

# Towards a Basic Theory to Model Model Driven Engineering

Jean-Marie Favre

ADELE Team, Laboratoire LSR-IMAG  
Université Joseph Fourier, Grenoble, France  
<http://www-adele.imag.fr/~jmfavre>

**Abstract.** What is a model? What is a metamodel? What is a language? What is a transformation? How are these concepts related? It is striking to see that, though MDE is supposed to be about precise modelling, MDE core concepts are usually described in natural language or at best, using sketchy UML diagrams. When precise descriptions are provided, it is only to describe a specific technology. But Model Driven Engineering is about supporting multiple Technological Spaces (TS). The concepts of model, metamodel, and transformation must be understood, not only in the context of the MDA TS, but also in the Grammarware TS, the Documentware TS, the Dataware TS, etc. This paper shows how the set theory and language theory could help in understanding essential MDE concepts. A first version of a very rudimentary theory for reasoning about MDE concepts is provided in the form of a megamodel.

## 1 Introduction

MDE is not MDA. The MDA standard from the OMG is just a specific incarnation of the Model Driven Engineering (MDE) approach. MDE is on the contrary an open and integrative approach that embraces many other *Technological Spaces* (TSs) in a uniform way [24]. An important aspect of MDE is the emphasis it puts on *bridges* between technological spaces, and on the integration of bodies of knowledge developed by different research communities. Examples of TSs include not only MDA and MOF, but also *Grammarware* [22] and the BNF, *Documentware* and XML, *Dataware* and SQL, *Modelware* and UML, etc. In each space, the concepts of *model*, *metamodel* and *transformation* take a different incarnation. For instance what is called a "metamodel" in Modelware corresponds to what is called a "schema" in Documentware, to a "grammar" in Grammarware, etc.

From our point of view, to be successful the MDE approach should re-invent the wheel but on the contrary to integrate and/or adapt existing bodies of knowledge from mature technological spaces. Fortunately, MDE is getting more attention among research communities from different fields. For instance, the "model-driven" prefix appears more and more in calls for papers. Most people are still puzzled however. The only thing that is clear, is that entering MDE is not easy. The huge pile of MDA standards is made of more than 2600 pages.

The true essence of the MDE concepts are not described there and the concepts are biased towards the particular orientation of the OMG. For instance, most people make the confusion between object-orientation and model-driven engineering<sup>1</sup>. To understand what is MDE, there are a number of papers and books that are business-oriented (e.g. [18]). But they are not really useful from a research point of view. Other books on the contrary concentrate on MDA concepts and on their use. For instance "MDA distilled" [21] and "MDA explained" [26] provide good introductory material. The "MDA Guide" from the OMG also aims at defining essential concepts [29].

It is striking to see however, that though MDE is supposed to be about *precise modelling*, MDE core concepts are *not* defined through precise models. One can find plenty of metamodels in the literature that describe particular technologies, or tools, but we are not aware of a single model that fully capture the MDE notions at the global level. In such conditions, it is very difficult to understand how the basic concepts of MDE are related, and how they can be mapped to existing Technological Spaces. What is needed is a megamodel for MDE [13]. A megamodel is just a model of MDE concepts. Since this include the concept of "model" and "metamodel", we use the term megamodel to avoid the confusion between the framework and the concepts themselves.

The megamodel presented in this paper is the result of ongoing work leading to a collection of papers addressing different topics. The notion of model have been discussed in [13], the notion of metamodel in [14], and the notion of transformation in [15]. This paper go further and discuss the relationship between the megamodel and two existing theories, namely the set theory and the language theory.

The rest of the paper is structured as following. In section 2, some existing megamodel of MDE are briefly commented. Basic concepts from the set theory and the language theory are then introduced in section 3. A megamodel for MDE is then presented in section 4 and section 5 concludes the paper.

---

1. It happens that object-oriented technologies are used to *implement* MDA standards, but this is just the result of a particular choice. Again Model-Driven Engineering is also about integrating other TS such as Grammarware for instance. And they are no objects there. By contrast, the concepts of model, metamodel, and transformation apply in other TSs. Simply put Model Driven Engineering is not about Object Orientation.

## 2 Existing megamodels of MDE

Let's start with a definition of what is a model. "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." [5]. According to this definition, we can say that this paper is a *model*: MDE plays here the role of the *system under study* using Seidwitz's terminology [35]; this paper is obviously a simplification of MDE; the goal we have in mind is to give an overall view on the MDE concepts. This paper is a model of MDE. So it is for [2][4][11][21][35], etc.

