MODELING SITUATIONS TO DESCRIBE A MODELING PROCESS

THIS DOCUMENT IS AN ENGLISH TRANSLATION OF THE FOLLOWING ARTICLE PUBLISHED IN FRENCH: ANTOINE BEUGNARD, FABIEN DAGNAT, SYLVAIN GUÉRIN ET CHRISTOPHE GUYCHARD. DES SITUATIONS DE MODÉLISATION POUR DÉCRIRE UN PROCESSUS DE MODÉLISATION. *Ingénierie des Systèmes d'Information*, 20(2): 41 – 66, 2015. DOI:10.3166/ISI.20.2.41-66.

Antoine Beugnard, Fabien Dagnat
IMT Atlantique, Lab-STICC, UMR 6285, Brest, France first.last@imt-atlantique.fr

Sylvain Guérin
ENSTA Bretagne, Lab-STICC, UMR 6285, Brest, France sylvain.guerin@ensta-bretagne.fr

Christophe Guychard
Openflexo christophe.guychard@openflexo.org

March 26, 2024

ABSTRACT

We propose to identify modeling situations highlighting elementary actions on modeling artifacts such as models and metamodels. We apply a modeling process to a detailed use case. We believe that identification of these basic situations helps better understand the modeling and therefore modeling tools' requirements. We present Openflexo a *free modeling* tool that helps us implement this approach.

Keywords Modeling, process, notation

1 Introduction

Modeling is the basis of most problem-solving strategies, especially in the scientific and engineering domain Morris [1967], Lachapelle and Cunningham [2007]. We model by relying on writings[1] in the form of sentences in the case of textual models or drawings for diagrams. These sentences and drawings, artifacts resulting from the modeling, must respect rules of construction that are more or less explicit or construction rules that are more or less explicit or formalized.

In "On the art of modeling" Morris [1967], W. Morris proposes to move from an intuitive modeling process to an explicit approach. He illustrates the article with a problem of transport planning. In addition to the reflection stages, he shows two stages that are found in any modeling process:

1. Identify a specific instance of the problem;
2. Determine their generalization or abstraction, and define their representations, through mathematical variables, for example.

He notices that the production of these artifacts (the examples and symbols) follows a process of elaboration by enrichment:

> The process of model development may be usefully viewed as a process of *enrichment* or *elaboration*. One begins with very simple models, quite distinct from reality, and attempts to move in

[1] Even if the first philosophers discussed on the Agora without leaving written records and that our ancestors had to solve many problems before the invention of writing...

> evolutionary fashion toward more elaborate models which more nearly reflect the complexity of the actual management situation.

This view, by enrichment only, is simplifying. In most cases, enrichments are followed by reductions (filtering, selection) of models. Finally, W. Morris also highlights the need for links (implicit or explicit) between these artifacts:

> *Analogy* or *association* with previously well-developed logical structures plays an important role in the determination of the starting point of this process of elaboration or enrichment.

More recently, for problem-solving learning, the authors of Lachapelle and Cunningham [2007] propose the following explicit cyclic process:

- 1. ASK – What is the problem? What have others done? What are the constraints?
- 2. IMAGINE – What are some possible solutions? Brainstorm ideas. Choose the best one.
- 3. PLAN – What diagram can we draw to help us here? Make a list of materials needed.
- 4. CREATE – Follow your plan and create it. Test it out.
- 5. IMPROVE – Discuss what works, what doesn't, and what could work better. Modify your design to make it better, Test it out.

In this article, we focus on the part of the process that uses tools: PLAN,CREATE and IMPROVE.

In the remainder of this article, we refer to an explicit representation of a system or a problem as a *model* and the set of explicit rules – whatever their mode of explicitation – that a model must respect as a *metamodel*. We shall see that the links between model and metamodel can be known *a priori*, but are sometimes elaborated as the result of a modeling choice *a posteriori*.

The formalization of the management of the writing of the modeling tools has been concretized, for sentences, by the theory of languages and grammars, for drawings, by the so-called model-driven approaches (MDE). In both cases, the representation of a model produced (sentence or drawing) must conform to the rules expressed respectively in the grammar or the metamodel (in the MDE sense). This relation of conformity is one of the two possible relations between models and metamodels. We will therefore consider two levels of modeling: the model (example, instance, concretization) and the metamodel (generalization, abstraction). In the sense that we use it, the metamodel Kleppe [2007] is the expression of the concepts (lexicon or vocabulary) that can be used to build a model, all the rules and constraints (grammar) on these concepts and the set of writing rules to represent them (concrete syntax). Our approach is abstract and can be considered as *categorical* in the mathematical sense of the term. We are interested in the relations between models and metamodels without taking into consideration their internal representation and the way they are realized or stored.

In most existing modeling tools and approaches, the conformity relation is considered as strict *i.e.*, the model keeps exactly and only the properties defined in its metamodel (which must pre-exist), but we do not retain this assumption in our approach. Indeed, it sometimes turns out to be a prescriptive constraint that prevents a designer from fully describing his model because the metamodel has not foreseen it. We will show that the relationship between a model and its metamodel is also developed during the modeling process. Conformity is not necessarily an initial data, it is built with the model. In Kelly and Pohjonen [2009],Kelly and Pohjonen cite several bad practices linked to conformity imposed *a priori* (Predetermined Paradigm, If You Have a Hammer. . . , The Library Is the Language, Sacred at Birth, etc.). Thus, in our approach, the conformity can be chosen as prescriptive or not, and this, on all or part of the model, depending on the needs of the designer. Tools can be devised to check whether a specified non-prescriptive conformance specified between a model (element) and a metamodel (element) is true.

Finally, one also often encounters the instantiation relation (*instanceof*) which links a model created from a metamodel to this metamodel. This relation admits in most modeling tools a definition which expresses conformity to the metamodel in the sense already seen, but adds that the realization of the instance is obtained directly according to the form defined by the metamodel (which becomes a kind of instance mold). It should be noted that this vision of the instantiation relation is not as restrictive in ontology-based approaches. This gives rise in Kühne [2006] to an instantiation relation divided into two sub-concepts: linguistic and ontological instantiation. In the context of this article and following the categorical view already outlined, we are not interested in the realization of modeling artifacts. We therefore do not restrict the definition of the instantiation relation to the (linguistic) one inspired by object programming languages.

