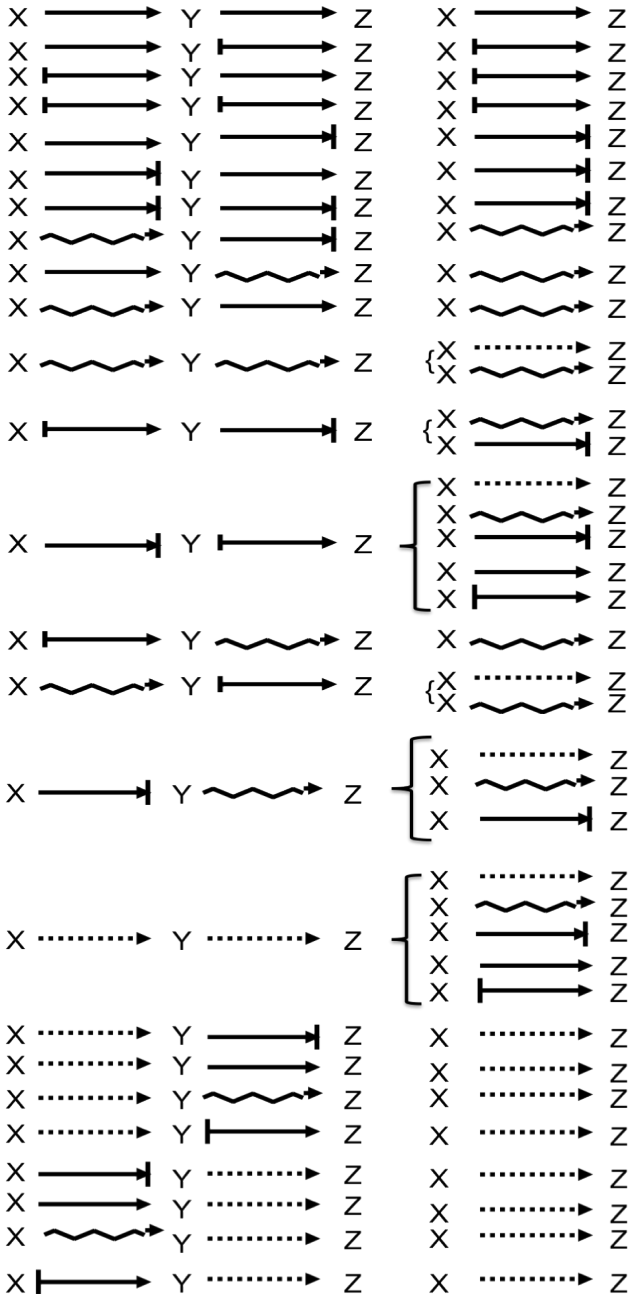


Table 4: Composition law for representations

We have identified three main regular clauses (that we expressed as Prolog rules) in a sequence μ* of μ-relations:

1. if all μ-relations in μ* are of the same kind K then, the composed μ*-relation is of the same kind K. There is one exception to this rule: when all relations are of kind `share`, the composed μ*-relation can be of kind `share` or `different`.
2. if a μ-relation of kind `same` (X and Y share the exact same intention) appears in μ*, then it has no impact on the kind of the composed μ*-relation.
3. if a μ-relation of kind `different` (X and Y share absolutely no intention) appears in μ*, then the kind of the composed μ*-relation is `different`. There is one exception to this rule: when all relations in μ* are of kind `different` then the composed μ*-relation can be of any kind.

All μ* sequences that don't fall in one of these cases have to be dealt with separate clauses. These cases are expressed as 12 specific Prolog rules.

Using this translation of μ-relations in Prolog, we have inferred the composition laws given in Table 4. We can remark that 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 (kind `allInSource`) and if Y has the same intention as Z (kind `same`), then X is an extended representation of Z (kind `allInSource`). In some cases there are 2, 3, 4 or 5 possible results when composing relations. For example, Figure 4 (corresponding to line 11 of Table 4) illustrates the two situations that can occur when X is an extended and partial representation of Y (kind `share`) and Y is an extended and partial representation of Z (kind `share`). 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 (kind `different`). 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 (kind `share`).

