Automatic Generation of Models from Their Metamodels Using Multilayer Perceptron Network

Karima Berramla[1,2][a], El Abbassia Deba[1][b] and Abou El Hassene Benyamina[1][c]
[1]LAPECI Laboratory, University of Oran1 Ahmed Ben Bella, Algeria
[2]University of Science and Technology Mohamed Boudiaf, Algeria

Keywords: Space Modeling, Models, Metamodels, Automatic Generation of Models, Model Driven-Engineering, OCL Language, Machine Learning Technique, Multilayer Perceptron Network.

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 work (Batot, 2015; Fleurey et al., 2009; Ben Fadhel et al., 2012) 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.

1 INTRODUCTION

During the 1990s, the software engineering has faced an exponential increase in the complexity of systems to be modeled 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 systems to be modeled and to minimize simultaneously development time and costs of new software especially those that are classified in the same family.

MDE (Schmidt, 2006) is therefore an approach to software engineering with which the model is defined as a first presentation of the system to be modeled, and which aims to construct, maintain and evolve the software based on transformations of this model. In a broad sense, MDE paradigm proposes to merge all aspects of 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 (Budinsky et al., 2003) and KM3 (Jouault et al., 2006). Then, a metamodel is an abstract description of the system to be modeled where its instantiations present models with which the transformation can be executed to have 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) (Soley et al., 2000) 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 classified 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

[a] https://orcid.org/0000-0002-2847-4895
[b] https://orcid.org/0000-0003-2948-2093
[c] https://orcid.org/0000-0003-4778-0123

Berramla, K., Deba, E. and Benyamina, A.
Automatic Generation of Models from Their Metamodels Using Multilayer Perceptron Network.
DOI: 10.5220/0012439300003645
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering (MODELSWARD 2024)*, pages 272-279
ISBN: 978-989-758-682-8; ISSN: 2184-4348

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 ((Gómez et al., 2012), (Ben Fadhel et al., 2012), (Ehrig et al., 2009) and (Batot, 2015)) 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 objctive is not only the automation of space modeling creation but also to simplify and facilitate the software development process by ensuring the transformation test (Berramla et al., 2017), (Berramla et al., 2016) and the programming phases by examples such as (Berramla et al., 2020).

The rest of this paper is structured as follows: Background, Motivation, and Problem statement are presented in section 2. Detailed description and evaluation of the proposed approach are defined in section 3 and 4. In section 5 we presents related work and finally section 6 concludes this paper.

2 BACKGROUND, MOTIVATION & PROBLEM STATEMENT

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

2.1 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 information about MDA technology which will be used in the rest of this paper.

More specifically, MDA has three types of models (Blanc and Salvatori, 2011) that are used in software development. In The following, we define each one and the relationship between them (see Figure 1):

Computation Independent Model (CIM). A CIM describes exactly what the system is supposed to do and masks all technical details related to system implementation.

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

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

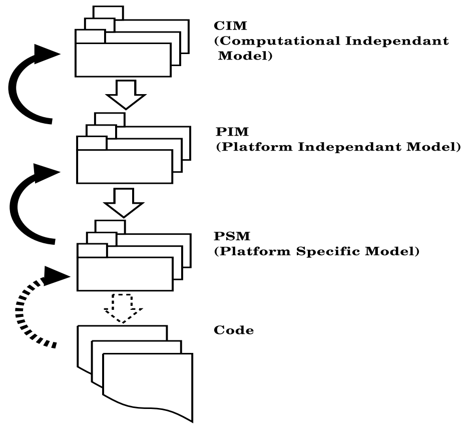
CIM
(Computational Independant Model)
PIM
(Platform Independant Model)
PSM
(Platform Specific Model)
Code

Figure 1: An overview of MDA approach (Adapted from (Blanc and Salvatori, 2011)).

Figure 1 gives an overview of MDA approach. In this context, building a new application starts with the specification of one or more requirements model (CIM). Then, it continues with the definition of analysis and abstract-design models (PIM). These models must be partially created from the CIMs that traceability relations are established. We have seen that PIM models are perennial models, which do not contain any information about the execution platforms. To create an application, it is necessary to define 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 (Blanc and Salvatori, 2011).