To show the need for a living modeling approach with various interactions between models and metamodels and evolving metamodels, we present some examples (part 2). We follow by detailing a case study (part 3), conducted with

a local authority. Then, to reason on the dynamics of modeling and to serve as a basis for the definition of requirements for the modeling tools, we identify uses of these two levels of modeling through situations (part 4) encountered during modeling work by illustrating them with concrete examples. We quickly discuss the sequence of these situations (part 5) and discuss them. We then present the approach followed with the Openflexo framework (part 6) which allows us to experiment with our vision of free modeling. Finally, we compare our approach with other works in the part 7 before concluding.

2 Modeling examples

To begin with, we describe real-life situations of modeling with professionals. Through these examples, we wish to show how models are developed when the metamodel is not known (or shared). Such a model cannot be compliant *a priori*. Nonconformity is an initial necessity and conformity the result of the modeling activity. Models are developed with the purpose of constructing a "representation" and a shared "interpretation" (meaning) for the purpose of communicating (or calculating) with oneself or with others.

2.1 Modeling the activity of a local authority

A local authority brings together very different populations. On the one hand, the elected officials who define the orientations, develop the strategic vision and who have a perception of the organization of the community. They rarely need to know how things are done. On the other hand, services departments have a precise functional hierarchical organization that puts the "how" into action, and provides information to the elected officials to help them make decisions.

This vision is pure theory. Resistance exists, information is hidden in both directions, power issues are at stake. In the current economic context, the optimization of the functioning of these organizations becomes crucial and demands for tools appear.

We are participating, in collaboration with a local authority in the development of a tool that will facilitate the matching of the two visions of the functioning of this community. Each comes with its own conceptualization, sometimes its own tools, to describe this complexity. Our job is to build languages and the associated tools to allow for a common interpretation. It is inconceivable that either party would adopt the other's model, but it is fundamental to find an area of consensus.

The dialogue with each of the actors leads to a simultaneous emergence of representations built in the form of drawings or spreadsheets. Each one of them answers, in addition to the need for communication with the other actors, to the concerns of each party. Our work consists, first of all, in identifying with them the conceptual elements allowing the common understanding (the consensus), and to differentiate them from the elements specific to each class of actors (concepts and writing elements). The next step is then to work on this set of concepts and representations in order to link them, constrain them, interpret them and to build tools for their manipulation.

The result we are trying to obtain must allow each person to appropriate the information and to manipulate it in their own language (with different representations) without distorting it (conceptual conformity), while relying on existing data sources: organizational charts, maps of the community's missions, inventory of strategic directives, etc. The result is several domain-specific languages, articulated around intersections (in the sense of common elements) on which the interpretation is shared. The correctness of the exchange is then guaranteed because no change of interpretation is necessary.

This case study is developed in the sections 3 (method) and 6 (tools) to highlight the approach and the modeling situations described in the section 4.

2.2 Using a DSL to describe the interaction of components

Integration of software components within the same multi-platform system (different languages, different target architectures) requires a perfect mastery of the interfaces between these components. This implies sharing the definition of these interfaces with all the actors in charge of the development of these components, while respecting the specificities of each target system.

This situation, encountered in an industrial company developing complex embedded systems, led us to work on the definition of a dedicated Domain Specific Language (DSL) to describe these interfaces. A collection of generators, co-constructed with the specialists of each platform, exploits this language to produce specific code for each target platform.

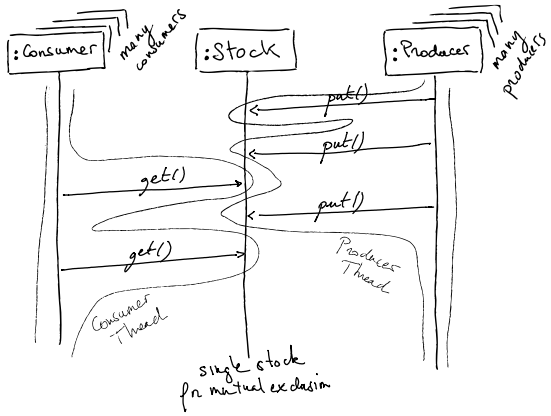

Figure 1: A handmade and enriched UML sequence diagram

The success of such an approach depends on the ability of the language to capture the concerns of its user community. Although it brings together engineers whose disciplines have a certain proximity, the communities present a diversity of uses, practices and constraints that make it difficult to establish a consensus.

2.3 Modeling with UML

The UML modeling language is widely considered to be a leading software system modeling standard for modeling systems with preponderant software. It covers a large part of the software life cycle, from analysis and detailed design to testing. It is used in teaching as well as on a large scale in industry.

However, it is interesting to observe that between its first version and its second one new concepts have appeared (components, constraints) and new notations too. Let's also stress that some extension proposals have been made, for example, to add colors Coad et al. [1999] or introduce elements of method elements Cheesman and Daniels [2000]. Users (especially students and their teachers) also know how to *twist* some notations or add some annotations to specify the meaning of some entities.

The figure 1 illustrates how a UML sequence diagram can be enriched with manual annotations, not provided by the tools, to show the notion of activity flows and multiple instances to illustrate mutual exclusion. Paper and pencil are still very good modeling tools.

2.4 Discussion

These experiments show that whatever the spectrum of a language, users always bring their language, bring their point of view and adapt it to their needs. To be satisfied with what is offered by a modeling framework is to adopt a way of thinking that if it is perfectly adapted to the need may be sufficient, but if it is not perfectly adapted, may force one to *twist* one's way of thinking. Is this acceptable?

The situation is the same with programming languages and the paradigms they concretize. Some problems can be represented very efficiently (and solved efficiently) by thinking in terms of objects, others by thinking functional, or synchronous, or declarative, etc. Thus, the design of a programming language is a matter of compromises and choices that necessarily reduce their spectrum of application Hoare [1973], Mitchell [2003].

Finally, let's remember what Abraham Maslow (1908 – 1970), an American psychologist, noted : *"When the only tool you have is a hammer, every problem begins to resemble a nail"*.

