

Towards a Megamodel to Model Software Evolution Through Transformations

Jean-Marie Favre^{*} Tam NGuyen

*Laboratoire LSR-IMAG
University of Grenoble, France
<http://www-adele.imag.fr/~jmfavre>*

Abstract

Model Driven Engineering is a promising approach that could lead to the emergence of a new paradigm for software evolution, namely Model Driven Software Evolution. Models, Metamodels and Transformations are the cornerstones of this approach. Combining these concepts leads to very complex structures which revealed to be very difficult to understand especially when different technological spaces are considered such as XMLWare (the technology based on XML), Grammarware and BNF, Modelware and UML, Dataware and SQL, etc. The concepts of model, metamodel and transformation are usually ill-defined in industrial standards like the MDA or XML. This paper provides a conceptual framework, called a megamodel, that aims at modelling large-scale software evolution processes. Such processes are modeled as graphs of systems linked with well-defined set of relations such as *RepresentationOf* (μ), *ConformsTo* (χ) and *IsTransformedIn* (τ).

Key words: model driven engineering, meta-model, software evolution, mda, megamodel

1 Introduction

Model Driven Engineering (MDE) is a promising approach to develop and evolve software. Model, Metamodel and Transformations are the basic concepts of MDE. These concepts are far from new. They were already used in Ancient Egypt, though there were not formalized as such [3]. More recently, these concepts have been studied in many fields of Computer Science, may be under different perspectives and using other terminology. The *Model Driven Architecture* (MDA) standard, launched by the OMG in 2001 [14], had just popularized the vision that models, metamodels and transformations could play a central role in software engineering.

^{*} jmfavre@imag.fr

The OMG grand vision presenting the MDA as the next paradigm in software engineering [14] is a source of strong debate. MDA is poorly defined, too complex, restrictive with the imposed use of MOF standard [14]. More importantly previous, yet similar approaches, such as syntax-driven approaches, have failed to find their path in industry. In fact we believe that there is nothing new in MDA, but that's why this may work this time (Episode I[6]). MDA could be more successful than previous because the software engineering community is more mature, concepts are better understood and tools are already there.

1.1 MDE and Technological Spaces

MDE is not MDA however. In fact, MDA is just a specific incarnation of the Model Driven Engineering approach which is applied to software systems. MDE is by no means restricted to the MDA standard. In fact, the MDE approach might not be restricted to the development and evolution of software systems, though this is on what we concentrate. MDA is a complex set of technologies dominated by the MOF god (Episode II[7]). MDE is on the contrary an open and integrative approach that embraces many other *Technological Spaces* (TSs) in a uniform way [13]. In this paper, the focus is on Software Technological Spaces, that is those used to produce software. The emphasis of MDE is on bridges between technological spaces, and on integration of bodies of knowledge developed by different research communities. Examples of TSs include not only MDA and MOF, but also Grammarware [12] and 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 and Dataware, a "grammar" in Grammarware, or even a "viewpoint" in the software architecture community [9]. In fact the concept of model, metamodels, and transformation are poorly defined in MDA, and this is the same in other standards such as XML. The true essence of these concepts is deeply buried into complex technologies.

1.2 Modelling software evolution

Getting a better understanding of these concepts is important, in particular to model software evolution. The focus of this paper is not on small scale software. These software products can be evolved without problem in an ad-hoc way. We are on the contrary interested in *evolution-in-the-large*, that is the evolution of large-scale industrial software systems. The evolution of these systems often involve various Technological Spaces over time, and various TSs are usually used at the same time. Whatever the technology used, recognizing the concepts of model and metamodels is important in this context [5]. In particular these concepts explain the metamodel/model

co-evolution phenomena. The notion of model itself is also required to understand model/code co-evolution. These problems are not theoretical. They correspond to actual issues with strong implication on software industry development processes.

1.3 Towards a megamodel for MDE

Following the series "From Ancient Egypt to Model Driven Engineering" [3], the goal of this paper is to provide a *megamodel* that is "good enough" to describe MDE. Simply put this "megamodel", which is a model of MDE, should explain what is a model, a metamodel, a transformation, but also what is a transformation model, a model transformation, a model of model, a metamodel of transformation, and any combination of these terms. The megamodel should make it possible to reason about a complex software engineering process without entering into the details of technological space involved. Obviously the results obtained when reasoning on the megamodel must be consistent with those that would be obtained directly with the reality. Technically this megamodel is a metamodel, and therefore a model [3]. But since these terms are defined by the megamodel, calling it a metamodel would be confusing.