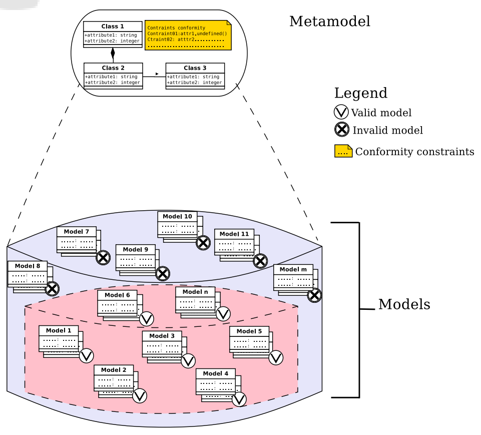

Figure 2: Modeling space and its limit (Adapted from (Gómez et al., 2012)).

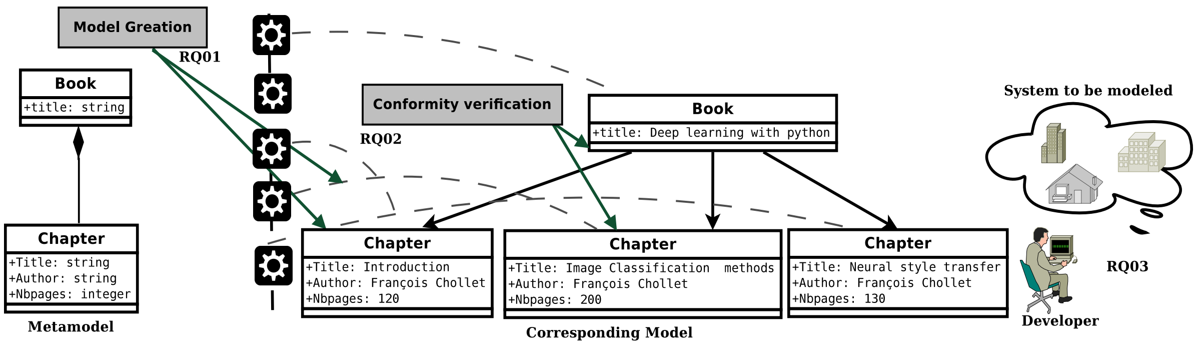

Figure 3: Modeling requirements.

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

Figure 2 borrowed from (Gómez et al., 2012) presents a general description about the space modeling and its limit. This space have two model types, the first one describes the well-formed models and the second defines the ill-formed models (see figure 2). When we generate the models from their metamodels we receive one of these two types, if the model is well instantiated and it verifies the defined constraints so we obtain the well-formed model otherwise we receive an 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.

2.2 Modeling Requirements

In this sub-section, we define the modeling requirements as questions to provide simple way of comprehension for readers. Also, we present these requirements schematically in the figure 3.

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 their conformity verification are done manually. This process requires 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 researches such as (Gómez et al., 2012). The goal of this paper is to solve this problem in different way compared with the previous work (Batot, 2015; Fleurey et al., 2009; Ben Fadhel et al., 2012).

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 a Technique to Simplify the System Modeling Process?

The recent software development such as MDA revolves around the use of models to separate the preoc cupations 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. Then, no proposed technique is considered as the best solution to simplify the developement software without mastering one or more tools or languages.

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.

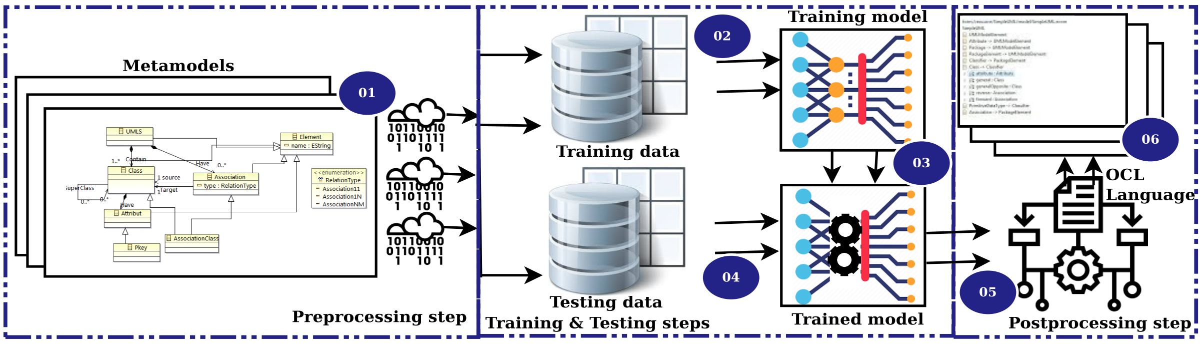

Figure 4: Proposed approach.