During modeling activities, the simple solution is to adopt a common metamodel and to make all users adhere to it. This solution can only be used in specific cases, often when the stakeholders know each other and already share an important culture. Most of the time, it is necessary to build a common interpretation, and it is only when the differences are encountered that we realize that they exist. This difficulty becomes central when one does not know the users who will work with these models, a common situation in companies.

It should also be noted that a modeling language always has limits and is therefore only well suited to the production of a certain class of models. In order to choose a modeling language, it is necessary to know in advance the form of the

models that we want to produce. This makes it difficult to choose a language *a priori* when we do not know well the domain of the systems we want to model.

Thus, in the case of the most formal models, the semantics is perfectly defined, and users will have to comply with it. For example, a compiler imposes the meaning of the program and can, at runtime, provoke the surprise of a user that does not share the interpretation. Any change of compiler or even of version of compiler can also cause surprises. In any cases, the compiler becomes the guarantor of the interpretation on which users can rely on to check the possible interpretation(s) of a model (the program), possibly by experimenting. For modeling languages whose semantics are less formalized or does not come with tools, this interpretation arbiter is not available.

When we tackle a new problem, or when we meet new people, the modeling process becomes iterative. Starting from a given choice of representation (based on the history or knowledge of the people involved) if the interpretation is shared[2] the objective is achieved. If not, the interpretation must be changed. Two cases arise: either it is possible, with the chosen representation, to converge towards a new interpretation, or it is impossible. In the second case, it is necessary to evolve the representation (new colors, new symbols, new grammatical structures, new words, etc.), or by creating a new representation and linking it (both in the representation and in the interpretation) to the initial representation.

For a model designer, the ideal situation is one where metamodels exist prior to model. We have shown with these three examples that there are many situations where existing metamodels are not sufficient. In these cases, we propose the identification of elementary modeling situations where the model and its metamodel (and their representations) are co-constructed.

3 Case study: the case of a local authority

Our case study, whose context is that of the example 2.1, is typically one of those where the starting situation is a blank page in terms of metamodeling.

None of the people we spoke to in the local authority were familiar with the principles of MDE, a minority had used modeling tools (acoustic processing for one, and process modeling for the other), and only one person had used a relational database engine.

In this type of context, it is difficult to impose an existing modeling language. Especially since we had to integrate other constraints such as: the existence within the community of a shared (although not formalized) vocabulary, and the desire to reuse existing graphic charters.

The data at our disposal were heterogeneous, both in form and in origin:

- A spreadsheet containing the activities mapping data;
- A representation of the same data in the form of a mind map;
- The document presenting the strategic objectives (unstructured data);
- A few free diagrams presenting the framework for use;
- Some examples of documents using this data: framework reports, activity reports, support for popularizing/promoting the local authority's actions, etc.

The project team wanted to reuse as much of this existing material as possible to avoid introducing too many changes in usage within the organization. Several objectives were set for the project:

- To facilitate the maintenance of the data and their appropriation by the producers;
- To help align the *strategic objectives* with the *activities*;
- To produce graphic representations for different stakeholders;
- To open up new uses, by cross-referencing with other data.

We have therefore worked to develop a set of tools based on the capabilities of the Openflexo infrastructure (introduced in the section 6). This has the advantage of providing the means to dynamically capture and interpret conceptual (meta-)models and representation models (diagram definitions or forms). During our working sessions, this allowed us to co-construct and test our models with the local authority team.

[2]This situation is difficult to detect in an absolute way. An assumed consensus is used, beliefs... must sometimes be questioned.

Figure 2: Identification of concepts (DÉCLINAISON, MISSION) from an example

The three examples that follow illustrate modeling situations that we have encountered in the course of this work. They correspond to a phase of analysis in which we drew up the constituent concepts of the cartography from the documents provided.

Some concepts have emerged quite simply from their representation in the documents provided. As illustrated in figure 2, the work done in this case consisted of:

Step 1 Isolate from the examples a representation which was agreed upon;

Step 2 Identify the concepts[3] (DÉCLINAISON, MISSION);

Step 3 Produce example diagrams to validate the choices made.

During discussions, some concepts also appeared without direct representation. The figure 3 illustrates this case:

Step 1 Identification of a new concept (DIRECTION) during exchanges on a document;

Step 2 Definition of a representation for this new concept;

Step 3 Creation of an example diagram for validation.

The example diagrams produced by the users were sometimes used to identify concepts that had not appeared in the first place. As in the case of the figure 3, they have sometimes changed the color of a particular shape, thereby distinguishing a new concept and invalidating the conceptual model.

In this type of case, the method we applied to review the models is illustrated on the figure 4 :

Step 1 Differentiate the two representations that make sense;

Step 2 Associate a new representation with the pre-existing concept (here [DIRECTION]);

Step 3 Construct a new concept from the other representation (here [PÔLE]).

In addition to the work on the conceptual model and its graphic representations, we also had to implement the identified concepts onto the databases (spreadsheets) used by the local authority. This phase enabled us to specify the rules for interpreting and producing the data. The interpretation of these rule models by the tooling then allows the

[3] We have voluntarily kept the names in French. They only have illustration purpose.

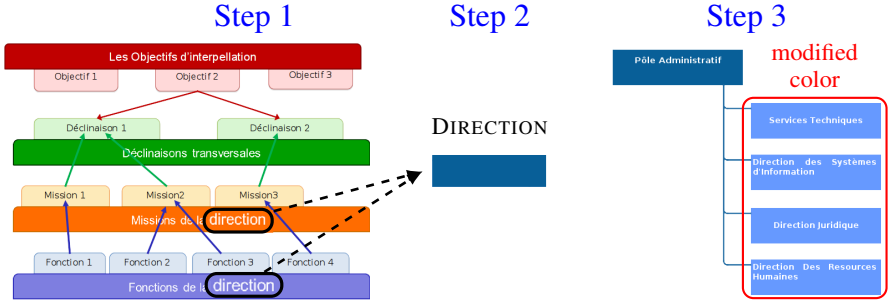

Figure 3: Emergence of a concept with no representation (DIRECTION)