Table 5 : Computing the transitive relation between X, Y, Z

```
1. ?- consult('/tmp/modeling2.pl').
2. true.
3. ?- directrepresentation(x,y,share).
4. true.
5. ?- directrepresentation(y,z,share).
6. true.
7. ?- findall(Kind,represents(x,z,Kind),R).
8. R = [share, different].
```

These two possibilities can be automatically inferred using the Prolog clauses that model the transitivity rules between different types of representations and the built-in predicate `findall` that collects objects resulting from successful computations. Table 5 shows the console output for the previous example. First, the clauses specifying the transitivity rules are compiled - i.e., consulted - (lines 1 &2), independently of any specific example. Then, the predicate `directrepresentation` is used to specify

the concrete example[2] (lines 3 & 6). Finally, the `findall` predicate collects all possible kinds of representation between x and z in the list R (lines 7 & 8). "

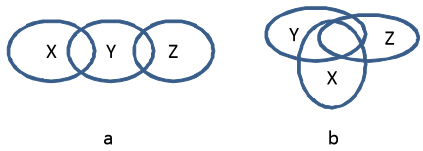


Figure 4 –Intention overlapping when composing partial extended relations

4. Metamodel for modeling

We introduce a structural definition of modeling through the use of a metamodel (Figure 5).

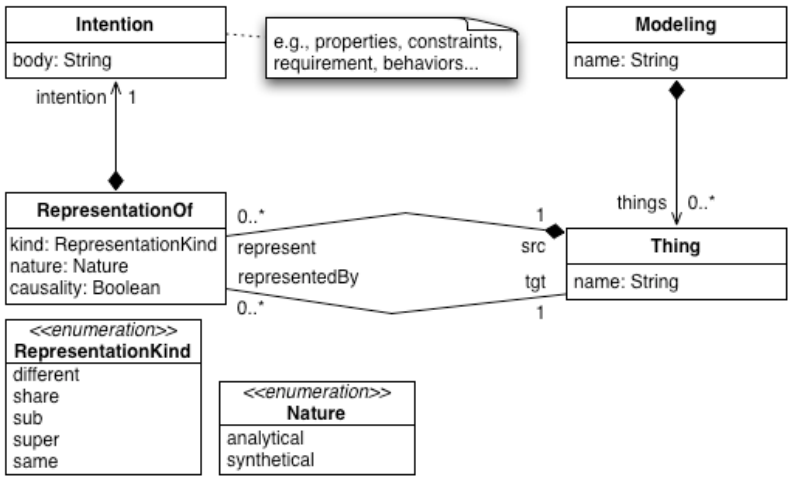


Figure 5 - Metamodel for modeling

A modeling activity (`Modeling`) may be named, and is characterized by a set of named things (`Thing`), without any consideration of their internal structures. The only information captured by a thing is the possible intentional representations of another thing (`RepresentationOf`) corresponding to the μ-relations (contained by the source thing).

Thus, a model of modeling may be considered as a finite directed multigraph (i.e., allowing multiple edges between any two different nodes) without loops (i.e., reflexive edges). The definition of self-representations (i.e., loops) are constrained by the following OCL property:

```
context RepresentationOf inv NoLoops : self.src <> self.tgt;
```

[2] Note that this specification could be automatically generated from the model using our formalism

The μ-relation is mainly characterized by its intention (`Intention`), which captures all or part of the intention of the source thing (`src`). We assume that a thing is defined without any intention, but it is captured through its representation of another thing. In other words, one `RepresentationOf` relation holds the intention of the `src` with respect to the `tgt` (in the metamodel, `src` and `tgt` are the two roles that a `Thing` can play in a `RepresentationOf`). This appears in the metamodel as a `represent` containment reference between `Thing` and `RepresentationOf`, whereas the `representedBy` reference is not a container because the target thing is independent of the representation. Also, we allow the possibility to define a thing without any representation of another thing (and thus unintentionally) but only to consider its internal structure (information), as a black box in our formalism.

We also assume in our formalism a black box definition of the intention of each representation between things (except an informal description using `body`). Thus, no consideration can be made on the content of intentions, and the link with the information of a thing (i.e., its internal structure). However, in the context of one representation, the `kind` attribute allows to characterize the perimeter of the intention of the source thing (`src`) compared to the intention of the target thing (`tgt`), according to the Table 4.