The goal of this paper is by no means to invent new concepts. On the contrary we just want to model what already exist. Nothing more. Instead of defining new words this paper relies on existing research on MDE [17][2][11][10][1]. In [17], Seidewitz describes informally, yet thoughtfully, models and meta-models. Bézin identifies two fundamental relations coined *RepresentationOf* and *ConformsTo* [2]. Atkinson and Kuhne study the relationship between MDA and ontologies [1]. Almost all pieces of work carried out to define MDE concepts are either very specific and restricted to a particular TS, or they are expressed in plain english. By contrast the megamodel presented in this paper is expressed in UML with OCL constraints.

This paper presents the current version of the megamodel we have built so far. This megamodel has been carefully designed, and more importantly it has been validated through a large number of examples from different technological spaces. In [3], the study of MDE is taken from an historical perspective and it is shown how artefacts from Ancient Egypt to modern software technologies all conform to the megamodel in a smooth way.

The megamodel is summarized in Figure 10 at the end of this paper. It is made of 5 core associations, namely δ , μ , ϵ , χ and τ . It describes the concepts of model, language, metamodel, and transformation. The reader is invited to refer to the series "From Ancient Egypt to Model Driven Engineering" in which, each association is described in a different episode with plenty of concrete examples. For instance Episode I [6] concentrates on *models* and μ . Episode II [7] concentrates on *languages* and *metamodels*, that is ϵ and χ . Other episodes are under construction.

1.4 Structure of the paper

The remainder of the paper is structured as following. The basics of the megamodel are presented in Section 2. Transformations and *IsTransformedIn* (τ) are then introduced in Section 3. Finally Section 4 shows first results in modelling evolution and Section 5 concludes the paper.

2 Models, Languages, and Metamodels

As shown in the next UML class diagram, the core of MDE megamodel is centered around four relations: δ , μ , ϵ , and χ (Figure 2). Each relation is briefly discussed below in a separate section. For further information about models and μ , refer to Episode I [6]; for languages, metamodels, ϵ and χ , please refer to Episode II [7].

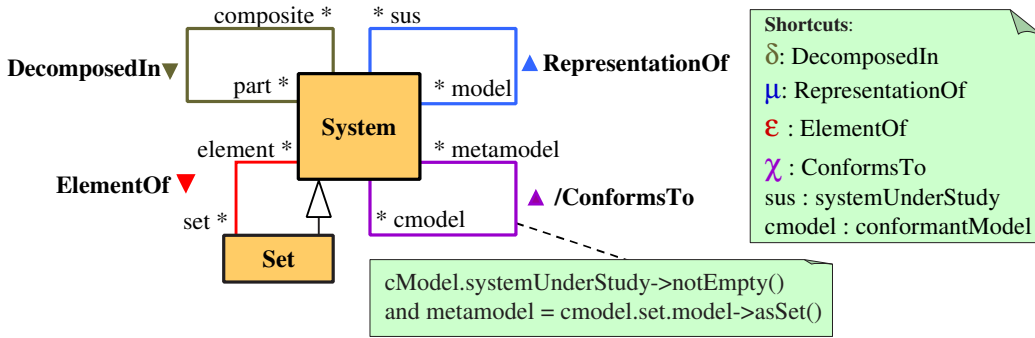


Fig. 1. MegaModel: δ , μ , ϵ , χ

2.1 Systems, Parts and DecomposedIn (δ)

A system is the primary element of discourse when talking about MDE.

This very abstract definition is just here to ensure a broad application of the megamodel. In short everything is a system, yet the use of the term "system" is not really important. Systems can be very simple. For instance the trigonometric value π is a system. The pair (0000011, 0001101) is also a system. Complex system can be decomposed in subsystems or *parts*, leading to the definition of the *DecomposedIn* relation (δ) (Figure 2).

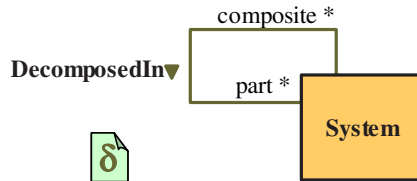


Fig. 2. MegaModel: δ

For instance (0011, 1101) δ 1101 just indicates that the second system is a "part" of the first system. This information could be represented as a δ link

in a UML object diagram, but to save space, we prefer in this paper to use the traditional xRy mathematical notation, which is a shortcut to $(x, y) \in R$. Remember that relations are simply set of pairs in the set theory.

2.2 Models and RepresentationOf (μ)

Instead of providing yet-another definition of what a model is, lets cite existing definitions.

"A model is an abstraction of a physical system, with a certain purpose." (UML Std). "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." [10]. "A model is a set of statements about some system under study (SUS)." [15].