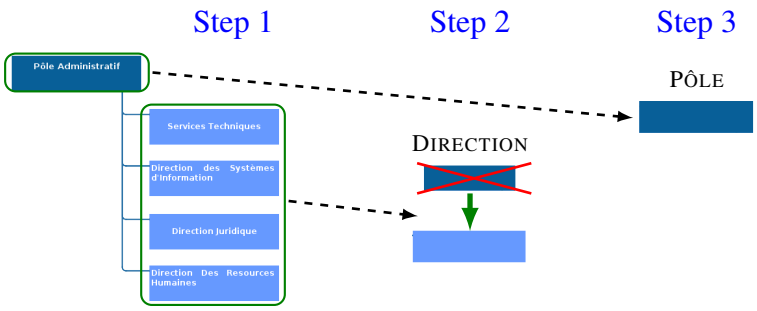

Figure 4: Correction of the conceptual model from an example diagram (introduction of PÔLE)

manipulation of the data in the spreadsheets through graphic representation, as well as the generation of diagrams from the spreadsheets.

At the end of the experimentation with the local authority, we managed to build an environment and a working method that allow the team members to be more autonomous in developing their tools. This collaborative modeling process has been well supported by the tooling we are developing.

In this study we have not used the terms *model* or *metamodel*, but rather 'examples' and 'concepts'. In the following section, we use this case to identify elementary situations of modeling.

4 Modeling situations

To describe modeling situations, we have two types of artifacts to consider: models and metamodels. The situations vary according to the order in which these artifacts appear or are linked in the process. We simplify the description by considering only one actor in each situation.

A more detailed analysis of these situations could be interesting by taking into account different actors and therefore different intentions in the process. The figures 5, 6 and 7 present the 12 situations under consideration, with an arrow indicating the elements that are created. We will also use the words *instance* to describe an element at the model level and *concept* for an element at the metamodel level. Each of the situations is described in detail and illustrated in one of the following subsections by a concrete example.

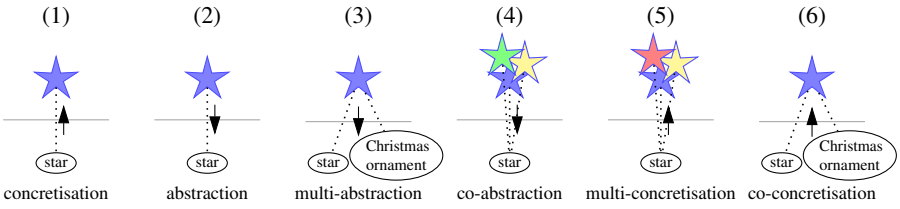

Figure 5: Modeling situations (high: model level, and low: metamodel level)

(1) A metamodel exists, the job is to produce a model [*concretization*].
(2) A model exists, the job is to find a metamodel [*abstraction*].
(3) A model exists, several metamodels must be found [*multi-abstraction*].
(4) Some models exist, the task is to find a common metamodel [*co-abstraction*].
(5) A metamodel exists, the job consists in building several models [*multi-concretization*].
(6) Some metamodels exist, the task consists in building a model [*co-concretization*].

(7) A model and a metamodel exist, they must be linked [*interpretation*].
(8) Some models and metamodels exist, the work is to link them [*co-interpretation*].
(9) A model exists, the job is to build another model (without any metamodel). [*exemplification/extension*].

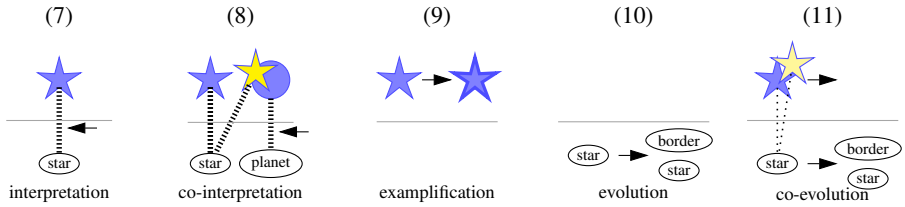

Figure 6: Modeling situations (high: model level, and low: metamodel level)

(10) A metamodel exists, the work consists of building another metamodel (without any model) [*evolution/extension*].

(11) A metamodel exists with several of its conforming models, the work consists in evolving the metamodel (previous case) by adapting (or not) its models [*co-evolution*].

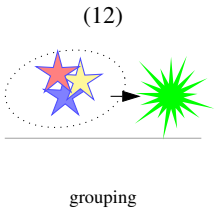

Figure 7: A grouping situation (model level)

(12) A set of models is grouped together to form a whole [*grouping*].

The cases (9) and (10) are probably equivalent in the context of a multi-level interpretation of the model. Indeed, at a given level of abstraction, the absence of reference to another element, more concrete or more abstract, makes the work equivalent. We differentiate them because most tools offer very different means of manipulating these two levels, based on their mode of representation.

The case (12) allows adding details in a model independently of its metamodel. This situation allows, for example, to relate; a concept (which plays the role of a relation) is grouped with other concepts which play the other roles of this relation. We consider this situation only at the model level, by sliding[4] it would be applicable at the metamodel level.

We illustrate and comment on these different situations in the following sections.

4.1 Concretization

Concretization is the most classical form of use of a modeling tool. The designers of the tool have prepared editors adapted to a set of concepts (metamodel). Users build models according to the rules provided. This approach is very effective when the model to be built is adapted to the metamodel used, *i.e.*, the necessary concepts are present. In the case where the concepts do not line up exactly, users are sometimes forced to twist the interpretations or imagine unintended uses of the concepts.

Example 1 *In the context of our case study, this situation corresponds to the production of example diagrams, after a representation has been chosen for each concept: the concretization then produces a graphic form.*

4.2 Abstraction

This case, which is also very classic in a modeling approach, consists, from an example, in identifying the concepts and rules to build a metamodel to which the initial model conforms. The result is used by tool developers to build model editors.

Example 2 *In our case study, this situation is illustrated on figure 2.*

[4] See the 5.2 for a definition.

4.3 Multi-abstraction

This less classical case generalizes the situation of abstraction, where the aim is to both abstract and organize the abstraction by bringing out complementary points of view on the model (2 in situation 3 of the figure 5). We do not differentiate between metamodels, one can pre-exist or they can all be designed at the same time. The important thing is that there are several metamodels.