The μ-relation (`representationOf`) is also characterized by its nature (`nature`), used to distinguish analytical and synthetical representations. Finally, the causality of a μ-relation is captured by the boolean attribute `causality` of `representationOf`. We assume that the formalization of the causal relationship appears very important (e.g., for process engineering, megamodeling, etc.) but we consider this concern outside the scope of modeling modeling.

We have built a preliminary version of a graphical editor on the basis of this formal structural definition of modeling. It is implemented with GMF[3] and supports the concrete syntax introduced in Section 3.

5. Practical uses of modeling modeling

We foresee at least two kinds of applications for the work presented in this paper: an abstract one for mental purposes, and a concrete one for tool implementations.

On the abstract side, our intension based modeling approach is a mean to facilitate reasoning and discussions about metamodels and their relations. This is how we use it in this paper, for instance in Figure 12, we show that the structure of a Java program can either be analyzed from the model, or from the program itself, which basically means that the intention of 'understanding the structure of that program' can be shared between the model and the program. The figure also shows that a UML class diagram and a Java skeleton can be two different models of a thing, while being both written in order to represent, analyze and understand the structure of that thing.

On the concrete side, tools - such as model (metamodel) repositories - could be extended by the intentional modeling elements represented in figure 5, so as to capture, maintain and check, intentional properties expressed against the artefacts

[3] The Eclipse Graphical Modeling Framework, cf. http://www.eclipse.org/modeling/gmf/

manipulated by these tools. As an example, let us consider expressing intentional constraints over aspect weaving within a model repository.

The model in Figure 6 can be seen as the specification of a constraint that the model repository may enforce. This constraint states that the order of aspect weaving (respectively weaving either first *Aspect1* and then *Aspect2* into the *Base*, or the inverse, first *Aspect2* and next *Aspect1*) shall not affect the resulting intention. *M'1* and *M'2* have the same intention, this is stated by the two enclosing *same* arrows.

M1 and M2 are built as extensions of the Base, by incorporating intention carried respectively by Aspect1 and Aspect2. The intention of M1 (and M2) includes the intention of the base: this is represented by a `Super` representation relation. The intention of M1 (and M2) does not completely include the intention of Aspect1 (respectively Aspect2); M1 (and so M2) does not have to be an aspect (for instance, it does not bear weaving information), this is represented by a `Sub` representation relation. M1 (and M2) is then further extended by M'1 (respectively M'2). In the end, M'1 and M'2 do have the same intention. This is stated by a `Same` representation relation.

This constraint is not saying that the order in which Aspect1 and Aspect2 are woven does not matter. It is stating that we are interested in preserving intention during the weaving process, and that we are especially interested in the fact that the weaving order will not affect the resulting intention. Thus, this is clearly the specification of a constraint.

As seen earlier, a tool could check such constraint. For instance, if the content of intentions is detailed, then it is possible to use the kind of Prolog clauses to check the preservation of the expected property in a model repository.

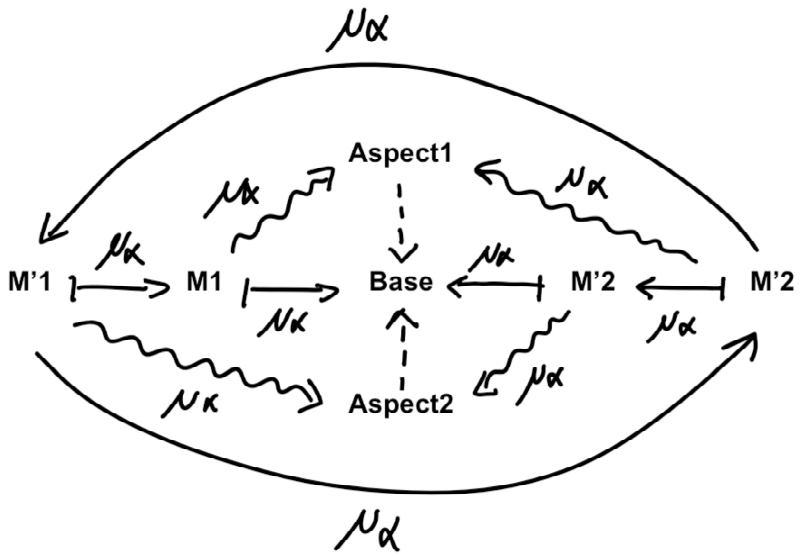


Figure 6: Expressing a constraint over aspect weaving: independence of weaving order

6. Examples

6.1. 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 the information that it contains may be used to have an idea of the color of the real world pipe.

