Automatic Generation of Models from their Metamodels using Multilayer Perceptron Network

Anonymized Authors
University, Department
Location
email or 0000-0000-0000

***Abstract*—Model driven-Engineering (MDE) is one of the most recent disciplines of software development that enables us to use models and their transformations during the software life-cycle, from requirements to implementation and maintenance instead of using the classic programming languages. In this context, the generation of models is generally done manually to ensure conformity to their metamodels. There are several works [1]–[3] proposed to automate this process, but no one of them can ensure complete automation without starting from a set of models already defined by the user or with a good verification of all conformity constraints. In this paper, our objective is not only (i) how to generate the models in an automatic way and verify all conformity constraints but also (ii) how to use one of the most machine learning techniques to solve modeling problems in MDE context.**

***Index Terms*—Space Modeling, Models, Metamodels, automatic generation of models, Model driven-Engineering, OCL language.**

I. Introduction

During the 1990s, the software engineering has faced an exponential increase in the complexity of computer systems due to the difficulty of implementing development strategies for the lack of appropriate tools as well as the emergence of software product line problem. In return, often, the produced software did not meet expectations. Thus, it can be said that the specifications do not always properly reflect the requirements. As in other sciences, generally, we focus much more on modeling in order to master this complexity and even to produce the software than to validate it. Likewise, Model-Driven Engineering (MDE) is one of the best-proposed solutions to solve the complexity of computer systems and to minimize simultaneously development time and costs of new software especially those that are classified in the same family.

MDE [4] is therefore an approach to software engineering with which the model is considered as a first presentation of the system to be modeled, and which aims to develop, maintain and evolve the software by integring transformations of this model. In a broad sense, the MDE paradigm proposes to merge all aspects of the life-cycle process using the concepts of model and transformation. In this context, the use of models needs to describe its metamodel by manipulating one of the metamodeling languages such as Ecore [5] and KM3 [6]. Then, a metamodel is an abstract model description where its instantiations present models with which the transformation can be executed to obtain at the last step an executable code.

To facilitate model-driven development (MDD), the Object Management Group (OMG) has proposed an approach called Model Driven Architecture (MDA) [7] in November 2000. This approach gives new software development strategies with which the specifications considered more important than the implementation by concentrating on modeling steps.

Moreover, MDA is considered as the best solution that has different exploited advantages such as the automatic bridge construction from one environment to another, the possibility to regenerate the source code from a platform-independent model with changing the infrastructure over time thus it facilitates developing and maintaining the most important step of an application's life-cycle and the abundance of necessary MDA tools leads to make the software development process easy.

Although the transformation during the development was semi-automated using a set of tools and languages, the modeling space creation was not completely automated until now. Therefore, this paper gives a new solution that complements previous works ([8], [2], [3], [9], [1] and [10]) by (i) building the models from their metamodels automatically and (ii) exploiting the use of machine learning technique (iii) without focusing on a set of an initial models. Our Objctif is not only the automation of space modeling creation but also to simplify and facilitate the software development process by ensuring the transformation test and the programming phases by examples (models).

The rest of this paper is structured as follows: Background, Motivation, and Problem statement are presented in section II. Detailed description and evaluation of the proposed approach are defined in section III and IV. In section V we presents related work. Finally, section VI concludes this paper.

II. Background, Motivation & Problem statement

This section gives a general description of MDA technology and the modeling requirements.

A. Definitions & MDA Basic concepts

The principle key of MDA consists to rely on the UML standard in order to describe separately the models about the different development phases of an application in life-cycle. In the following we provide some definitions and informations about MDA technology which will be used in the rest of this paper. More specifically, MDA has three types of models [11]