Example 3 *In the case under study, the situation illustrated by figure 4 produced a multi-abstraction. Indeed, the concepts of [Pôle] and [Direction] partially overlap. A [Pôle] being essentially a [Direction] which groups together several of them.*

4.4 Co-abstraction

This practice is a generalization of the abstraction presented in 4.2. Several models are used as examples to build a metamodel to which all examples will conform. Taxonomic or classification approaches are situations of co-abstraction. The interest of using several example models is to confront the concepts to bring out consensus or, on the contrary, differences in interpretation and conceptual conflicts.

The co-abstraction and multi-abstraction situations can be combined into an unrepresented multi-co-abstraction. We will discuss "chaining" of situations in the section 5.

Example 4 *In our case study, this situation arose in every analysis of a series of examples. For example, each time the identification of a concept needed to cross-check both the spreadsheet data and the framing documents provided.*

4.5 Multi-concretization

This case is a generalization of the first. The same metamodel is used repeatedly to produce different models. The risk is that when different interpretations are used to produce the models, inconsistencies may appear without being detectable. The risk of this approach is increased by the fact that there exists several interpretations of the metamodel as for example in UML with its multiple points of semantic variation OMG. For example, the semantics of state diagrams is not precisely defined in UML which may lead to various interpretations Chauvel and Jézéquel [2005].

Example 5 *In the case study, the attempt to validate the concept [Direction] (figure 3) used multi-concretization to highlight a defect in the model.*

4.6 Co-concretization

This situation arises when several experts, each with their own metamodel, come together to describe a system in which their point of view must be represented. In this situation one would like to leave the abstractions separate and not build a common metamodel.

Example 6 *We did not encounter this situation in the case study.*

4.7 Interpretation

This case, although less usual, undoubtedly contributes to the modeling approach. An example model exists as well as a metamodel. This situation is common and relies on the knowledge of concepts and their organization by an expert. The work consists of linking one or more elements of the example to a concept present in the metamodel in order to :

- either ensure that the example respects the rules of the metamodel ;
- or produce a counter-example (the example model) to the metamodel in order to evolve it.

Example 7 *In the case study, after the identification of the concept of [Direction] without representation in a document provided, this concept could have been recognized in another document; this new example would then have been interpreted as the concept [Direction].*

It should be noted that the interpretation relation is not the instantiation relation. Instantiation is based on a mechanism for constructing a specific representation. The same concept can be instantiated in multiple ways. Interpretation evokes the intention to relate a representation and its meaning, defined and described in a metamodel.

4.8 Co-interpretation

This case is a generalization of the previous case. Several example models exist as well as several metamodels. This situation is based on the knowledge of the concepts and their organizations by an expert. The work consists in linking the elements of the models to one (or more) concept(s) present in the metamodels in order to :

- either ensure that the examples respect the rules of the metamodels;
- or produce a counter-example to the metamodels in order to evolve them.

Example 8 *During the case study, the intersection of the points of view between* elected representatives *and* services members *gave rise to several co-interpretations.*

4.9 Exemplification/Extension

This is often the first situation encountered. The aim is to produce examples of representation that serve as a basis for reflection. It is possible to start from a blank page or from previous examples. For drawings or diagrams, colors, shapes, and relative positions of shapes will be chosen to represent the problem. The excellent article by D. Moody Moody [2009] reviews many examples of graphical representations. A history of representations is presented by M. Friendly in Friendly [2005], Friendly and Denis [2001]. For languages, an extraordinary variability of languages and grammars is observable in the dictionary of languages Peyraube et al. [2010]. It should be noted that the exemplification is carried out in the absence of the identification of an associated concept, without interpretation.

Example 9 *In our case study, the starting point is a set of documents describing examples.*

4.10 Evolution/Extension

This situation, parallel to the previous one, can only be implemented once the conceptualization has been completed, *i.e.* when a metamodel is available. It is a question of evolving the concepts or the rules governing them. This work is often carried out – explicitly or not – with reference to model evolutions.

Example 10 *Several times during the case study, we evolved the models and metamodels.*

4.11 Co-evolution

This is a related situation, where the metamodel evolves and is linked to existing models; what happens to the existing models? The article Sprinkle and Karsai [2004] is one of the first to identify the problem with the vocabulary of model-driven engineering.

This case is complex and can take many forms. For example, if a new concept is introduced in the metamodel without any instance of this concept having been previously created, the solution is trivial. On the contrary, if the structure of a concept must be adapted and instances had already been created, the solution may not be automatic and require the intervention of a human to guide the evolution.

Example 11 *During our study, we did not go so far as to make the local authority's databases evolve as a result of (meta-)modeling evolutions.*

4.12 Grouping

This situation is very common and fundamental. It is a kind of exemplification[5] (situation 9) which makes it possible to show a new instance by grouping. The example thus identified can, if necessary, be interpreted (situation 7), if the concept of the group is recognized, or abstracted (situation 2), if the concept of the group is to be invented. This operation remains at the instance level.

Through the grouping mechanism, it is possible to encompass all relationships (in the entity-relation sense). It is sufficient that, at the time of each grouping, to identify the *roles* the instances must (or should if they do not exist yet) play in the relation.

Example 12 *In the case study, the directorates have missions and are broken down into clusters. The highlighting of these relationships are groupings.*

[5] without the notation dimension

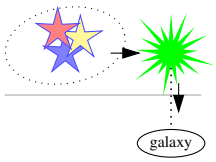

Figure 8: A grouping and a concept identification are chained together

4.13 Synthesis

The twelve situations presented fall into five main categories:

From concept to instance. This is the most frequently *proposed* approach in tools. The toolbox brings its share of concepts to concretize in order to model. The metamodel pre-exists and the modeler refers to it. When concretizations is impossible, some tools allow the set of concepts to be extended before concretization. The instance does not exist without the concept.

From instance to concept. This is the approach most often *used*, before moving on to the previous category. We draw on a board, a sheet of paper or in a drawing tool. The concepts are identified, then we change tools and go from concepts to instances.

Interpretation acknowledgement. The intermediate approach allows the independent existence of concepts and instances, to finally recognize their interpretation link.