This is an illustration of the point that we emphasized in the introduction. The information contained by the picture was made with some intention in mind. The distribution of colors was produced with a totally different intention, and the dashed arrow reflects this.

So, the dashed arrow represents the intention of modeling. Should this intention be different, for instance to reflect the fact that the distribution of colors could be used to retrieve information about the real world pipe, then the arrow would be different too (we would use a partial representation).

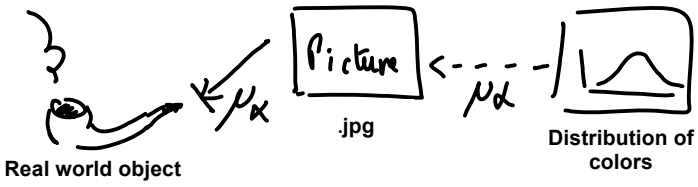


Figure 7: Example of μ_α relations

6.2. Jackson's Problem Domain and Machine

In table 1, 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 8 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).

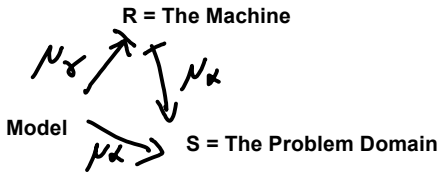


Figure 8: Generated machine implementing a μ_α representation

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*" [8]. This view is also in line with Bran Selic, who states: "*the model eventually becomes the system that it was modeling*" [12].

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 9.

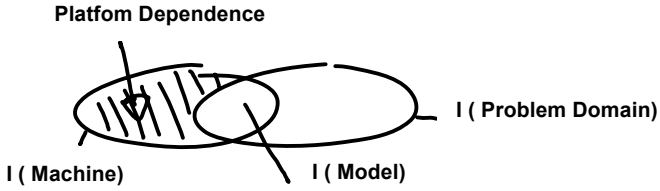


Figure 9: The machine implements the subset of intention of the problem domain, represented by the model

6.3. 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 10 shows how all successive levels of refinement extend the intention of the System, with platform dependent information required for implementation. We also see 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).

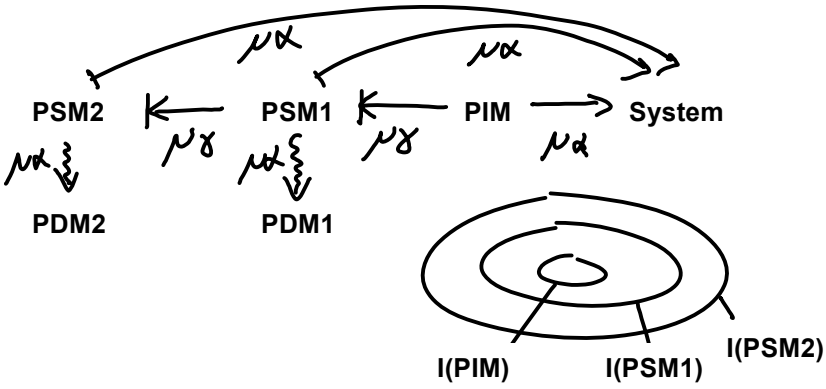


Figure 10: Refinement of PIM into PSM, with platform specific information

6.4. 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).

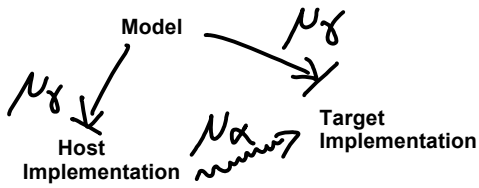


Figure 11: The host implementation provides information about the target implementation.

6.5. 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. The UML class diagram and a Java skeleton are two different models of a thing that share the same intention: they are both written in order to represent, analyze and understand the structure of the thing. 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.