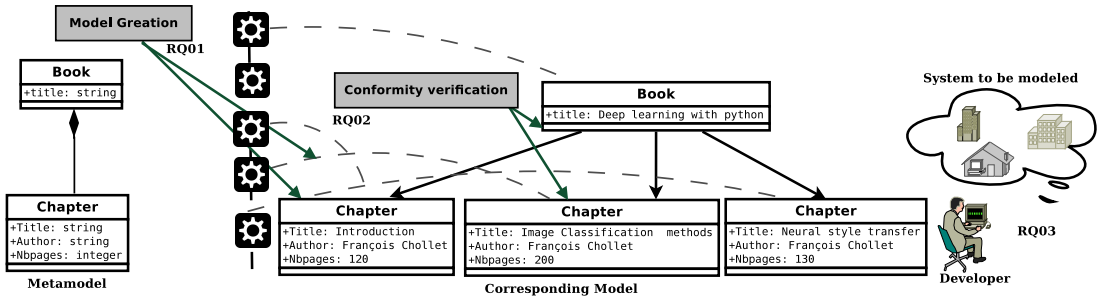

Fig. 1. Modeling problems.

that are used in software development. In The following, we define each one and the relationship between them (see Figure 2):

Computation Independent Model (CIM): A CIM defines exactly what the system is supposed to do but masks all technical details related to the implementation of the system.

Platform Independent Model (PIM): A PIM enables to specify system views independently from the platform where it can be related to several platforms at the PSM level.

Platform Specific Model (PSM): A PSM refines the PIM with the technical details needed to specify how the system can use a specific platform. A PIM can be translated to one or more PSMs, which are for a specific implementation technology.

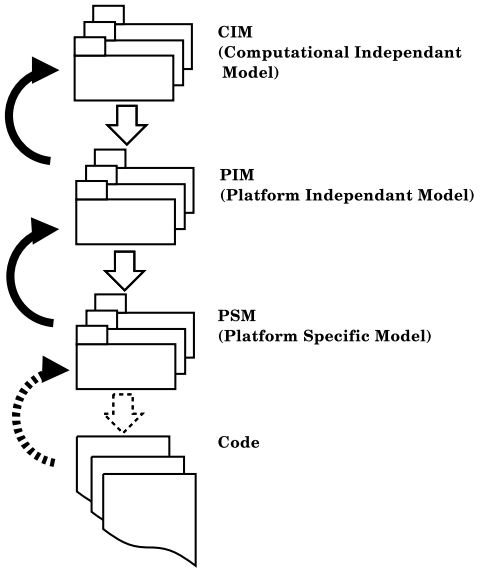

Fig. 2. An overview of MDA approach (taken from [11]).

Figure 2 shows an overview of MDA approach. In this context, building a new application begins with the development of one or more requirements models (CIM). Then, it continues with the development of analysis and abstract-design models of application (PIM). These must be partially generated from the CIMs that traceability links are established. We note that PIM models are perennial models, which do not contain any information about the execution platforms. To realize the application, it is necessary to build specific models of the execution platforms (PSMs). These models are obtained by a transformation of PIM(s) by adding the technical information about the execution platforms. Their main function is to simplify code generation that is considered as a translation into a textual formalism [11].

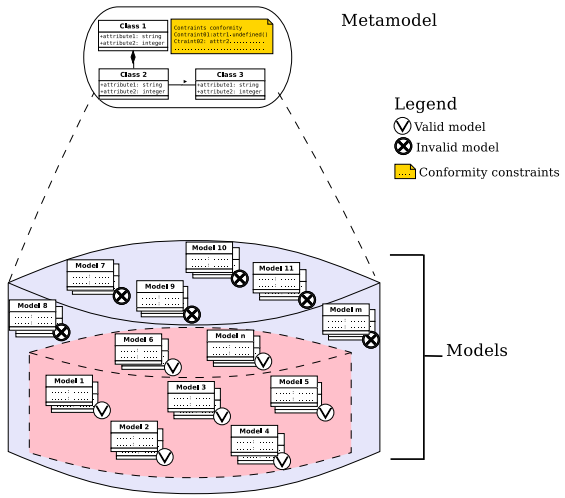

Fig. 3. Modeling space and its limit (Adapted from [8]).

Finding 1: Through the separation between the dependent models on the platform and the abstract independent models of the application, MDA approach offers the following objectives: portability, reusability and interoperability.