Composition. Independent of the interpretation link, composition applies to instances as well as to concepts. It introduces the notion of composition, which makes it possible to represent associations, relations, content, etc.

Evolution. The evolutions highlight the dynamics of construction of the two levels involved: instances and concepts. They can be independent of the interpretation link or not.

5 Various ways of sequencing

Once the modeling situations have been identified, the question of their sequence[6] naturally arises. We will only illustrate a few examples of this broad problem. In particular, we do not consider cases of cooperative modeling where several people could sequence situations where it would be necessary to manage the synchronization of their actions.

5.1 Sequencing as a sequence of situations

The multi-concretization (respectively multi-abstraction) situation can be seen as a composition of several concretizations (respectively abstractions). This is not necessarily the case, as grouping several actions in the same situation is not exactly the same as reproducing *independently* the same action several times.

Different situations can also be sequenced. The figure 8 shows a classic sequence of two situations where one begins by grouping (12) a set of models and then abstracting (2) them into a concept.

5.2 Sliding as a stack of abstractions

A model can sometimes be interpreted as a metamodel and lead to the creation of models of different “levels”. Conversely, a metamodel can also be seen as the concretization of a metametamodel. We call this change of point of view a shift. It is therefore possible to pile up the situations of concretizations, abstraction or interpretation that are sequences of shifts.

The figure 9 shows a formulation of the algebraic process in 2 steps. The first (left) consists in identifying a problem (the model) and a theory (the metamodel). The second step (right-hand side of the figure 9) consists in producing a formula which represents the problem and which is a concretization of the theory and an interpretation of the model for the problem considered. We see here that we combine (and pile up) two elementary situations: concretization and interpretation.

The situations 12 of grouping, 9 of exemplification and 10 of evolution apply to a single level. It is natural to be able to consider them on another level after having changed the point of view by sliding.

[6] we do not use the term composition, but the term sequence, which highlights the temporal dimension more clearly.

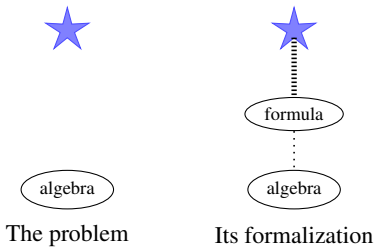

Figure 9: A sequence to represent an algebraic approach

5.3 Multilevel grouping

One might seek to interpret the interpretations as groupings between concepts of different levels. This is true, but their nature deeply rooted in the semantics of modeling makes us consider them as separate; they participate in the definition of the meaning of modeling. This is why we only consider the links of concretizations (1), abstraction (2) and interpretation (7) between these two levels without seeking to further abstract (or generalize) through a situation of grouping between even more abstract concepts.

5.4 Sequence of grouping

We present only one grouping axis. This is a special case. In the general case, there are many possible axes of grouping where the same model element could belong to several groupings. Tools that would put this modeling view into practice would also have to manage information about the grouping "logics" to help the user. Among the grouping information, the representations (languages) and business rules (constraints) are also essential.

6 Tools

In order to validate the relevance of our modeling situations, and to support our case study, we have designed a tooling that relies on a software infrastructure developed by the Openflexo[7] company. The situations presented above are, in principle, all feasible using this infrastructure. In reality, we have only used 10 of the 12 situations described. Situations 11 and 12 (co-evolution, grouping) require software developments to be fully addressed.

In the remainder of this section, we present the *free modeling* environment that has been used. Then we will describe how it was applied on the case study, illustrating concretely some of the modeling situations.

6.1 FreeModelling Editor

The Free Modeling Editor (FME) is a prototype designed to experimenting with new ways of interacting with models. It is used to facilitate the emergence of a graphical syntax simultaneously with the associated conceptual model.

Its interface, illustrated by the figure 10, is divided into a *drawing area* part on the right, and a *conceptual model* and *instances* part on the left. The user can freely draw in the space in the center by placing graphic shapes selected from one of the palettes shown on the right.

Two concept browsers are provided on the left side of the interface. Instances appear categorized under the concept name in the first browser. Graphical forms (instances) not yet associated with a concept appear under the term *None*. All defined concepts appear in the second browser.

It is possible to create a concept using a context menu in the second browser. In this case, the new concept is not associated with any graphic form. The association of graphic shapes with concepts is done from the drawing area. A contextual menu accessible from a selected shape allows this. A context menu is available from a selected shape to enable this association. If the corresponding concept does not yet exist, the user can define a new one.

For the creation of graphic shapes, three types of palettes are available to the user:

1. A palette allowing the re-use of shapes already defined in the drawing currently being edited ("Used shapes");
2. A palette offering shapes already associated with concepts ("Concepts");

[7] `openflexo.org`

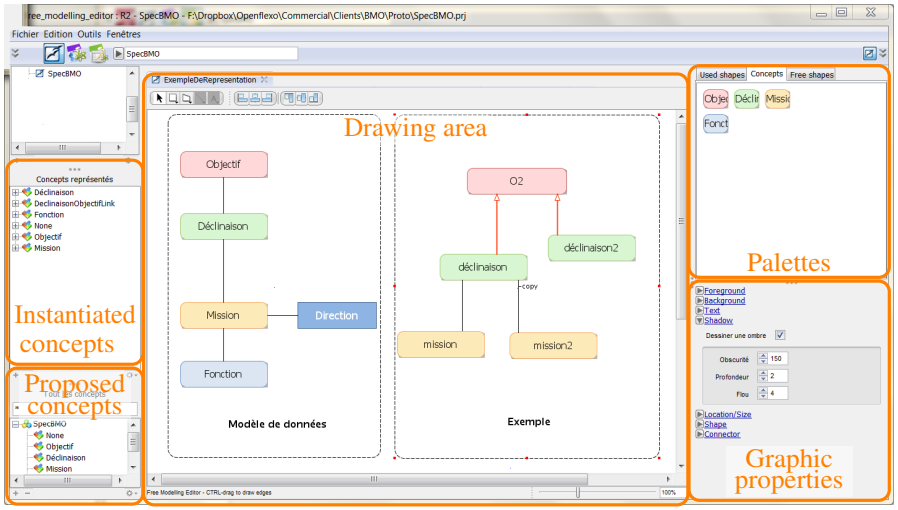

Figure 10: *Free Modeling Editor* organization

3. Simple predefined shapes (“Free shapes”).

The behavior of the tool differs depending on the palette used:

- Choosing a shape from the “Concepts” palette triggers the creation of a graphic shape on the drawing, and a new instance of the concept concerned, linked to the graphic shape;
- The choice of a predefined shape or the reuse of an existing shape only triggers the creation of a graphic shape.