From these definitions we can at least identify three notions: the notion of *model*, the notion of *system under study* (SUS) and a relationship between these notions. This relation is called *RepresentationOf* in [2], so we kept the same terminology. We just use μ as shortcut to avoid wrong connotations and misinterpretations. The μ association is depicted in Figure 3. Episode I [6] is dedicated to the study of this association.

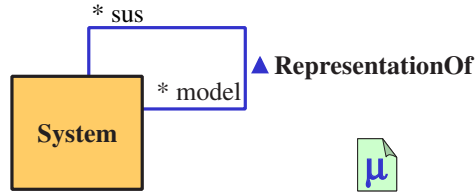
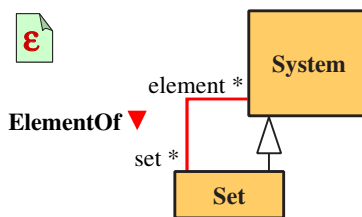


Fig. 3. MegaModel: μ

Lets just summarize here the main properties of this relation. It is key to recognize that the notion of model is relative. This is not an intrinsic property of a system. For instance, $(0011, 1101)$ is a system and it could be just a system. But one can state that this system plays the role of model by arguing that $(0011, 1101) \mu \pi$. One can indeed interpret the two parts of this pair as sequence of bits, and the result as a decimal representation of this the 3.14 value. We can state $(0011, 1101) \mu (3, 14)$ and $(3, 14) \mu \pi$. This example shows that μ links can be combined. The combination of μ and δ links leads to the notion of *interpretation* which is well explained in [17].

2.3 Languages, Sets, and ElementOf (ϵ)

In the language theory a *language* is defined as a *set* of sentences. For instance the set $\{ "h", "ho", "hoo", "hooo", \dots \}$ is the language of words that start with an *h* and continue with *o* letters. Lets call this set *hoL*. The language theory is built on the set theory. In the megamodel, the association *ElementOf* (ϵ) models this concept (see Figure 4).

Fig. 4. MegaModel: ϵ

This association denotes \in in the set theory. Nothing less, nothing more. The relationship between the megamodel, the set theory and the language theory is described in [8]. A language is a set and "*hooo*" ϵ *hoL* holds. As another example, the Java language is the (infinite) set of all java programs. UML is a modelling language. It is the set of all UML models, so *Figure 2* ϵ *UML*.

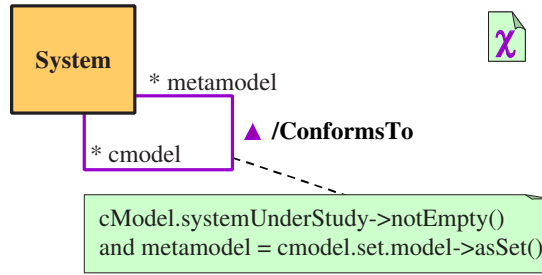
2.4 Metamodels and *ConformsTo* (χ)

Languages are very abstract systems. One need practical means to deal with languages, leading to the (derived) notion of *model of language*. There is nothing new here because languages are just particular systems. For instance, the regular expression $h(o)^*$ is a model of the language *hoL* described above, so $h(o)^* \mu$ *hoL*. A grammar is a model of a language, not a language. An XML DTD, lets say *x.dtd*, is a model of a language, not a language. It is well known that a given language can be modelled by many models (expressed themselves in the same language or using different languages). In the Grammarware technological space [12], this fact is expressed by saying that they are many grammars for a single language (and various grammar languages to express these grammars, such as BNF and YACC). As an example we also have $h(o)^* (o)^* \mu$ *hoL*. Instead of using regular expressions, the *hoL* language can be also modelled by a grammar expressed in BNF or using YACC.

Models of languages ($\mu\chi$) must not be confused with languages of models ($\chi\mu$). *Modelling languages* is the common name used for "languages of models". But in fact, the important concept in MDE is the concept of *models of languages of models* ($\mu\chi\mu$), that is, models of modelling languages. These models are called *metamodels*. This concept leads to the association *ConformsTo* (χ) in the megamodel (Figure 5).

A model must *conform to* its metamodel. These relation has different incarnations depending on the Technological Space considered. For instance in the Grammarware TS, a phrase must conform to the grammar; in the XMLWare TS, an XML document must conform to a DTD; in the Dataware TS, the content of a database must conform to the schema of this database.