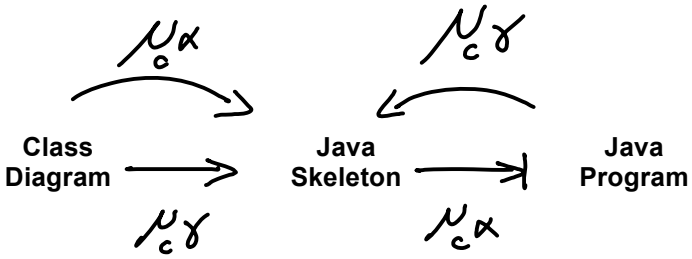


Figure 12: Using causality to model round-trip engineering

6.6. Model-based testing

Model-based testing is performed by generating test cases that can be used to test the program. As represented in Figure 13, 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 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.

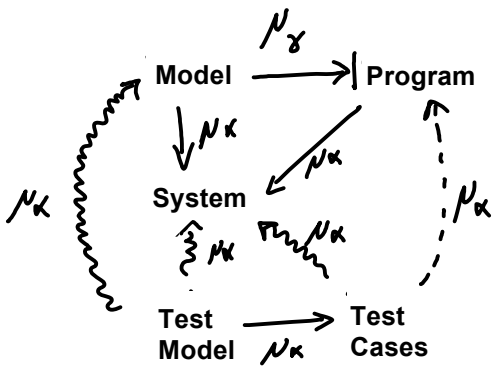


Figure 13: Model-based testing

6.7. 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 14.

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.

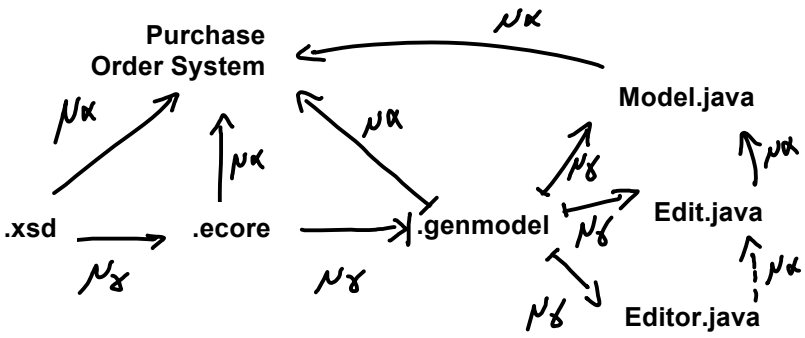


Figure 14: Purchase order Eclipse EMF tutorial

6.8. Modeling Modeling Modeling

This paper is entitled "modeling modeling modelling". This is to reflect the fact that the presented work is about building a formal model (F in the picture) of a language (L in the picture), which in turn is a representation for a set of models of systems (M and S in the picture). This journal paper (modelling modeling modeling) extends the conference paper (modeling modeling) by a third level of modelling. Hence, this third modeling is the contribution of this paper.

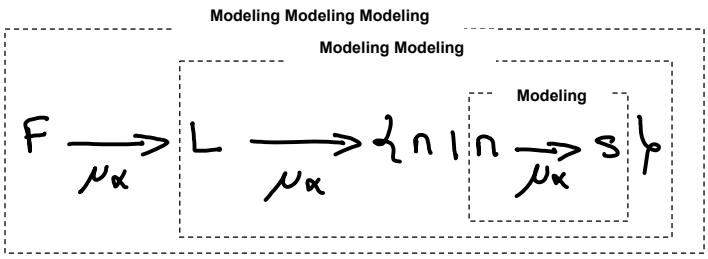


Figure 15: Modeling modeling modeling

7. Conclusion

This work analyzes various definitions of models, as found in the related works, and proposes 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 kinds of representation relation (based on their intention), two natures (analytical and synthetical), and taken causal dependencies and transitivity into account. We have also introduced a formal definition of the domain of modeling modeling as well as precise semantics for relations between things that are manipulated when modeling. 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 intention-aware metamodeling tools, in the same way as relational algebra triggered the development of efficient databases. One step in that direction would consist in formally capturing the operational semantics of the modeling modeling language.

Acknowledgements

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, Christophe Gaston, and Régis Fleurquin. We would also like to acknowledge the invaluable comments of anonymous reviewers of an earlier version of this paper.