Figure 3 borrowed from [8] gives a general description about the space modeling and its limit. This space contains two model types, the first one describes the well-formed models and the second defines the ill-formed models (see figure 3). When we generate the models from their metamodels we receive one of these two types, if the model is well instantiated and it checks the defined constraints so we obtain the well-formed model otherwise we receive a ill-formed model. These results depend on the instantiation process, which is done manually or automatically. In the following, we present Modeling requirements to provide models automatically.

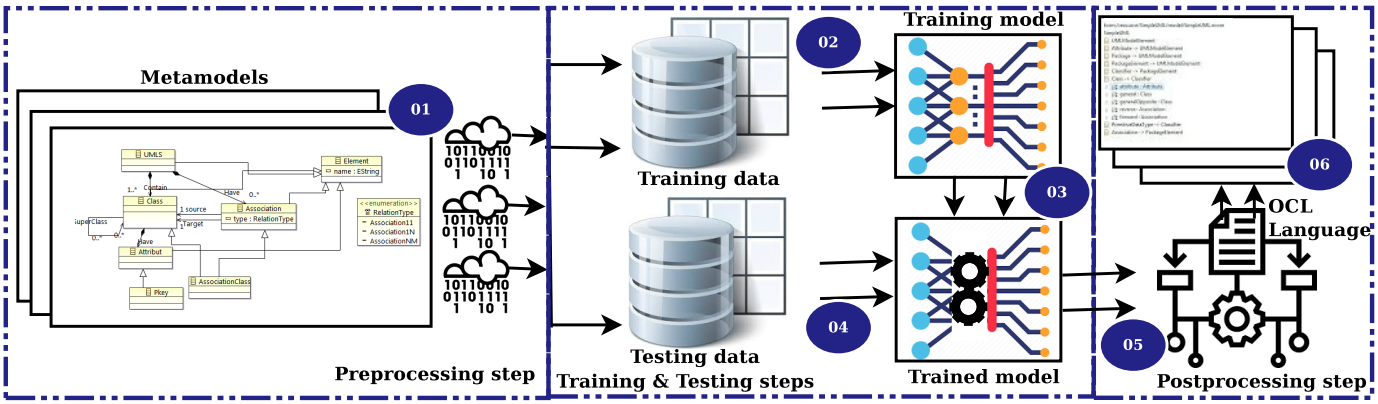

Fig. 4. Modeling requirements.

B. Modeling requirements

In this sub-section, we describe the modeling requirements as questions to provide simple way of comprehension for readers. Also, we define these requirements schematically in the figure 1.

RQ1: How can we generate the models from their metamodels automatically or semi-automatically?. In MDE field, the generation of models from their metamodels and the verification of their conformity are done manually. This process demands the efforts and the times even is done by the system developers so as not to make the mistakes. Thus, the automation of this process is considered as the challenge of several researchs such as [8]. The principal objectif of this paper is to solve this problem in diffrent way compared with the pervious work [1]–[3].

RQ2: Can we verify the conformity of models automatically or semi-automatically?. In general, the conformity verification of models to their metamodels is done semi-automatically by defining the constraints in OCL language. At this time, no proposed approach automates the model creation and the conformity verification in parallel to facilitate the development in MDE context.

RQ3: Can we find the a technique to simplify the system modeling process?. The recent software development such as MDA revolves around the use of models to separate the preoccupations between application logic and implementation techniques in order to increase the productivity. Therefore, the system developers must have a solid experience on modeling by manipulating models.

> **Finding 2:** At this moment, we distinguish that we have modeling & metamodeling processes. In our case **RQ1**, **RQ2** and **RQ3** are asked to describe principal requirements in the modeling step. This latter is considered as the heart of the development process.

This question set describes the crucial challenges in the modeling process with which the software development productivity is incresed. In the following section, we discuss on our proposed approach to answer the previous questions in order to solve the principal problems of modeling step in MDE context.