As shown in Episode II [7], this association was identified as a foundation of MDE in [2], but our contribution was to show that this is not indeed a basic association as previously thought. *ConformsTo* is on the contrary a

Fig. 5. MegaModel: χ

derived association as shown in Figure 2. In fact the *ConformsTo* takes its root in the set theory since it summarize a particular composition of μ and ϵ links. That is, it merges the notion of set and the notion of models [8]. The notion of metamodel given in this paper is indeed compatible with most of the definition found in the litterature. For instance the following definitions consistently express the fact that a metamodel is a model of a language of models: "A meta-model is a model that defines the language for expressing a model" [16]. "A metamodel is a specification model for a class of SUS where each SUS in the class is itself a valid model expressed in a certain modelling language" [17].

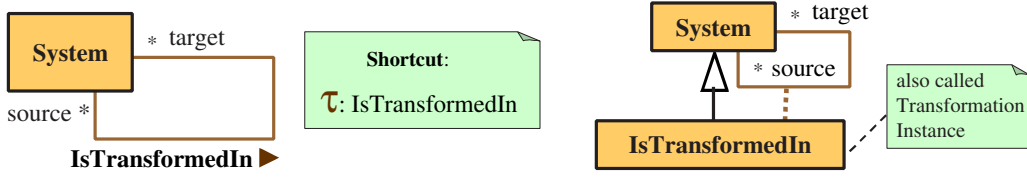
3 Transformations and IsTransformedIn (τ)

Though the initial version of the MDA standard did not put emphasis on transformations, this is really a core concept of MDE, just as model and metamodel. Furthermore, transformations are fundamental to software evolution. In particular Model Driven Software Evolution could also be called Transformation Driven Evolution. Unfortunately though notion of transformation is rather intuitive, there is no consensual terminology and this term is often used to refer to distinct concepts [8]. So let us clarify this notion. It is important to distinguish transformation instances, transformation (functions), transformation models, transformation modelling languages, and transformation metamodels. The goal of this paper is by no means to define a standard terminology. It is just to show that there is a serious issue here and that the megamodel can improve understanding and reasoning about MDE.

3.1 Transformation instances and τ

Following the set theory style, we can say that a system is transformed into another system by modelling this as a simple pair. For that it is enough to introduce the IsTransformedIn (τ) association in the megamodel. This is done on the left of the following figure.

As an example $1 \tau 2$ means the integer 1 IsTransformedIn 2. The pair (1, 2) will be called a *transformation instance*, or transformation application. 1 plays the role of *source* for this transformation while 2 plays the role *target*. If $p1$


 Fig. 6. MegaModel: Transformation instances (τ)

and $p2$ are programs, $p1 \tau p2$ means that the program $p1$ IsTransformedIn the program $p2$. Then $(p1, p2)$ is a program transformation instance. It is important to note that there are no constraints on the transformation instances, and that no reference is made to the complexity of the transformation. $(p1, p2)$ could simply correspond to the addition of a white space, the modification of an algorithm, the renaming of a procedure, or it could be a complete reimplementation of the program.

As the reader might have noticed, the fact that τ is defined on system makes this association very general. It can be combined arbitrarily with the other associations introduced so far. For instance we can say that a model is transformed into another model, that a metamodel is transformed into another metamodel, that a model is transformed into a metamodel, etc. While this last example might seem strange at the first sight, just consider that some tools are able to extract a DTD (e.g. $X.dtd$) from an XML file $x1.xml$. The file $X.dtd$ is the incarnation of a metamodel in the XMLWare Technological Space, while $x1.xml$ is a model. This situation can be modelled by the following facts: $x1.xml \tau X.dtd$ and $x1.xml \chi X.dtd$. As it will be shown by a graph pattern in Figure 8, such transformation is the incarnation of "metamodel inference". Some commercial tools do that.

3.2 Transformation instances as systems

Considering transformation instances as first-class entities, bring even more power to the megamodel. Transformation instances are systems themselves. The associative class on the right of Figure 6 is both a class and an association. The class should be called *TransformationInstance* but unfortunately the association already received the IsTransformedIn name, and only one name can be defined in UML for a given element. Anyway, objects of this (associative) class can be the origin or destination of any link according to UML semantics. This is exactly what is needed. In particular τ links can be combined in many ways leading to complex τ -graphs. For instance the source and/or the target of a transformation instance could be another transformation instance. The power of higher order functions and curriification is well known in Computer Science, and this is exactly what happens in the various technological spaces, though these terms are not necessarily used.