Each creation of a shape whose graphic properties vary from those already present on the drawing causes a new entry to appear in the palette “Used shapes". The graphic properties of the used shapes are all adjustable. It is also possible to use images supplied by the users.

It should be noted that the association of a graphic form with an instance is only partially prescriptive in the FME. Indeed, when an instance is created with a prescribed form (the one associated with the concept), the user can always modify it. In this case, the form associated with the instance differs from that associated with the concept. This situation has been encountered in the case study and is illustrated in figure 4. Tools are also made available to restore the prescribed forms, or, on the contrary, to replace the form associated with a concept and impose it on all the other instances.

Given the flexibility of the tool, it is easy to experiment with the different modeling situations described above with the FME. The separation of workspaces and responsibilities leads to a tool operation that allows practicing the first 10 situations described. In the following section, we describe how it was applied in the case study, by illustrating the modeling situations encountered.

6.2 Implementation on modeling situations

The *free modeling* tool was at the heart of the modeling method presented in 3. The first diagrams used were initialized using the import functionality of *Powerpoint* slides. At first, it was used as a simple drawing tool to:

- re-use the examples provided by the community,
- bring out all the representations that illustrate the problem.

In this case, it is essentially the situations of exemplification (4.9) and grouping (4.12) that have been activated. The diagrams were reworked by the users until the representations were suitable, an evolutionary situation described in (4.10). This also led to the deletion of certain elements, a situation not addressed in our work of identifying situations.

In a second step, starting from the diagrams obtained, we were able to bring out all the concepts by using the support of the *FME* for all the situations of abstraction (abstraction 4.2, multi-abstraction 4.3 and co-abstraction 4.4).

The validation phase of this second stage required the use of another tool in the Openflexo infrastructure: the *View Editor*. This allows the creation of new example diagrams from the conceptual and representation models produced by the *FME*. The new diagrams produced therefore conform to the established specifications (concepts and representations). The modeling situations invoked here are the concretization (4.1) as well as the multi-concretization (4.5).

As illustrated by figure 4 on the case study, it was sometimes necessary to make the conceptual model evolve following this initial validation phase. The situations evolution (4.10) and interpretation (4.7) were used in this case:

- Evolution of the representation: change of the color of a graphic form ;
- Evolution of the conceptual model: identification of a new concept;
- Interpretation of the instance whose representation has been changed as an instance of the new concept.

It was also sometimes necessary to go beyond the functionalities of the *FME*[8], in order to respond to user requests in situations such as co-evolution (4.11). However, it was possible to go through with the process, and to have a conceptual model and various diagram specifications that we were able to re-use in the rest of the process.

7 Related works

We approach related work from two perspectives. The first concerns work on modeling situations that are investigated by a few empirical studies. The second is related to reflections on the nature of modeling itself.

In Hermans et al. [2009], the authors study the success factors of using DSLs. In a very limited setting – Windows Communication Foundation (WCF) services – the authors studied 30 projects using ACA.NET[9]. The study focuses on the interest of using a DSL and not on how to define one, even if the factors studied such as usability, expressiveness or reusability are related to it. The article concludes on the value of DSLs for productivity and reliability and suggestions to improve reusability.

Conversely, in Kelly and Pohjonen [2009], Kelly and Pohjonen present the main pitfalls of Domain Specific Modeling (DSM) using 75 projects in very different domains (avionics, medical, system configuration, etc.). The difficulties encountered such as "Only Gurus Allowed", "Tool: If You Have a Hammer ..." or "Sacred At Birth" reinforce us in the idea of the importance of the modeling situations we proposed (exemplification, interpretation, multi-abstraction) and that (meta)-modeling is a living and iterative process.

Another pitfall, presented in Wouters [2013], concerns the difficulty of sharing the definition of a language with all the actors. It is often more efficient to use examples and representations (graphics, text or semi-structured data depending on the habits of the interlocutors) to establish a common understanding of the underlying concepts. This observation is also shared by approaches such as Canovas Izquierdo and Cabot [2012], which propose to the community of users of a language to design it and make it evolve through stereotypical representations.

Concerning the reflections on modeling, the work of J.-M. Favre Favre [2004], Favre and Nguyen [2005] lay the foundations of the relations between models and metamodels by introducing the relations δ of compositional links, μ of modeling, ϵ of links between an instance and the set of possible instances, χ of conformity, and τ of trace. We can see that the situations we present are refinements of τ, some linked to the conformity χ (situations 1 to 5, 7, 9 and 10), others to the composition δ (situation 12). We put out of the field the relations of this article μ and ϵ. Situations 6 and 8 are not considered. Situation 11 of co-evolution is presented as a composition of μ (or χ) and τ.

In Muller et al. [2010] the authors study other relationships between models by formalizing various intentions with the aim of "facilitating reasoning and discussion about metamodels and their relationships" and to serve as a basis for the development of "intention-aware" metamodeling tools.

C. Atkinson et al. Atkinson et al. [2009] highlights the difference in meaning between conceptual abstraction and syntactic abstraction (of languages) to propose a multi-level modeling approach. We find this distinction in our decoupling between the model and its representation and then between the model and its metamodel. Our description allows multi-level modeling according to the two interpretations of abstraction.

In Thalheim [2010, 2012], B. Thalheim studies the notion of model and proposes a formalization. His approach considers a model as a finite artifact and describes its properties and relations with what it represents and with the person who interprets it. Our vision of the notion of model is more linked to the dynamics of construction during modeling.

[8] by intervening in the models manipulated by the FME through the intermediary of the other tools available within the infrastructure.