Some readers might be disagree though. Because in the context of MDE, some authors provide a more restrictive definition of models. In [21], Warmer and his colleagues state: "A *model* is a description of a (part of) systems written in a well-defined language. A well-defined language is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer" [21]. We have not found any well-defined model of MDE in the literature. Just books, papers, ideas. Nothing that could be put in the computer to get a useful result. MDE experts claim that models should be precise, but they just use plain english and informal drawings to define what is MDE. For instance an attempt to provide a UML model for MDE was made in a previous version of an OMG document [31]. Unfortunately, this was very informal and almost useless. As a matter of fact, the UML diagrams were not used in the rest of the OMG document [31]. These diagrams were later removed in the next versions. The need to reason about models, metamodels, transformations and their combination has not disappeared however. The "MDA explained" book [21] uses a simple yet appealing notation to depict these concepts and their relationships. Though the pictures in the book are rather suggestive, they have no formal meaning.

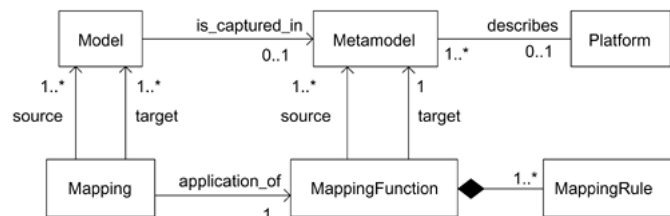


Figure 1 "MDA Distilled" megamodel [26]

By contrast, the "MDA distilled" book [26] provides a simple model of MDE expressed in UML. The full model contains 11 classes and 17 associations. The figure on the left shows only the relevant part here. This megamodel constitutes one of the strong contributions of this book. It greatly helps the reader in understanding what are the important concepts of MDE and what are their relationships. Unfortunately, this megamodel is not

fully consistent with the text, and worse, it provides wrong answers. For instance, according to the diagram above, a metamodel is not a model and there is no such thing such as a meta-metamodel. Describing the metamodel of a mapping function is not considered either, though this is considered as an important aspect in MDE. For instance, the QVT standard will not fit in this schema, first because QVT is a language, and the concept of language is not modelled here, and then because QVT is a language to express mapping functions, and there is no such class like *MappingFunctionLanguage*. The reader might argue that QVT is a metamodel, but this is wrong. A language is not a metamodel (section 4.1). In short, the "MDA Distilled megamodel" is wrong in the sense that it sometimes provides wrong answers about MDE. As a matter of fact, it is neither used nor validated in the rest of the book. Instead of drawing instance diagrams to validate the class diagram above, examples are expressed using an informal notation with no explicit link with the megamodel.

All the megamodels we are aware are sketchy models. None of them describe MDE accurately. While these models could be used to get simple answer to simple questions, they become useless to describe those situations where informal reasoning is not enough. And this is precisely in such situations that precise modelling is required. In fact, it is surprising to see that while everybody in the MDE community advocates for precise modelling, it seems that no-one applies this principle to MDE.

## 3 From the set theory to the language theory

We are in the search of a basic theory for MDE; of something that can help in reasoning about MDE, not in simple cases but in those situations where intuition is not enough. When things turn really complex, the best solution is to use simple and elementary concepts and to deal with complexity by combining these simple concepts in regular structures. Let's forget everything about MDE technology and go back to some of the roots of Computer Science, namely the set theory and the language theory. With this background in mind, a megamodel will then be presented in section 4.

### 3.1 Elements from the set theory: Sets, Pairs, Relations and Functions

Though there are actually various "set theories", we use here simple concepts as defined in the Z language [36]. The Z formulas below are *not* part of the megamodel. They just attempt to *illustrate* the concepts that will be included in the megamodel. The concepts of interest here are *sets* ( $\mathbb{P}$ ), *pairs* ( $\times$ ), *relations* ( $\leftrightarrow$ ) which are basically set of pairs, (partial) *functions* ( $\mapsto$ ) which are relations, and *total functions* which are (partial) functions. The symbol  $\equiv$  is used in Z to define shortcuts. The last column give an example for each notation.