Seeing transformation instances as systems also means that τ can be combined with all other associations from the megamodel (e.g. ϵ , χ , μ). This

is necessary to model the realm of software development. For instance let us assume that we want to analyse the transformation instance from the program $p1$ to the program $p2$, that is the pair $(p1, p2)$. The result of this analysis might be an XML file *diff12.xml* which models the differences between the source and the target. We want something smarter than the output of the unix "diff" tool. We have $diff12.xml \mu (p1, p2)$ and $p1 \tau p2$. So *diff12.xml* is "model of a transformation instance". Continuing with the same example, this XML file might be composed by other transformation instances. For instance the function $f1$ which is a part of $p1$, might have been transformed in the function $f2$ in $p2$. So we have $diff12.xml \delta (f1, f2)$, $p1 \delta f1$, $p2 \delta f2$ and $f1 \tau f2$. Analysing the transformation instance $(p1, p2)$ and producing the summary *diff12.xml* can be useful to understand program evolution. This can be modelled by the following facts $(p1, p2) \tau diff12.xml$ and $diff12.xml \mu (p1, p2)$. From the occurrence of pattern involving τ and μ in the opposite direction, it can be deduced that this is a "reverse engineering transformation", and since it applies to a transformation instance this concrete operation is an example of "transformation instance reverse engineering".

3.3 Transformation (functions)

So far, we have seen that individual systems can be transformed into other individual systems. Software evolution can be seen as a succession of transformation instances, but this is a very weak result which brings no concrete benefits. On the contrary, the challenge of MDE is to automate transformations as far as possible. This could be done only if transformation instances are considered in isolation. They should be described at a higher level of abstraction; not on individual systems, but on set of systems.

A *transformation function*, or *transformation* for short, is a function in the mathematical sense of the term, that is a set of pairs with the constraint that a value map in the domain maps to at most one value in the range [18]. To be more precise *a transformation (function) is a set of transformation instances*. A transformation instance is an ElementOf (ϵ) zero or more transformations. The *domain* of a transformation (function) is the set of systems that can be transformed. The *range* of a transformation (function) is the set of systems that can be obtained via this transformation. This modelling directly comes from the set theory and the Z mathematical language [18]. We just use here the term transformation function or simply transformation instead of function because the term "software evolution through functions" is less popular than "software evolution through transformations". A transformation *is* however a function. Hence the fact that the correct term is "transformation function".

For instance $(1, 2) \in \tau$ means that the integer value 1 is transformed in 2. The set $\{(0, 1), (1, 2), (2, 3), (3, 4), \dots\}$ is a transformation (function) if all its elements are transformation instances. Lets call this transformation *add1* while *mul2* will refer to the transformation $\{(0, 0), (1, 2), (2, 4), (3, 6), \dots\}$. In

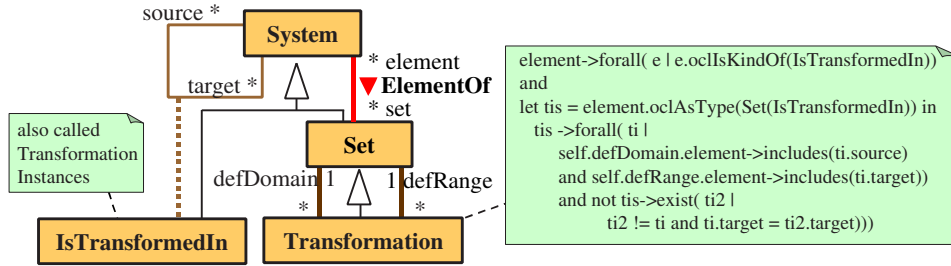


Fig. 7. MegaModel: Transformations Functions vs. Transformation Instances

this example we can see that a transformation instance can be element of various transformation function since $(1, 2) \in add1$ and $(1, 2) \in mul2$. Classifying transformation instances in terms of existing transformation is a known problem in evolution analysis. For instance the goal might be to recognize refactorings from transformation instance. In this case one tries to figure out by observing the difference between two successive versions of a program, first what has changed, and then in which set this change (this transformation instance) can be classified. A refactoring such as "rename method" is a transformation function while the particular application of a refactoring is a transformation instance.

3.4 Transformation models

Transformation functions as defined above are abstract systems and there are therefore not operational. What is required in MDE, is a concrete means to describe transformations. This naturally leads to transformation models. *A transformation model is a model of a transformation (function).*

For instance lets call *dblC* the C program `int f(int x) {return x+x; }`, *dblP* the Pascal program `function f(x : integer) : integer begin return x * 2 end` and *dblC2* the C program `int f(int x){return x<<1;}`. It should be clear that $dblC \mu mul2$, $dblP \mu mul2$ and $dblC2 \mu mul2$. As suggested by this example, they are plenty of ways and languages to write transformation models. This example is simple but it is not representative of software evolution through transformation. A compiler, a refactoring tool, or the YACC tool are better examples of transformation models for software evolution because they transform models.