III. Model's automatic generation

Nowadays, the use of machine learning classified at the heart of many areas such as image, speech and natural language processing since it replaces the classic programming. Therefore, we are interested to exploit its advantages in the modeling step. Figure 4 describes our proposed approach to generate models from their metamodels foncused on the machine learning technique that called "Multilayer Perceptron Network"(MLP). This approach is divided into three main steps, the first one describes data preprocessing, the second one occupies the definition of our MLP architecture and the third step provides the detailed description of Model generation from MLP output. The following sub-sections detailed these steps.

A. Preprocessing Step

In this step, we are based on the extraction of all metamodel informations whose will be used in the next steps. Generally, each metamodel is defined by a set of classes and their relationships. In this case, we can say that a metamodel is a set of segments and each segment contains one relationship with its target class or a root class.

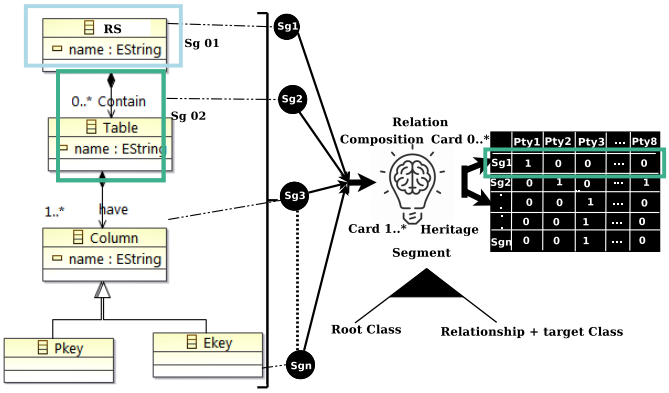

Fig. 5. Extracted Informations & Segment Structure.

Figure 5 shows the basic informations about each segment and digital extracted information process. In this case, We consider a segment as one or two metamodel elements and each element can be either root class or relationship with its target class (see figure 5 exactly right side). Once all segments are extracted, we test a set of properties for each segment in order to compute digital informations about the used metamodel. These properties are descriped in the table I. For instance, the root class in figure 5 is defined as the segment number (1) and its digital informations is shown in the first column of the table defined in the same figure. These binary informations reflect the properties of each relationship or root class for each segment. In our case, figure 5 shows the relational database that contains a set of tables each table has one or more column and each column can be defined as primary or foreign key. The segment number "**1**" is defined by the root class ***RS***. The segment number "**2**" is presented by the class ***Table*** and the relationship ***Contain***. The same principle is applied to other elements in order to extract the rest of segments. Once this process is done, the comparison of extracted segments with a set of propertises can be calculated by insering one if the segment verify the property and zero otherwise.

TABLE I
DESCRIPTION OF PROPERTIES.

Properties	***Property intituled***
Pr"1"	The root-class property.
Pr"2"	The heritage property.
Pr"3"	The composition property.
Pr"4"	The association property.
Pr"5"	The class-association property.
Pr"6"	The 1..1 cardinality.
Pr"7"	The 0..n cardinality.
Pr"8"	The 1..n cardinality.

TABLE II
EXTRACTED INFORMATIONS FROM RS METAMODEL.

	Pr"1"	***Pr"2"***	***Pr"3"***	***Pr"4"***	***Pr"5"***	***Pr"6"***	***Pr"7"***	***Pr"8"***
Seg1	1	0	0	0	0	0	0	0
Seg2	0	0	1	0	0	0	1	0
Seg3	0	0	1	0	0	0	0	1
Seg4	0	1	0	0	0	0	0	0
Seg5	0	1	0	0	0	0	0	0

Following table II gives these extracted informations from RS (Relational Schema) Metamodel as an example of data informtion that will be used as input elements of training & testing steps.