$[X,Y]$	$X$ and $Y$ are "given sets", that is set of elements.	$[RomanNumber]$
$\mathbb{N}$	The set of positive integers. Built-in in $Z$ .	$1 \in \mathbb{N}$
$\mathbb{P} X$	The set of sets of $X$ elements. $\mathbb{P}$ can be read "set of".	$\{1,2,7\} \in \mathbb{P} \mathbb{N}$
$X \times Y$	The cartesian product of $X$ and $Y$ , that is a set of pairs.	$(1,5) \in \mathbb{N} \times \mathbb{N}$
$X \leftrightarrow Y$	The set of relations from $X$ to $Y$ , basically a set of pairs.	$\{(0,1),(1,2),(1,3)\} \in \mathbb{N} \leftrightarrow \mathbb{N}$
	$X \leftrightarrow X == \mathbb{P}(X \times Y)$	
$dom R$	The domain of the relation $R$ . (the set of elements with an image).	$dom\{(0,1),(1,2),(1,3)\} = \{0,1\}$
	$dom R = \{x:X \mid (\exists y:Y \mid (x,y) \in R)\}$	
$ran R$	The range of the relation $R$ .	$ran\{(0,1),(1,3)\} = \{1,3\}$
	$dom R = \{x:X \mid (\exists y:Y \mid (x,y) \in R)\}$	
$A \mapsto B$	The set of functions from $A$ to $B$ . At most one target for one source.	$\{(0,1),(1,3)\} \in \mathbb{N} \mapsto \mathbb{N}$
	$X \mapsto Y == \{f: X \leftrightarrow Y \mid (\forall x:X,y1,y2:Y \mid \{(x,y1),(x,y2)\} \subseteq f \Rightarrow y1=y2)\}$	
$A \rightarrow B$	The set of total functions from $A$ to $B$ .	$+ : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$
	$X \leftrightarrow Y == \{f : X \mapsto Y \mid dom f = X\}$	

In fact a relation is not only a set of pairs. In mathematics, relations are often defined as a triple  $(X,Y,G)$  where  $X, Y$  are sets, here called definition domain and definition range, and  $G$  is the set of pairs from  $X \times Y$ . In  $Z$ , when one states  $div : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{N}$ , then  $\mathbb{N} \times \mathbb{N}$  is the *definition domain* and  $\mathbb{N}$  the *definition range* of the  $div$  partial function. This function is partial because the element  $(2,0)$  has no image:  $(2,0)$  is in the definition domain, but not in the domain.

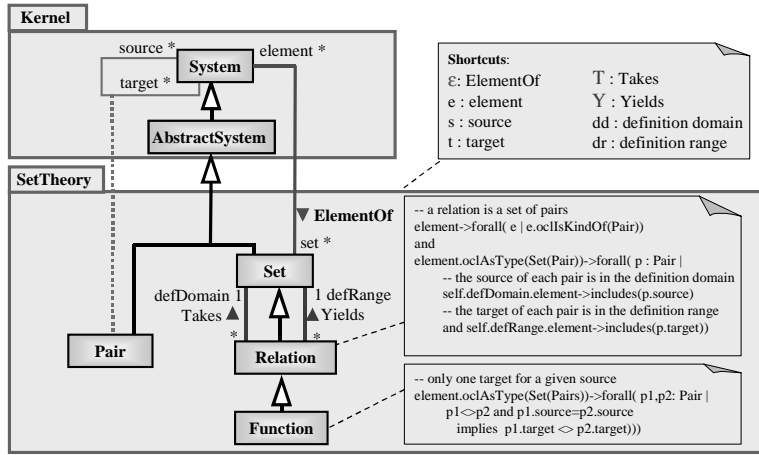


Figure 2 SetTheory package

The concepts from the set theory form one of the basis of our megamodel as shown on the left. The content of the *Kernel* package will be explained in the next section in Figure 5. Note that though the concepts from the set theory are modelled here using the UML/OCL notation there are by no way linked the UML. A set is a set, irrespective of the language it is expressed in. We use UML here because we assume that most people can read this diagram. At the time of writing this paper we are also developing other megamodels with the same meaning but expressed in Prolog, in  $Z$ , and in Hieroglyphics [34]. All these languages are just used as notations,

and we carefully avoid to mix the megamodel concepts with those that come from the language used. For instance in the UML megamodel, the  $\in$  concept is represented as standard UML association named *ElementOf*, not the as the built-in  $\rightarrow$  includes OCL expression. In Prolog, it becomes the *elementOf* predicate, not the built-in *member* predicate. In  $Z$  this is a relation named *ElementOf*, not the built-in  $\in$  symbol. Finally it becomes the  $\in$  ideogram in hieroglyphics [34].