3 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 create 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.

3.1 Preprocessing Step

Here, we present our extraction process of all metamodel information that 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.

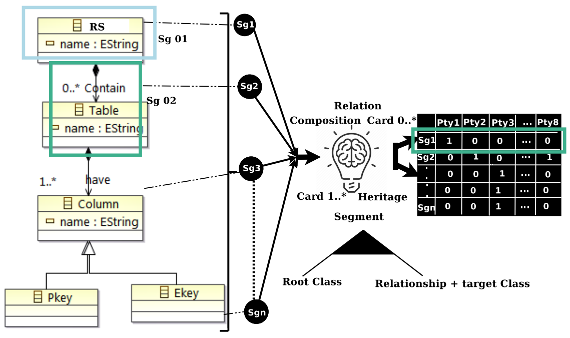

Figure 5: Extracted information & segment structure.

Figure 5 shows the basic information about each segment and digital extracted information. In this case, We consider a segment as one or two metamodel elements and each element can be either root class or relationship and its target class (see figure 5 exactly right side).

Table 1: Description of properties.

Property Number	***Property intituled***
P"1"	Root-class property.
P"2"	Heritage property.
P"3"	Composition property.
P"4"	Association property.
P"5"	Class-association property.
P"6"	1..1 Cardinality.
P"7"	0..n Cardinality.
P"8"	1..n Cardinality.

Once all segments are extracted, we test a set of properties for each segment in order to compute digital information about the used metamodel. These properties are descriped in the table 1.

For instance, the root class in figure 5 is defined as the segment number (1) and its digital information is shown in the first column of the table defined in the same figure. These binary information reflect the properties of each segment. In our case, figure 5 shows the Relational Schema (RS) 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.

Following table 2 gives these extracted information from RS metamodel as an example of data infor-

Table 2: Extracted information from RS metamodel.

	P"1"	*P"2"*	*P"3"*	*P"4"*	*P"5"*	*P"6"*	*P"7"*	*P"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

mation that will be used as input elements of training & testing steps.

3.2 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 computes 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 write the following equation 1:

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

where f is the transfer (or activation) 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

$$y = \begin{cases} 1, & \text{if } z \geq \theta, \\ 0, & \text{otherwise}, \end{cases} \quad (2)$$

which defines the binary output of the Perceptron. In the rest, we use this result to classify whether the input belongs to a specific class or not.

In the learning step, we modify the network parameters for example the weights and the bias to obtain good output result. This change is based on the following rule:

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

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 factor that permits in order to change the magnitude by which the weights are modified.

Once the network parameters are well-changed the testing step can be started using other input data information to evaluate the capacity of the network. In this step, we calculate diretly the ouput $\hat{y}$ without recompute the network parameters such as the weights.

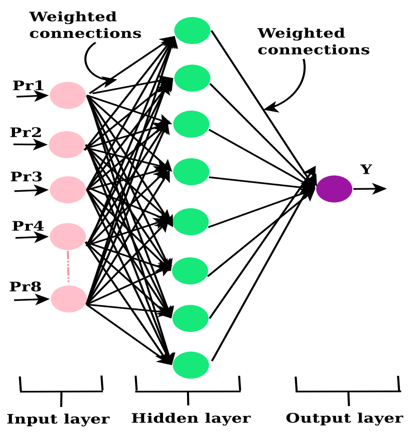

Figure 6: Our MLP architecture.

Figure 6 shows our MLP architecture where the input data are x_i and i was varied from 1 to 8 according to the used properties. These input data are defined by binary values of each segment while the ouput data $\hat{y}$ gives the binary information about each input segment if will be instancieted or not.

3.3 Posteprocessing Step

This step occupies the creation of models from MLP output using the following algorithm.