B. Training & Testing Steps

The first formal neuron appeared in 1943 proposed by Mac Culloch and Pitts by introducing it as a calculation unit. This formal neuron calculates a weighted sum of its inputs $x_1, \ldots, x_n$ and returns 1 if the sum is greater than a certain threshold θ and 0 otherwise. Mathematically, this returns to writing the following equation 1:

$$z = f(\sum_{i=1}^{n}(w_i \cdot x_i) + b), \tag{1}$$

where f is the transfer function, w_i is the weight where it is often referred to as *preactivation*. Generally the bias term b will be replaced by an equivalent input term $x_0 = 1$ weighted by $w_0 = b$. The result of this previous equation is then passed through a step function of the form

$$y = \begin{cases} 1, & \text{if } z \geq 0, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

which specifies the binary output of the Perceptron. In the following, we use this result to classify whether the entry belongs to a specific class or not. Adding that the model can be extended to handle multiple classes, only by adding more dimensions to y and assigning each of them to a different class. In the learning step, we are based on the modifications of the network parameters such as the weights and the bias in order to have good output values. This change is based on the following rule:

$$w_i^{\text{new}} = w_i^{\text{old}} - \eta \cdot (\hat{y} - y) \cdot x_i, \tag{3}$$

where $\hat{y}$ is the Perceptron output, y is the target (i.e., desired) value, x_i and w_i^{old} are the i-the input and weight at the previous iteration and η is a scaling factor that allows to adjust the magnitude by which the weights are modified. Once the network parameters are well adjusted the testing step will be executed using other data information in order to test the capacity of the chosen system or network. In this step, we compute diretly the ouput $\hat{y}$ without recalculate the network parameters such as the weights.

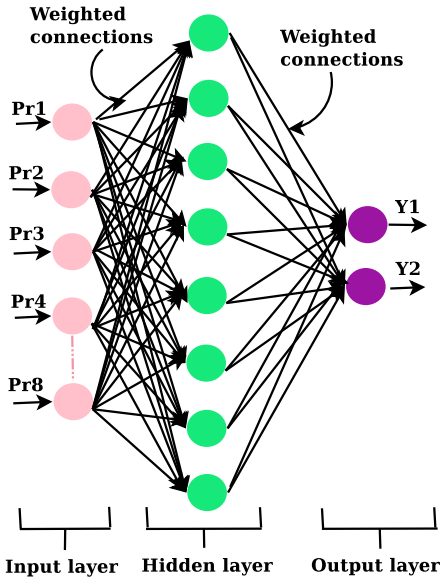

Fig. 6. Our MLP Architecture.

Figure 6 describes our MLP architecture where the input data x_i [1] are defined by binary values of each segment and the ouput data $\hat{y}$ gives the binary informations about each input segment if will be instancieted or not.

C. Posteprocessing Step

This step occupies the generation of Models From MLP Output using the following algorithm.

Algorithm 1 From MLP outputs To models

Require: MLP Outputs;
Ensure: Creation of Models;
1: Compute number of MLP outputs $N = size(Classes)$;
2: Create set $C1 = outputs\ of\ Class1$;
3: Create set $C2 = outputs\ of\ Class2$;
4: Create set $Cn = outputs\ of\ Classn$;
5: **for** $i = 1\ \ to\ \ N$ **do**
6: **while** $Ci <> \emptyset$ **do**
7: $C = dequeue(Ci)$;
8: Let $X = \ Corresponding\ fragment(C)$;
9: Find X position in its metamodel & intanciate it;
10: **if** AttrCont(X) $<> \emptyset$ **then**
11: Execute Verify_contraint_attibut(AttrCont(X));
12: **end if**
13: **end while**
14: **end for**
15: Redefine C1 and C2;
16: **for** $i = 1\ \ to\ \ 2$ **do**
17: **while** $Ci <> \emptyset$ **do**
18: $C = dequeue(Ci)$;
19: Let $X = \ Corresponding\ fragment(C)$;
20: Find X position in its metamodel & intanciate it;
21: $Dequeue(brother(C))$;
22: **if** AttrCont(X) $<> \emptyset$ **then**
23: Execute Verify_contraint_attibut(AttrCont(X));
24: **end if**
25: **end while**
26: **end for**