The good way to test a UML model is to describe object diagrams showing particular situations. This is the only way to get reasonable confidence in a MDE megamodel. A model might be good enough, until it is shown that it provides bad answers in a given situation. We have built simple test cases that acts as counter examples to show the invalidity of existing megamodels and none pass the tests.

Object diagrams can also be used to illustrate MDE concepts by considering particular situations. The object diagram shown in Figure 3 conforms to the UML language. This model describes the fact that a function called  $f$  maps the  $IV$  roman number to the integer number  $four$ . The top part of Figure 3 describes the function itself, while the bottom part is the application of the function to a particular value (it states  $f(IV) = four$ ). Figure 3 is fully compliant with UML. The same information can be represented in other ways (Figure 4): (1) using the  $Z$  language to specify this particular situation, (2) using the megamodel in Prolog (symbols in bold are part of the Prolog megamodel), and (3) using the megamodel in  $Z$  (symbols in  $Z$  are parts of the  $Z$  megamodel).

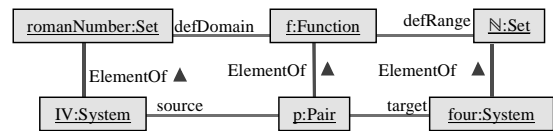


Figure 3 A model representing  $f(IV) = four$

- |     |  |     |  |     |  |
|-----|--|-----|--|-----|--|
| (1) | $[romanNumber]$<br>$iv : romanNumber$<br>$f : romanNumber \mapsto \mathbb{N}$<br>$f(iv) = 4$ | (2) | <b>function</b> ( $f$ ) .<br><b>takes</b> ( $f$ , romanNumber )<br><b>yields</b> ( $f$ , $n$ )<br><b>elementOf</b> ( <b>pair</b> ( $iv$ , $four$ ) , $f$ ) | (3) | $romanNumber, f, n, iv, four : \mathbf{System}$<br>$f \in \mathbf{Function}$<br>$(f, n) \in \mathbf{Yields}$<br>$(iv, four) \in \mathbf{Pair}$<br>$((iv, four), f) \in \mathbf{ElementOf}$ |
|-----|--|-----|--|-----|--|

Figure 4 The " $f(IV) = four$ " model in various languages

The reader might have noticed that there is more information in the UML object diagram than the prolog facts. Some facts have been omitted because they can be automatically deduced thanks to the set theory, which is in fact encoded in each megamodel. For instance the fact `elementOf(iv,romanNumber)` can be derived by the prolog megamodel. Note that this information could also be derived from the UML megamodel, because this model describes the same rule. However, the benefit of the prolog megamodel is that it is executable.

Finally note that though the notion of model is *not* part of the set theory, this notion can be found implicitly in languages that deal with this theory. For instance the Z language introduce the notion of "definition".

Set definitions by comprehension	$\{ x : \mathbb{N} \mid (x+2)^2 > 4 \}$	$\{ x : \mathbb{N} \mid x \neq 0 \}$
Relation definitions by comprehension	$\{ (x,y) : \mathbb{N} \times \mathbb{N} \mid y > x \}$	$\{ (x,y) : \mathbb{N} \times \mathbb{N} \mid x < y \}$
Function definitions by comprehension	$\lambda x : \mathbb{N} . n+1$	

In fact the expressions given above can be considered as models. For instance  $\{ x : \mathbb{N} \mid (x+2)^2 > 4 \}$  is a model representing a set, it is not the set. They are plenty of other models that represent this set.

### 3.2 Elements from the language theory: Languages, Programs, and Interpreters

There is no such thing such as "the definitive language theory", but let's introduce here some of the concepts that are traditionally defined when dealing with programming languages. The relationship between the set theory and the language theory was further described in [17]. Here we will continue to use Z for illustration purposes.

A *language* is a set, at least according to people from the Grammarware community. For instance *RomanNumber* is the language of numbers used by Romans. Similarly *Java* is the set of all java programs.  $\{ "a", "ab", "abb", "abbb", "abbbb", \dots \}$  is the language of the strings that start with an "a" and then continue with some "b". Since languages are usually infinite, reasoning with these languages require to deal with *models of languages*. The string "ab\*" is a model of the language described above. Grammars are models of languages, they are not languages.