8. References

1. Molière, *Le Bourgeois gentilhomme*. 1607
2. Rabelais, F., *Les horribles et épouvantables faits et prouesses du très renommé Pantagruel Roi des Dipsodes, fils du Grand Géant Gargantua*. 1532
3. Rothenberg, J., *The nature of modeling*, in *AI, Simulation and Modeling*, L.E. Widman, K.A. Loparo, and N.R. Nielsen, Editors. 1989, John Wiley & Sons. p. 75–92.
4. Muller, P.-A., F. Fondement, and B. Baudry. *Modeling Modeling*. in Proceedings of *MODELS'09*. 2009. Denver, CO, USA: p. 2-16.
5. Ludewig, J., *Models in software engineering - an introduction*. SoSyM, 2003. **2**(3): p. 5-14.
6. Bézivin, J. and O. Gerbé, *Towards a Precise Definition of the OMG/MDA Framework*, in *ASE, Automated Software Engineering*. 2001.
7. Brown, A.W., *Model driven architecture: Principles and practice*. SoSyM, 2004. **3**(3): p. 314-327.
8. Jackson, M., *Some Basic Tenets of Description*. Software and Systems Modeling, 2002. **1**(1): p. 5-9.
9. Kühne, T., *Matters of (meta-) modeling*. SoSyM, 2006. **5**(4).
10. OMG. *Model Driven Architecture*. 2003 [cited 2006; Available from: http://www.omg.org/mda/
11. Seidewitz, E., *What models means*. IEEE Software, 2003. **20**(5): p. 26-32.
12. Selic, B., *The pragmatics of Model-Driven Development*. IEEE Software, 2003. **20**(5): p. 19-25.

13. Steinmüller, W., *Informationstechnologie und Gesellschaft: Einführung in dieAngewandte Informatik*. 1993, Darmstadt: Wissenschaftliche Buchgesellschaft.
14. Stachowiak, H., *Allgemeine Modelltheorie*. 1973: Springer, Wien.
15. Favre, J.-M., *Foundations of Model (Driven) (Reverse) Engineering : Models - Episode I: Stories of The Fidus Papyrus and of The Solarus*, in *Dagstuhl Seminar 04101 on "Language Engineering for Model-Driven Software Development"*. 2004: Dagsthul, Germany.
16. Hesse, W., *More matters on (meta-)modeling: remarks on Kühne's "matters"*. SoSyM, 2006. **5**(4): p. 387-394.
17. Mellor, S.J., K. Scott, A. Uhl, and D. Weise, *MDA Distilled: Principle of Model Driven Architecture*. 2004: Addison Wesley.
18. Fowler, M., K. Scott, and G. Booch, *UML distilled*. Object Oriented series. 1999: Addison-Wesley.
19. Bézivin, J., *In Search of a Basic Principle for Model-Driven Engineering*. Novatica Journal, 2004. **Special Issue March-April 2004**.
20. Favre, J.-M., *Foundations of the Meta-pyramids: Languages and Metamodels - Episode II, Story of Thotus the Baboon*, in *Dagstuhl Seminar 04101 on Language Engineering for Model-Driven Software Development*. 2004: Dagsthul, Germany.
21. Favre, J.-M. *Towards a Megamodel to Model Software Evolution Through Software Transformation*. in Proceedings of *Workshop on Software Evolution through Transformation, SETRA 2004*. Rome, Italy: p.
22. Fokkinga, M.M., *A Gentle Introduction to Category Theory - The calculational approach*. 1994: University of Twente.
23. Favre, J.-M., *Foundations of Model (Driven) (Reverse) Engineering: Models - Episode I: Stories of The Fidus Papyrus and of The Solarus*. 2004, Dagstuhl Seminar 04101 on "Language Engineering for Model-Driven Software Development: Dagsthul, Germany.
24. Yu, E. and J. Mylopoulos. *Understanding "Why" in Software Process Modelling, Analysis, and Design*. in Proceedings of *16th International Conference on Software Engineering (ICSE)*. 1994. Sorrento, Italy: p. 159-168.
25. Kuehne, T., *Matters of (Meta-) Modeling*. Software and Systems Modeling, 2006. **5**(4): p. 369-385.
26. Gasevic, D., N. Kaviani, and M. Hatala. *On Metamodeling in Megamodels*. in Proceedings of *MODELS'07*. 2007: p. 91-105.
27. Venn, J., *On the Diagrammatic and Mechanical Representation of Propositions and Reasonings*. Dublin Philosophical Magazine and Journal of Science 1880. **9**(59): p. 1-18.
28. Shannon, C.E., *Communication in the presence of noise*. Proc. Institute of Radio Engineers, 1949. **37**(1): p. 10-21.