Algorithm 1: From MLP outputs To models.

```
Require: MLP Output;
Ensure: Creation of Models;
  Compute number of MLP outputs $N = size(Classes)$;
  Create set $C1 = outputs\ of\ Class1$;
  Create set $C2 = outputs\ of\ Class2$;
  for $i = 1$ to $N$ do
    while $Ci <> \emptyset$ do
      $C = dequeue(Ci)$;
      Let $X = Corresponding\ Segment(C)$;
      Compute number of Segments $K = size(X)$;
      for $J = 1$ to $K$ do
        Execute From Metamodel & Segment To instantiated_Segment (Xj);
        if AttrCont(Xj) <> $\emptyset$ then
          Execute Verify_contraint(AttrCont(Xj));
        end if
      end for
    end while
  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 two classes (in algorithm 1 N egals two) the first one regroups the segments that will be instantiated and the second class presents the segments that will not be instantiated.
Output Data. Output data of algorithm 1 is the models. From this Algorithm 1 we create the models and we verify some constraints with OCL language (Cabot and Gogolla, 2012) 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***.

Following algorithm 2 explains the segment instatiation process which extracts from metamodel and segment a part of model that is computed according to metamodel structure. Firstly, we find the segment position and its elements after, we instantiate these elements by using instantiation data. After that, we excute algorithm 3 to verify the conformity constraints.

Algorithm 2: From Metamodel & Segment To Instantiated Segment.

```
Require: Metamodel;
Require: Segment;
Ensure: Instantiated Segment;
1: Compute number of segment elements $N = size(Elt - Segment)$;
2: for $i = 1$ to $N$ do
3:     Xi= Elt-Segment_i;
4:     while Xi not instantiated do
5:         Find its position in its metamodel.
6:         create its definition in the output model by using Instantiation Data;
7:         Verify its indexation in its metamodel;
8:     end while
9: end for
```

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

Following algorithm 3 expresses verification of conformity constraints by translationg OCL constraints into instructions written in Java language. Verification conformity algorithm requires a lot of steps starting by reading and extracting OCL constraints, finding their elements and finishing by its translation into programming language.

Algorithm 3: Conformity contraint Verification.