Transformation models must not be confused with transformation functions or with transformation instances. In the XMLWare Technological Space, an XSLT stylesheet is an example of a transformation model. It models a transformation (function) defined on XML files. This transformation could be expressed in any other language, for instance XQuery. The application of the stylesheet on a particular XML file leads to a transformation instance. The application of a compiler on a particular program, or the application of YACC on a particular grammar are examples of transformation instances.

Every modification can be seen as a transformation instance. In fact the

huge majority of transformation instances applied during software evolution are ad-hoc. That is they are not elements of an existing transformation functions. Software engineers just change programs, without wondering if this is an instance of a transformation. Refactorings are examples of transformations, that can be modeled and therefore automated, but these are isolated examples. In fact currently software evolution is driven by ad-hoc transformation instances while the goal of Model Driven Engineering is to drive the process through a set of reusable transformation functions.

4 Applying the Megamodel to Software Evolution

We believe that all software evolution processes can be modelled as a graph using the megamodel presented above. Example of graphs are provided in [3] in the form of UML object diagrams, but Episode I and II present only a static vision. Before considering transformation and evolution, lets first consider such a static vision. Simply put each version of a large scale software is a complex system, so each version can be modelled as a graph built on the following elements:

- δ links, for instance to model the fact the software under study is made of packages, which are made of source files, which are made of functions, etc.
- μ links, for instance to model that a Z model is a specification of an Ada program.
- ϵ links, for instance to model that an XML model pertains to a Domain Specific Language (DSL). Languages should be considered as integral parts of software, especially in the long term since they will invariably evolve [5].
- χ links, for instance to model the fact that an XML model is conform to a DTD which model the DSL mentioned above. Or to model that the conformity of an Ada program is checked by the Ada compiler, which is a metaware tool [5].
- τ links, for instance to model the fact that a binary file has been produced from an Ada source file, itself produced from a Z specification model.

So each version can be represented by a graph. The evolution of the software can be modelled by a composition of these graphs using τ links. At the end, we just obtain a bigger graph with all kind of links. The megamodel is by no means restricted to model evolution or software evolution. Since everything is a system, everything could be transformed. That is, every system can be the source or the target of a τ link. This is required because in very large software companies everything evolve soon or later[5]. In fact evolution can be modelled by combination of τ links and other kind of links determining the kind of evolution. If we consider *evolution-in-the large* [5], languages evolve ($\epsilon\tau$). Metamodels evolve ($\mu\epsilon\mu\tau$). Transformation models evolve ($\tau\epsilon\mu\tau$). And so on.

Moreover when two systems connected by a link evolve, this leads to *co-evolution* issues. This is because consistency must be maintained between the ends of the link. Examples of co-evolution phenomena, include for instance *model/code co-evolution* ($\tau\mu\tau$). *Metamodel/model co-evolution* [5] is another example ($\tau\mu\tau$).

The sequence of greek letters used here above are ambiguous, in particular because there is no formal rule for the ordering of letters. This is because the concepts described above corresponds to graph patterns, not simply sequences. We have identified a lot of interesting patterns that corresponds to known concepts. Some examples are provided in the next figure.