*Programming languages* are languages of programs, that is sets of programs, where each *program* is an *executable model of a function*. Not all models of a function are executable. For instance the following signature of a C function "int power(int a, int b)" is a model of that function, but it cannot be executed. If a model of a function is executable, then it is called a *program*. Before to continue let's introduce a simple programming language called *RomanStackPL*. This programming language implements a stack machine with *push* and *pull* operations and *incr*, *add* and *mul* arithmetic operations. Programs take a stack of roman numbers as input and yield a stack.

```
[RomanStackPL,RomanStack]
Plus IFun : RomanStack -> RomanStack
{"incr", "push I; add", "add;push II;mul"} ⊆ RomanStackPL
{"LX.", "LXI.", "II-I.", "III-I.", "I-IV-III.", "X-III."} ⊆ RomanStack
{"(LX.", "LXI."), "(II-I, "III-I."), "(I-IV-III.", "II-IV-III.")"} ⊆ Plus IFun
```

The function *Plus IFun* adds one to the first element of the stack. With the programming language above, there are at least two programs that model this function "incr" and "push I; add". These programs are semantically equivalent. An *interpreter function* of a programming language is a function that takes a program with an input value and that returns an output value that corresponding to the result of the execution of the program. *Interpreter functions* of a programming language *L* that allows to describe programs that transforms *Xs* into *Ys* are therefore characterized by the following pattern:  $(L \times X) \rightarrow Y$ . An example is given below to show that an interpreter function is a mathematical object.

```
InterpreterPLFun[L,X,Y] == (L x X) -> Y
{(("incr", "LX."), "LXI."), (("incr", "II-I"), "III-I."), (("incr", "I-IV-III."), "II-IV-III."), ...
 ("add;push II;mul", "II-I.", "VI."), (("add;push II;mul", "I-IV-III."), "... }
∈ InterpreterPL[FunRomanStackPL, RomanStack, RomanStack]
```

## 4 Towards a theory for MDE

Now that basics of the set and language theories have been introduced, let's define a megamodel for MDE.

### 4.1 Megamodel infrastructure

Let's start with the infrastructure of the megamodel which is totally technology independent. The main package of the infrastructure is shown in Figure 5. So far only mathematical entities have been considered. They are special cases of Abstract Systems. The relationship between the *Kernel* package and the *SetTheory* has been shown in Figure 2. *Abstract systems* are those processed by human minds.

By contrast the dog Fido, which is often cited in the meta-modeling literature, is an example of *Physical System*. MDE aims at building better computer programs and better com-

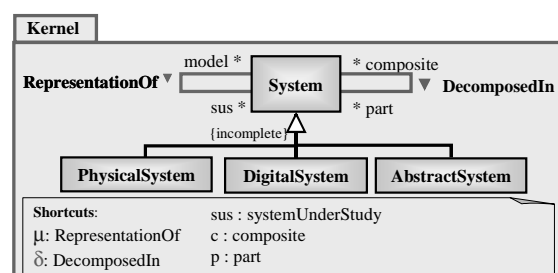


Figure 5 Excerpt of Kernel

puter model. They are *Digital Systems*. This raw classification is not really important [13]. It is just used to show that the term *System* (we could have used any other term) cover a very wide range of artefacts.

Since a system could be a complex thing, it can be *decomposed in many parts* (Figure 5). A system can play the role of *model* with respect to another system, which then play the role of *system under studies* (Figure 5) [13][35]. This relation, called *RepresentationOf* ( $\mu$  for short), have been identified by many authors (e.g. [2][4][35]).