[9] `http://www.avanade.com/delivery/acanet`

8 Conclusion

Modeling is a complex process, above all intellectual, but we can provide tools. Paper and pencil (or their equivalent) remain basic tools thanks to their flexibility and the freedom they offer. Computer tools provide additional functionality to check properties such as compliance. However, this can be at the expense of freedom. How can we reconcile verification and freedom? The identification of modeling reference situations can be used as a usage specification for modeling tools. We have observed:

- that modeling should not be reduced to drawing; the notion of metamodel brings verification tools and can guide during the concretization ;
- that a metamodel is built and must be able to evolve; one must be able to go beyond what is planned;
- that metamodel and model are not necessarily linked a priori, but that it is a modeling activity to identify these links.

Modeling requires all situations, combined in various orders. For example, one might start with *exemplify*, then *co-abstract* and *interpret*, then *generalise*, to *exemplify* again to validate the metamodel. A modeling tool must allow working at all these levels: intra-level and inter-level.

Most of the current tools are too restrictive – they force you to adopt a fixed metamodel (or set of metamodels) – or too complex – and working on the metamodel work is a programming activity that often reduces the concepts to the paradigms of computer scientists.

The Openflexo infrastructure proposes a multi-level approach by assembling or federating models Koudri et al. [2013]. It can be used as a simple drawing tool, as a classical modeling tool, but also as a metamodeling tool to elaborate a metamodel without programming and finally to put in relation – interpret – drawings and metamodels.

To extend this work, we are considering a more detailed analysis of the situations, taking into account, for example, the different actors, but also situations that are not constructive but destructive, such as the deletion of an interpretation or the removal of an example. Other situations are certainly interesting, such as the change of level mentioned in the part 5.2, when a model becomes a metamodel or vice versa. Finally, a precise study of the modeling tools should be undertaken with regard to the different modeling situations identified.

References

William T Morris. On the art of modeling. *Management Science*, 13(12):B707–B717, 1967.

C. Lachapelle and C.M. Cunningham. Engineering is elementary: Children's changing understandings of engineering and science. In *American Society for Engineering Education Annual Conference & Exposition*, Honolulu, HI, June 2007.

A.G. Kleppe. A language description is more than a metamodel. In *Fourth International Workshop on Software Language Engineering*, Grenoble, France, October 2007. megaplanet.org.

S Kelly and R Pohjonen. Worst Practices for Domain-Specific Modeling. *Software, IEEE*, 26(4):22–29, 2009.

Thomas Kühne. Matters of (Meta-) Modeling. *Software and Systems Modeling*, 5(4):369–385, 2006.

Peter Coad, Eric Lefebvre, and Jeff De Luca. *Java Modeling in Color with UML.* Prentice Hall, 1999.

J. Cheesman and J. Daniels. *UML Components: A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2000.

C. A. R. Hoare. Hints on programming language design. Technical report, Stanford, CA, USA, 1973.

John C. Mitchell. *Concepts in programming languages*. Cambridge University Press, 2003. ISBN 978-0-521-78098-8.

OMG. UML 2.0 superstructure specification. `http://www.omg.org/cgi-bin/doc?ptc/2003-08-02`, 2007.

Franck Chauvel and Jean-Marc Jézéquel. Code generation from uml models with semantic variation points. In Lionel Briand and Clay Williams, editors, *Model Driven Engineering Languages and Systems*, volume 3713 of *Lecture Notes in Computer Science*, pages 54–68. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-29010-0. doi:10.1007/11557432_5.

Daniel Moody. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, 2009.

Michael Friendly. Milestones in the history of data visualization: A case study in statistical historiography. In C. Weihs and W. Gaul, editors, *Classification: The Ubiquitous Challenge*, pages 34–52. Springer, New York, 2005.

Michael Friendly and Daniel J Denis. Milestones in the history of thematic cartography, statistical graphics, and data visualization. Web document, http://www.datavis.ca/milestones/, 2001.

Alain Peyraube, Emilio Bonvini, and JoÃ«lle Busuttil. *Dictionnaire des langues*. Presses Universitaires de France, 2010.

J. Sprinkle and G. Karsai. A domain-specific visual language for domain model evolution. *Journal of Visual Languages and Computing*, 15(3):291–307, 2004.

Felienne Hermans, Martin Pinzger, and Arie van Deursen. Domain-Specific Languages in Practice: A User Study on the Success Factors. In A Schürr and B Selic, editors, *MODEL 2009*, pages 423–437. Springer Verlag, Berlin, Heidelberg, 2009.

Laurent Wouters. Towards the notation-driven development of dsmls. In Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke, editors, *Model-Driven Engineering Languages and Systems*, volume 8107 of *Lecture Notes in Computer Science*, pages 522–537. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-41532-6. doi:10.1007/978-3-642-41533-3_32. URL http://dx.doi.org/10.1007/978-3-642-41533-3_32.

J.L. Canovas Izquierdo and J. Cabot. Community-driven language development. In *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*, pages 29–35, June 2012. doi:10.1109/MISE.2012.6226011.

Jean-Marie Favre. Towards a Basic Theory to Model Model Driven Engineering. *3rd Workshop in Software Model Engineering*, 2004.

Jean-Marie Favre and Tam Nguyen. Towards a Megamodel to Model Software Evolution Through Transformations. *Electronic Notes in Theoretical Computer Science*, 127(3):59–74, April 2005.

Pierre-Alain Muller, Frédéric Fondement, and Benoit Baudry. Modeling modeling modeling. *Software and Systems Modeling*, 2010.

Colin Atkinson, M Gutheil, and B Kennel. A Flexible Infrastructure for Multilevel Language Engineering. *IEEE Transactions on Software Engineering*, 35(6):742–755, November 2009.

B Thalheim. Towards a Theory of Conceptual Modelling. *Journal of Universal Computer Science*, 16(20):3102–3137, 2010.

Bernhard Thalheim. The science and art of conceptual modelling. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems VI*, pages 76–105. Springer, 2012.

Ali Koudri, Christophe Guychard, Sylvain Guerin, Antoine Beugnard, Joël Champeau, and Fabien Dagnat. De la nécessité de fédérer des modèles dans une chaîne d'outils. *Génie logiciel*, (105):18 – 23, juin 2013.