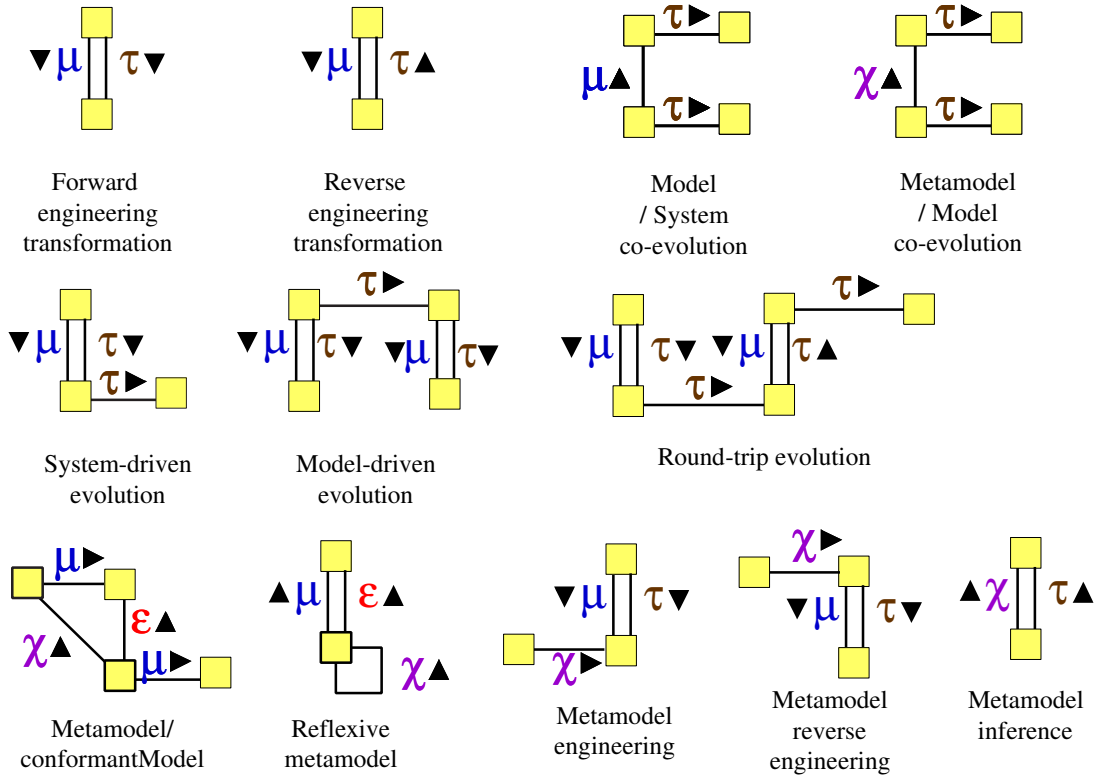


Fig. 8. MegaModel: Examples of interesting mega-patterns (τ)

5 Conclusion

In this paper we introduced a megamodel to describe MDE concepts and their relationships. This megamodel is summarized in Figure 9. The view presented here corresponds has been simplified for the purpose of this paper. A more complete view making explicit the relationships between the megamodel, the set theory and the language theory can be found in [8].

In fact, by using the megamodel we discovered that it was much more powerful than expected. It really helped us to connect concepts and technologies that were apparently disconnected. Surprisingly we discovered that a lot of

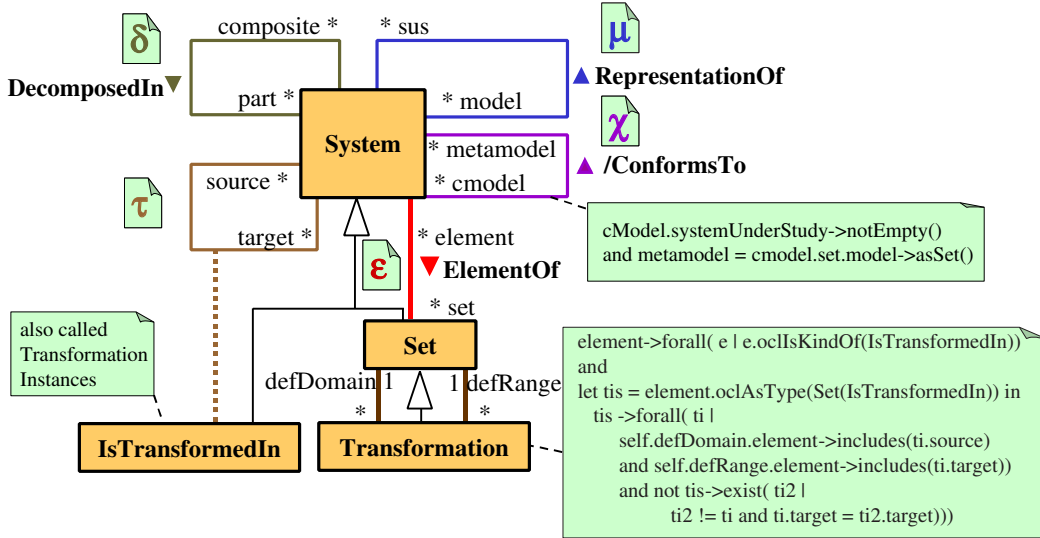


Fig. 9. MegaModel Overview

known issues could be model as graph patterns. And we are still discovering new ones. With respect to the form of the megamodel, the language used here is UML, but we are building other incarnations of the megamodel. At the time of writing this paper we are working on a version in Prolog, in Z [18], and in hieroglyphics [3][4]. The megamodel is expressed using different languages. Interestingly the Prolog megamodel is executable, so it can automatically recognize the patterns mentioned above and derive new facts from a given model expressed in the megamodel [8]. This program results from the transformation of the UML megamodel with the transformation of OCL constraints to prolog rules. In other words that megamodel transformation, so we applied MDE to its own megamodel... In practice the problem is not really with the semantics of the megamodel, but with its interpretation [3]. That is formalizing the metamodel is not really an the important issue. MDE is not per se a formal system and the problem is much more about how to represent MDE real world, that about the language used to describe the resulting model. In other words the issue is how to extract a model from a software evolution process. To ease this task we are in the process of defining systematic mapping between the megamodel and its concrete incarnations in each technological spaces. Experiments we have done so far are very promising. This should not be surprising, because we are building the metamodels to reflect our practical knowledge about existing technological spaces.