```
Require: Instantiated Segment;
Ensure: Well-Instantiated Segment;
1: Compute number of segment elements $N = size(Elt - Segment)$;
2: for $i = 1$ to $N$ do
3:     Xi= Elt-Segment_i;
4:     while Xi not Verified do
5:         Find its constraints in its metamodel.
6:         Verify its constraints defintion in its Instantiated Elt-Segment;
7:     end while
8: end for
```

Implementation in Code. We use Java language to implement our approach. The metamodels are defined by Ecore language and their models are encoded in XML Metadata Interchange (XMI) using Eclipse Modeling Framework (EMF). In this case, to use and evaluate MLP classifier we manipulate Matlab.

4 EXPERIMENTATION & EVALUATION

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

4.1 Results & Interpretation

Here, we discuss on the basic information about each training and testing data examples that are mentionned in the following table 3.

Table 3: Basic information about training & testing data.

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

These information relate to the specification of the class and the segment numbers with which the example complexity is given.

Figure 7 illustrates the mean squared error of training, validation and testing data. From this fig-

Table 4: Comparative study of proposed approaches about model generation.

Authors	Input Elements	Progm-Language	Used Method	Output Elements
Our Paper	MM[a] & BiTable[b]	Java & Matlab	MLP & Algo	Models
(Gómez et al., 2012)	MM & Models	Java	Simulated-Annealing	Models
(Ehrig et al., 2009)	Meta-Model	AGG	Graph Grammars	Models
(Batot, 2015)	MM & Models	Java	NSGA-II	Models
(Jackson, 2002)	Meta-Model	Alloy	SAT Solvers	Models
(Wang et al., 2013)	MM & STM[c]	//	Genetic Algorithm	Models

[a]Meta-Model.
[b]Binary information table was defined in section 3.
[c]Structural metrics.

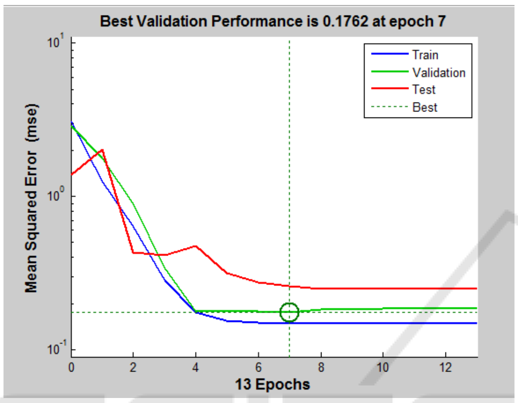

Figure 7: Performance of MLP classifier.

ure, we summarize that the training error reduces after two epochs and take the best value in seven epochs, but might stagnated on the training data set after five epochs. Generally, our MLP network takes the best values after seven epochs.

4.2 Threads 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 (Gómez et al., 2012), (Fleurey et al., 2009), (Ben Fadhel et al., 2012) and (Batot, 2015) our approach gives 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 2. For example, the ***RQ1*** and ***RQ3*** problems are solved by using "MLP" technique and our proposed algorithm 1 with which the models are created automatically without using an initial set of models and also by checking some levels of conformity without intervening the modeling designers where we answer also on ***RQ2***.

5 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 and we show the table 4 that summarizes these work by presenting some caracteristics for each one of them.

1) Using Optimization Techniques. The most important approaches for modeling space are based on metaheuristic techniques. For instance, this paper (Gómez et al., 2012) presents a metaheuristic approach for automatic generation of more precise models using Simulated Annealing (SA) in conformity with a set of criteria defined in (Fleurey et al., 2009). Another work (Ben Fadhel et al., 2012) 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 (Batot, 2015) has proposed also another approach to solve modeling space problem that allows using a non-dominated genetic algorithm (NSGA-II) using metamodel and a set of models.

2)Using Graph Grammar Methodology. Karsten et al. (Ehrig et al., 2009) propose 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 needs a lot of time and effort.

3) Using Formal Methods. In this context, the use of formal methods (Clarke and Wing, 1996) is not supported due to their difficulties nevertheless there are only some proposed work (Jackson, 2002). One best-known of them is (Jackson, 2002) that focused on instance generation by using Alloy (Beta, 2005). This instance generation is based on the translation of a class diagram to Alloy representation after, SAT solvers are used to create their 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 (Fleurey et al., 2009) describes one of the most genetic approaches by proposing a set of rules used to evaluate the correctness of input models.

6 CONCLUSION & FUTURE WORK

In MDE context, one of the most problems faced by developer's software is how to automate or facilitate model creation process in the software development system. In the last 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 by using "Multilayer Perceptron Network" to obtain good and verified models in order to reduce costs and 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

Batot, E. (2015). Generating examples for knowledge abstraction in mde: a multi-objective framework. In *SRC@ MoDELS*, pages 1–6.

Ben Fadhel, A., Kessentini, M., Langer, P., and Wimmer, M. (2012). Search-based detection of high-level model changes. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 212–221. IEEE.

Berramla, K., Deba, E. A., and Benhamamouch, D. (2016). Model transformation generation a survey of the state-of-the-art. In *2016 International Conference on Information Technology for Organizations Development (IT4OD)*, pages 1–6. IEEE.

Berramla, K., Deba, E. A., Benyamina, A., Touam, R., Brahimi, Y., and Benhamamouch, D. (2017). Formal concept analysis for specification of model transformations. In *2017 First International Conference on Embedded & Distributed Systems (EDiS)*, pages 1–6. IEEE.

Berramla, K., Deba, E. A., Jiechen, W., Sahraoui, H. A., and Benyamina, A. E. H. (2020). Model transformation by example with statistical machine translation. In *MODELSWARD*, pages 76–83.

Beta, T. A. A.-. (2005). http://alloy.mit.edu/index.php.

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

Budinsky, F., Brodsky, S. A., and Merks, E. (2003). *Eclipse Modeling Framework*. Pearson Education.

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

Clarke, E. M. and Wing, J. M. (1996). Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643.

Ehrig, K., Küster, J. M., and Taentzer, G. (2009). Generating instance models from meta-models. *Software & Systems Modeling*, 8(4):479–500.

Fleurey, F., Baudry, B., Muller, P.-A., and Le Traon, Y. (2009). Qualifying input test data for model transformations. *Software & Systems Modeling*, 8(2):185–203.

Gómez, J. J. C., Baudry, B., and Sahraoui, H. (2012). Searching the boundaries of a modeling space to test metamodels. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 131–140. IEEE.

Jackson, D. (2002). Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290.

Jouault, F., Bézivin, J., and Team, A. (2006). KM3: a dsl for metamodel specification. In *In proc. of 8th FMOODS, LNCS 4037*, pages 171–185.

Schmidt, D. C. (2006). Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25.

Soley, R. et al. (2000). Model driven architecture. *OMG white paper*, 308(308):5.

Wang, W., Kessentini, M., and Jiang, W. (2013). Test cases generation for model transformations from structural information. *MDEBE@ MoDELS*, 1104:42–51.