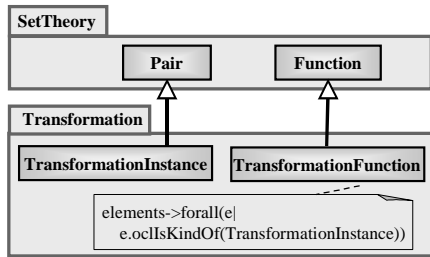


Figure 6 Excerpt of Transformation

The concept of *transformation* is also fundamental to MDE. The definition of this concept can be based on the set theory because the concept of relation is clearly a good abstraction for transformations. There is no consensus however about what is a transformation precisely. The meaning of this term depends on who use it. Many concepts that are closely related, but that are distinct, can be put under this common terminology. For instance the package *Transformation* contains the concepts of *TransformationInstance* and *TransformationFunction* (Figure 6). Figure 5 and Figure 6 are linked by the fact that Pairs and Functions are *AbstractSystems* as shown previously in Figure 2. We call *TransformationInstance* the application of a transformation to a particular input. By contrast *TransformationFunctions* are functions, and therefore sets of *TransformationInstances*. Thanks to the set theory, these concepts can be clearly defined. The goal of this paper is *not* to normalize the terminology, but to make explicit the existence of each concept. In fact, it is not strictly necessary to define the class *TransformationFunctions*. Formally speaking, this class does not convey more information that the invariant associated with this class. Introducing this class just gives a convenient way to refer to a particular role played by a Function in a pattern. We discovered that they are indeed a lot of interesting *MDE patterns*. Simple yet very common patterns are presented in [15]. The next section shows more elaborated patterns based on the theories described above.

#### 4.2 MDE patterns

The reader might be surprised to see that until now, no mention was made to the notion of metamodel. This is because this is indeed a derived concept [14]. A *metamodel* is a model of a language of models. This definition is compatible with those found in the literature (e.g. [35]).

**The meta-step pattern.** The metamodel concept corresponds to a role played by a system in a MDE pattern coined *meta-step*. This pattern is discussed at large in [14]. As shown on the left of Figure 7 is based on a particular configuration of two  $\mu$ -links (*Kernel::RepresentationOf*) and one  $\epsilon$ -link (*SetTheory::ElementOf*).

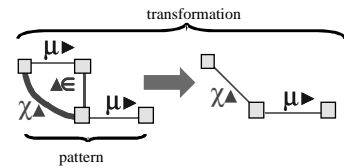
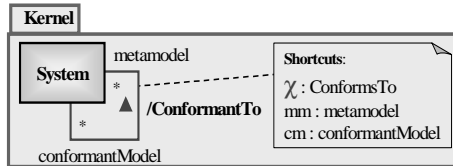


Figure 7 The "meta-step" pattern and transformation rule

A model *ConformsTo* ( $\chi$ ) a metamodel. This derived association is commonly used in MDE [4][35], so it makes sense to defined it in the *Kernel* package. This definition can also be seen as a transformation rule (Figure 7). Most of the time, representing  $\chi$  links in common situations is enough.

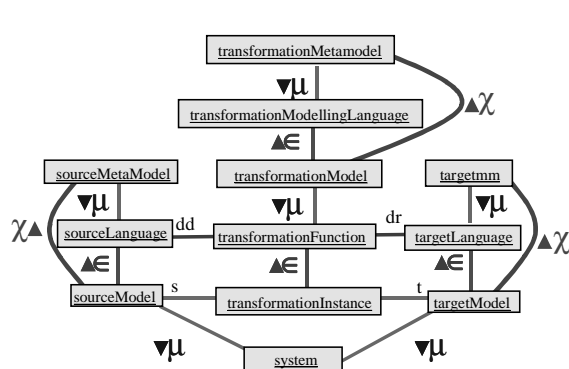


Figure 8 A typical pattern of model transformation

**A pattern of model transformation.** The pattern depicted in Figure 8 corresponds to the notion of model transformation. The names of the instances have been used here to name the role played by each element in the pattern. As suggested by the figure, one can distinguish *transformation instances* (e.g. an XML file transformed into another XML file), *transformation functions* (e.g. a set of somehow related pairs), *transformation models* (a description of a function, also called *transformation program* if it is executable, e.g. a XSLT file), *transformation modelling language* or *transformation programming language* if it can be executed (e.g. XSLT), *transformation metamodel* (e.g. the DTD of XSLT). In fact many other concepts could have been added in this figure using other links, but the approach is the same.

**The interpreter pattern.** A transformation program is a transformation model that can be executed. This leads to the "interpreter pattern" (Figure 9.a). The detailed description of this pattern is quite complex, and recognizing such pattern mentally is not easy. Fortunately one can define some derived associations (in bold in Figure 9.a) including the *LanguageTheory::Interprets* relation ( $\mathbb{1}$ ). Complex graphs can be simplified by applying for instance the rule shown in Figure 9.b. This leads to much more intuitive but still safe views. Figure 9.c illustrates the application of the simplification rule. It shows the various levels of



## 4.5 Discussion

Building a model is not easy when the system is complex. Thousands of years were necessary before to get a correct map of the world. Model Driven Engineering is a moving field and there is a lot to say about it. Building a good model for it is complex research issue, but this could be a rewarding task. First experiments are very encouraging and we were quite surprised of the results that can be get thanks to this very simple approach.