We do not claim however that the megamodel presented here is complete or perfect. Like any other model, the megamodel certainly presents of lot of room for improvements. For instance, to model co-evolution phenomena it is necessary to include the notion of distance to express to what extent a model conforms to a metamodel (metamodel/model co-evolution), or a model is representation of code (model/code co-evolution). Adding the notion of metrics is also further work.

Finally the reader might still wonder what this research is all about. This is no code there. And nothing new either. Just consider things from a different perspective. The MDA standards is made of more than 2500 pages. It is full of complex technologies and they are plenty of commercial tools that claim to be MDA compliant. This standard is more than three years old. Major actors in software industry, such as Microsoft and IBM, announced MDE as being integral part of their strategy. Despite of that most people in academy still wonder what could be a metamodel transformation, a metamodel-driven evolution process or even a model-driven evolution process through transformation. While the term meta has been adopted by software industry leaders, it is still considered as suspicious by many. The goal of this paper is just to improve the understanding of MDE concepts and to make them more accessible. We believe that this is a strategical issue for software evolution. After about 50 years of empirical software evolution, it makes no doubt that software can be evolved in an ad-hoc way. What is needed, is a new paradigm for software evolution. MDE might be a candidate for that.

6 Acknowledgments

We would like to thanks the anonymous reviewers for their comments. We also would like to thanks Jean Bzivin, Jacky Estublier, German Vega, and all members of the AS MDA project, as well as attendee of the Dagstuhl seminar 1401 on MDA for the fruitful discussions we had on this topic.

References

- [1] C. Atkinson and T. Kühne. Model-driven development: A metamodeling foundation. *IEEE Software*, September 2003.
- [2] J. Bézivin. In search of a basic principle for model-driven engineering. *Novatica Journal, Special Issue*, March-April 2004.
- [3] Series From Ancient Egypt to Model Driven Engineering. www-adele.imag.fr/mda.
- [4] J.F. Champollion. *Grammaire Egyptienne*. 1836. in French.
- [5] J.M. Favre. Meta-models and models co-evolution in the 3d software space. In *Workshop on the Evolution of Large scale Industrial Software Applications, ELISA, Joint workshop with ICSM*, sept. 2003. available at www-adele.imag.fr/~jmfavre.
- [6] J.M. Favre. Foundations of Model (Driven) (Reverse) Engineering: Models - Episode I: Stories of the Fidus Papyrus and of the Solarus. In *Post-proceedings of Dagstuhl Seminar 04101*, 2004. Episode of [3].

- [7] J.M. Favre. Foundations of the Meta-pyramids: Languages and Metamodels - Episode II: Story of Thotus the Baboon. In *Post-proceedings of Dagstuhl Seminar 04101*, 2004. Episode of [3].
- [8] J.M. Favre. Towards a basic theory for modelling model driven engineering. In *Proceedings of WISME*, November 2004. available at www-adele.imag.fr/~jmfavre.
- [9] IEEE. Ieee recommended practice for architectural description of software-intensive systems, iee std 1471, 2000.
- [10] O Gerbé J. Bézivin. Towards a precise definition of the omg/mda framework. In *Proceedings of ASE01*, November 2001.
- [11] A. Kleppe, S. Warmer, and W. Bast. *MDA Explained. The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.
- [12] P. Klint, R. Lämmel, and C. Verhof. Towards an engineering discipline for grammarware. Technical report, submitted for publication, CWI. available at homepages.cwi.nl/~ralf/.
- [13] I. Kurtev, J. Bézivin, and M. Aksit. Technological spaces: an initial appraisal. In *CoopIS, DOA'2002 Federated Conferences, Industrial track*, Irvine, 2002.
- [14] OMG. Mda web site. www.omg.org/mda.
- [15] OMG. Omg, model driven architecture (mda). ormsc/2001-07-01, CWI, july 2001. available at www.omg.org/mda.
- [16] OMG. Meta object facility (mof) specification, version 1.4. Technical report, april 2002. available at www.omg.org/mda.
- [17] E. Seidewitz. What models mean. *IEEE Software*, September 2003.
- [18] J.M. Spivey. *The Z notation: A Reference Manual*. Prentice Hall, 1992.