Input Data: Input data of algorithm 1 is our neural network outputs that describe if the segment will be instantiated or not. In our case, we have three classes (in algorithm 1 N egals three) the first one regroups the instantiated segments, the second class shows the segments that may be instantiated and the third class presents the segments that are not instantiated. ***Output Data:*** Output data of algorithm 1 is the models. From this Algorithm 1 we create the models and we verify some contraints with OCL langage [12] in order to have well-formed models. This creation process is also based on the use of ***instantiation data*** that aims to increase the rate of well-formed models creation. For instance, if we have as input data the metamodel of family in this case, we instantiate the name of person by using the stocked information from ***instantiation data***. the conformity verification , the instantiation process and the model comparison are done by other Algorithms that are not defined in this paper due to lack of space.

[1]Where i is varied from 1 to 8 according to the used properties.

Finding 3: Through the separation between different problems such as the conformity verification , the instantiation process and the model comparison the model creation is well done and in short-time.

IV. Experimentation & Evaluation

This section illustrates the results and the discussion about our proposed approach to present its initial evaluation.

A. Results & Interprtation

In this section, we discuss on the basic informations about each training and testing data examples that are mentionned in the following table III. These informations relate to the specification of the class and the segment numbers with which the example complexity is given.

TABLE III
Basic Informations about Training & Testing Data

	Traning Data	
$Metamodel_{name}$	$NO_{Classes}$	$NO_{Segments}$
Book	02	02
Publication	01	01
Person	03	02
Petrinet	03	05
Satemachine	13	24
	Testing Data	
Family	02	05
UML-CD	07	14
RS	05	05

Figure 7 illustrates the mean squared error of training, validation and testing data. From this figure, we summarize that the training error reduces after four epochs and take the best value in twenty one epochs, but might stagnated on the training data set after eight epochs. Generally, our MLP network takes the best values after twenty one epochs.

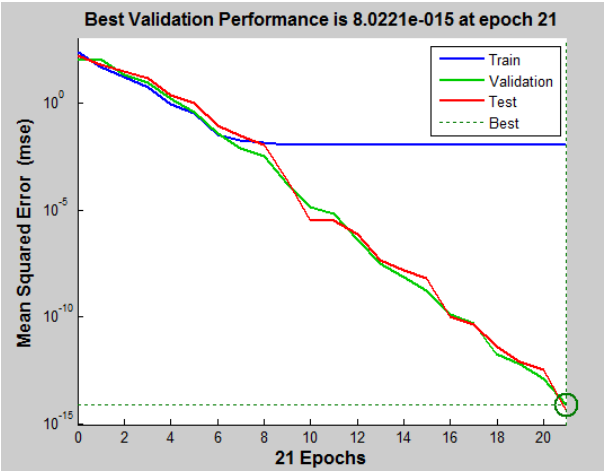

Fig. 7. Performance of MLP Network.

B. Thread to Validity

Our proposed approach provides good results in order to produce models from their metamodels without using an initiative set of models which generally requires a good knowledge about the system to be modeled and a lot of times. Compared with other solutions such as [8], [2], [3] and [10] our approach provides new idea that automates the modeling steps by using the machine learning technique " multilayer perceptron neural networks" and answering on ***RQ1***, ***RQ2*** and ***RQ3*** mentionned in the section II. For example, the ***RQ1*** and ***RQ3*** problems are solved by using "MLP" technique and our proposed algorithm 1 with which the models are generated automatically without using an initial set of models and also by checking some levels of conformity without intervening the modeling designers. In this case, we answer also on the ***RQ2***.

V. Related Work

Several approaches have been proposed for automatic model generation problem. In the following, we describe some of them dependent on the used mechanism.

1) Using optimization techniques: The most important approaches for modeling space are based on metaheuristic techniques. For instance, this paper [8] presents a metaheuristic approach for automatic generation of more precise models using Simulated Annealing (SA) in conformity with a set of criteria defined in [2]. Another work [3] processes this problem using another heuristic method called genetic algorithm (GA) to compute from an initial model set, other well-structured models in order to increase modeling space. Batot [10] has proposed also another approach to solve modeling space problem that allows using a non-dominated genetic algorithm (NSGA-II) from metamodel and a set of models.