Our approach consists in introducing concepts incrementally, trying at each step to add only truly essential concepts. At each step we take care of developing examples either from computer technology or real world to check the validity of the theory. Over the last years we have accumulated hundreds of metamodels from many different sources. This "metamodel zoo" is used as the basic material to understand how people from different communities use metamodeling techniques, and they are in fact great variations. Our metamodel zoo acts as basic material to test the megamodel. Conversely, the megamodel provides a way to organize the metamodel zoo, but further research is required on this topic.

All this is ongoing work, and the megamodel presented in this paper is subject to many refinement. For instance the notion of conformance is very weak here, but this is on purpose. We are defining various other kinds of more elaborated conformance relations, but this require the introduction of the notions of syntax, semantics and so on.

This very abstract vision of reality might give the false impression that these concepts are disconnected from reality, but this is not the case. For instance the conformance relations are typically checked by actual tools such as *LanguageRecognizers* or *Parsers*. All these concepts can be described in the megamodel. A very important requirement during the design of the megamodel was to give the ability to extend it to suit the needs of particular technological spaces. Another interesting aspect of the approach proposed here is that it allows to compare existing MDE techniques and tools with a common framework. Other concepts such as the notion of platform are really ill-defined and we are not sure that they should be included in the megamodel. May be they are simply wrong concepts but more research is required.

## 5 Conclusion

Though MDE experts advocate for the use of precise modelling, it is striking to see that MDE concepts are usually described only in plain english with sketchy informal pictures. The few UML models we have found are quite suggestive but they don't pass the tests. They often give wrong answers. In fact, it seems that none of these models have been seen as a way to build consistent instance diagrams. This is a pity, because a design which is not tested will invariably be thrown away, sooner or later. We strongly believe that the future of MDE is not necessarily in adding yet another complex technologies on top of already complex technology layers. On the contrary, instead of building (again) new tools from scratch and entering legacy-MDA very soon, it could be wise to learn from the past. The identification of Technological Spaces is fundamental with this respect. Again, MDE is not MDA. MDE is about integrating in a smooth way existing and mature bodies of knowledge. But this could be done only with a shared understanding. And one could not expect researchers from other community to enter the MDE space if MDE concepts are not properly identified. The set theory and language theory are parts of the lingua franca in Computer Science, so using these concepts can greatly help in identifying bridges with existing TSs. In fact, by using the megamodel to model real world situations we discovered that it was much more powerful than we had expected. It really helped us to connect concepts and technologies that were apparently disconnected. The introduction of the set theory had been decisive in this process. Surprisingly we discovered on the run a lot of interesting patterns and we are still discovering new ones. We are still very far from a sound and complete theory for MDE. This paper has only scratched the surface of the topic. Let's hope however that this very little step towards a better understanding of MDE could be helpful to others.

## 6 Acknowledgments

We would like to thank Jacky Estublier, German Vega, Jean Bézivin, and all members of the AS MDA project, as well as attendee of the MDE Dagstuhl Seminar 1401 for the fruitful discussions we had on these topics.

## 7 References

- [1] D.Akehurst, S. Kent, "A Relational Approach to Defining Transformations in a Metamodel", LNCS 2460, UML 2002
- [2] C. Atkinson, T. Kühne, "Model-Driven Development: A Metamodeling Foundation", IEEE Software, September 2003
- [3] J. Bézivin "From Object Composition to Model Transformation with the MDA"
- [4] J. Bézivin, "In Search of a Basic Principle for Model-Driven Engineering", Novatica Journal, Special Issue, March 2004
- [5] J. Bézivin, O. Gerbé, "Towards a Precise Definition of the OMG/MDA Framework", ASE'01, November 2001
- [6] J. Bézivin, E. Breton, G. Dupé, P. Valduriez, "The ATL Transformation-based Model Management Framework", TR03-08, University of Nantes, September 2003
- [7] J. Bézivin *and al*, "First experiments with the ATL model transformation language: Transforming XSLT into XQuery"
- [8] G. Caplat, J.L. Sourrouille, "Considerations about Model Mapping", Wisme 2003
- [9] DSTC, IBM, "MOF Query/View/Transformations", Initial Submission by DSTC IBM, ad/2003-02-03, March 2003
- [10] IEEE. "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems". IEEE Std 1471-2000, [http://www.pithecathropus.com/~awg/public\\_html/](http://www.pithecathropus.com/~awg/public_html/)
- [11] J.M. Favre, "Meta-models and Models Co-Evolution in the 3D Software Space", ELISA 2003, with ICSM, Sept.2003, [www-adele.imag.fr/~jmfavre](http://www-adele.imag.fr/~jmfavre)
- [12] J.M. Favre, "CaCophoNy: Metamodel-Driven Architecture Reconstruction", WCRE 2004, November 2004
- [13] J.M. Favre, "Foundations of Model (driven) (Reverse) Engineering - Episode I: Story of the Fidus Papyrus and the Solarus", post-proceedings of Dagstuhl Seminar on Model Driven Approaches for Language Engineering, May 2004, in [34]
- [14] J.M. Favre, "Foundations of Meta-Pyramids: Languages and Metamodels - Episode II: Story of Thotis the Baboon", post-proceedings of Dagstuhl Seminar on Model Driven Approaches for Language Engineering, May 2004, in [34]
- [15] J.M. Favre, T. NGuyen, "Towards a Megamodel to Model Software Evolution Through Software Transformation", SE-TRA Workshop, Elsevier ENCTS, October 2004
- [16] J.M. Favre, and al., "Reverse Engineering a Large Component-based Software Product", CSMR'2001
- [17] J.M. Favre, "Maintenance et Rétro-ingénierie globale des logiciels", PhD, University of Grenoble, 1996
- [18] D.S. Frankel, "Model Driven Architecture. Applying MDA. to Enterprise Computing", Wiley Publishing,
- [19] M. Griffiths, "Analyse déterministe et compilateurs", PhD, University of Grenoble, 1969
- [20] S.R. Judson, R.B. France, D.L. Carver, "Specifying Model Transformation at the Metamodel Level", Wisme 2003
- [21] A. Kleppe, S. Warmer, W. Bast, "MDA Explained. The Model Driven Architecture: Practice and Promise", Addison-Wesley, April 2003
- [22] P. Klint, R. Lämmel, C. Verhoef, "Towards an engineering discipline for Grammarware", submitted for publication, available at <http://homepages.cwi.nl/~ralf/>
- [23] I. Kurtev, K. van den Berg, "A Synthesis-Based Approach to Transformations in an MDA Software Development Process
- [24] I. Kurtev, J. Bézivin, M. Aksit, "Technological Spaces: an Initial Appraisal", CoopIS, DOA'2002, Industrial track, 2002
- [25] T.Levendovszky *and al*, "Model Reuse with Metamodel-Based Transformations", ICSR-7, LNCS 2319, 2002.
- [26] S.J.Mellor, K.Scott, A.Uhl, D.Weise, "MDA Distilled: Principles of Model-Driven Architecture", Addison Wesley, March 2004
- [27] M. Peltier, "Techniques de transformations de modèles basées sur la méta-modélisation", PhD, University of Nantes, October 2003
- [28] M. Peltier, M. Ziserman, J. Bézivin, "On levels of model transformation", XML Europe, June 2000
- [29] OMG, "MDA Guide Version 1.0.1", omg/2003-06-01, June 2003
- [30] OMG, MDA Web Site, [www.omg.org/mda](http://www.omg.org/mda)
- [31] OMG, "OMG, "Model Driven Architecture (MDA)", ormsc/2001-07-01, July 2001
- [32] OMG, "Meta Object Facility (MOF) Specification" Version 1.4, April 2002
- [33] QVT-Partners, "Revised Submission for MOF 2.0 Query / Views / Transformation RFP", <http://qvtp.org>, August 2003
- [34] Series "From Ancient Egypt to Model Driven Engineering", available at <http://www-adele.imag.fr/mda>
- [35] E. Seidewitz, "What Models Mean", IEEE Software, September 2003
- [36] J.M. Spivey, "The Z notation: A Reference Manual", Prentice Hall, 1992
- [37] E.D. Willink, "UMLX : A graphical transformation language for MDA", MDAFA 2003