2)Using Graph grammar methodology: Karsten et al. [9] proposes to use graph grammar for creating model instances of metamodels in automatic way without using an initial set of models. This approach requires to have a good-basic aspects about Graph grammar methodology that requires a lot of time and effort.

3) Using formal methods: In this context, the use of formal methods [13] is not supported due to their difficulties nevertheless there are only some proposed work [14]. One best-known of them is [14] that focused on instance generation by using Alloy [15]. This instance generation is based on the translation of a class diagram to Alloy representation, after SAT solvers are used to create the instances by enumerating them.

4) Other related work: More generally, there are also other generic approaches that aim to specify or verify modeling space automatically either to increase the size of test Data for Model Transformations with well-structured models or even to be used as input data for model transformation by-example. This paper [2] describes one of the most genetic approaches by proposing a set of rules used to evaluate the correctness of input models.

VI. conclusion & Future Work

In MDE context, one of the most problems faced by developer's software is how to automate or facilitate model generation process in the development of software system?. In the past few years, several studies are proposed to answer this question but in general, they generate models in random way or without verifying all conformity constraints. In this paper we proposed an approach to automate the generation of model focused only on its metamodel and using "Multilayer Perceptron Network" and also a simple algorithm to create good and verified models in order to reduce the costs and the time of software development.

As perspective work, we propose to take into account the verification of OCL complex constraints by using other techniques. Also, we propose to apply other maching learning methods to have a comparative study that aims to select from a set of propoerties the appropriate method for modeling step in MDE context.

References

[1] E. Batot and H. Sahraoui, "A generic framework for model-set selection for the unification of testing and learning mde tasks," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, 2016, pp. 374–384.

[2] F. Fleurey, B. Baudry, P.-A. Muller, and Y. Le Traon, "Qualifying input test data for model transformations," *Software & Systems Modeling*, vol. 8, no. 2, pp. 185–203, 2009.

[3] A. ben Fadhel, M. Kessentini, P. Langer, and M. Wimmer, "Search-based detection of high-level model changes," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 212–221.

[4] D. C. Schmidt, "Model-driven engineering," *COMPUTER-IEEE COMPUTER SOCIETY-*, vol. 39, no. 2, p. 25, 2006.

[5] F. Budinsky, S. A. Brodsky, and E. Merks, *Eclipse Modeling Framework*. Pearson Education, 2003.

[6] F. Jouault, J. Bzivin, and A. Team, "KM3: a dsl for metamodel specification," in *In proc. of 8th FMOODS, LNCS 4037*, 2006, pp. 171–185.

[7] R. Soley *et al.*, "Model driven architecture," *OMG white paper*, vol. 308, no. 308, p. 5, 2000.

[8] J. J. C. Gómez, B. Baudry, and H. Sahraoui, "Searching the boundaries of a modeling space to test metamodels," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE, 2012, pp. 131–140.

[9] K. Ehrig, J. M. Küster, and G. Taentzer, "Generating instance models from meta-models," *Software & Systems Modeling*, vol. 8, no. 4, pp. 479–500, 2009.

[10] E. Batot, "Generating examples for knowledge abstraction in mde: a multi-objective framework." in *SRC@ MoDELS*, 2015, pp. 1–6.

[11] X. Blanc and O. Salvatori, *MDA en action: Ingénierie logicielle guidée par les modèles*. Editions Eyrolles, 2011.

[12] J. Cabot and M. Gogolla, "Object constraint language (ocl): a definitive guide," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2012, pp. 58–90.

[13] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[14] D. Jackson, "Alloy: a lightweight object modelling notation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 2, pp. 256–290, 2002.

[15] T. A. A.-. Beta, 2005. [Online]. Available: http://alloy.mit.edu/index.php