Model Transformation: Concept, Current Trends and Challenges

Pallavi Kalyanasundaram
K.K.W.I.E.E.R., Nasik
Maharashtra, India

Sunita P. Ugale
K.K.W.I.E.E.R., Nasik
Maharashtra, India

ABSTRACT
Model Driven Engineering (MDE) is gaining popularity as an alternative to the code-centric software development approach. Model Transformation (MT) is one of the main components of MDE. MT can be visualized as a program with models as inputs. Model evaluation and processing is automated by a Model Transformation tool. In this paper, we walk through the terminologies involved in MT and elaborate the benefits of MT with practical usage scenarios. The paper highlights the most recent challenges faced in the process to make model transformation more sophisticated. The intent of the paper is to portray a complete picture of model transformation in a way to relate the practical implementations with respect to the theoretical aspects of MT. The paper concludes by putting lights on some of the current trends in the field and the areas in model transformation where significant contribution is the needed.

General Terms
Modeling and Simulation, Software Development, Software Design Methodologies.

Keywords
Model Driven Engineering, Model Transformation, Model Transformation languages.

1. INTRODUCTION
The concept of software abstraction persists from the time when researchers felt the need to abstract/transform binary/machine language to a higher level language for their better visualization and understanding. Today, replicating the real world scenarios in terms of models is becoming popular as it gives a better conceptual view of the undertaken application specific problem thereby increasing the level of abstraction. These models are further processed or information is extracted from these models for application development. This software engineering methodology is known as Model Driven Engineering (MDE).

Model-driven engineering technologies offer a promising approach to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively [1]. MDE promises gains in productivity, interoperability, maintainability and portability [2]. The underlying concepts in MDE include 1) Domain Specific Modeling 2) Modeling Language 3) Meta-Modeling 4) Model Transformations 5) Code generation. A conceptual overview of MDE is shown in Fig 1.

As highlighted in Fig 1, the gist of MDE is the transformation of input model into output model or code depending upon the application. MDE is attracting more and more users as it automates the process of software development to a great extent by using model transformation tools. The automation part of MDE lies inside the transformation block. As seen from Fig 1, models are represented by using a modeling language, example, Universal Modeling Language (UML). The modeling languages can be domain specific like MATLAB Simulink. Every ML is defined using a metamodel. A metamodel can be considered as the grammar of a ML.

The transformation of models is performed using a transformation language. Examples of transformation languages include: ATL (ATLAS Transformation Language), QVT (Query/View/Transformation) etc. The transformation description is specified by the metamodel. The transformation rules map the input metamodel to the output metamodel. The transformation rules are specified in the transformation language. Code can also be considered as a special type of model. The auto-generated code after transformation may be platform specific or generic as per requirement. Embedded Coder tool from MathWorks automatically generates C/C++ codes optimized for embedded processors and various other platforms [3].

Section 2 discusses the model transformation process in detail with the help of a block diagram. It specifies the terminologies which are frequently used while dealing with model transformations. It also describes the uses of model transformation along with one practical usage scenario for each. This is followed by a review of some current work related to model transformations with respect to the need for implementation, contribution and further challenges.

2. MODEL TRANSFORMATION (MT)
Model transformation is the heart of Model Driven Engineering. Model transformation tools automate the process of transformation of input models to desired output models. [4] Talks about the practices followed in the MDE industry which includes some survey results after interviewing many experts in the fields. The results demonstrate that 72% respondents use model to model transformation and 88 % incorporate automatic code generation in the application development process. These statistics state the significance of model transformation.

Model Transformation has its roots in compiler designing. [5] Mentions that an MDE developer needs to have both compiler development as well as abstraction skills. The terminologies that a designer should be well-versed while developing a model transformation tool is described in detail in this section.

2.1 Concept and Terminologies
Fig 2 explains the model transformation process. The flow of the whole process is indicated by arrows in Fig 2. The significance of each block in Fig 2 is described next. This section tries to cover the common terminologies and concepts around which model transformation revolves.

Modeling Language: It is the language in which models are represented. Modeling languages can be domain specific. They formalize the structure, behavior and requirements of a particular domain [1]. Examples include: UML [6], MATLAB Simulink [7], SysML[6], CPN (Colored Petri nets) [8] etc. Source model and target model are represented using modeling languages.

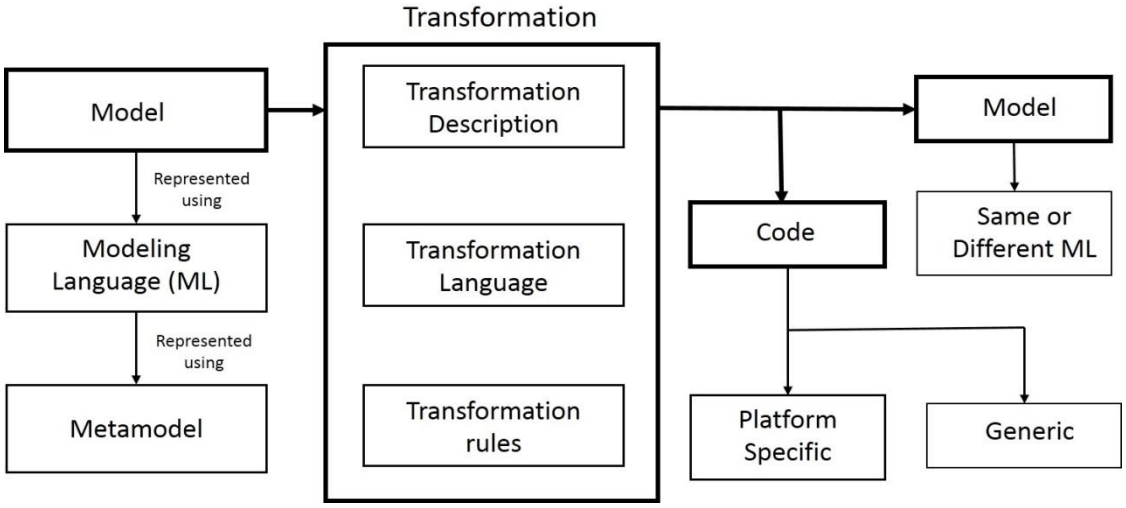

Fig 1: Model Driven Engineering - Conceptual Overview

Metamodel: As stated in Section 1, a metamodel can be considered as the grammar of a modeling language. Metamodeling is an architectural abstraction that provides the foundations for construction, manipulation and validation of models [6]. [7] Shows an excerpt from the Simulink metamodel which gives an idea to the Simulink users to visualize a metamodel.

Model Transformation Rule: These rules define the relation between the constructs in the source and target models which varies depending on the application. These rules are written in model transformation language. These rules specify "what" needs to be transformed to "what". [7] Uses graph transformation rules to repair Simulink models.

Model Transformation Description: It specifies "how" the transformation rules are executed to complete the transformation process. It is the one of the main components of the model transformation process as shown in Fig 2.

Platform Independent Model (PIM): The models which are not intended for a specific platform. The information about the platform in not present in the model.

Platform Specific Model (PSM): The models which are intended for a specific platform. The information about the platform is present in the model itself.

Model Driven Architecture (MDA): A software design approach launched by Object Management Group. It is used to transform the PIM to PSM.

Technical Space: Also known as technological space, it represents the associated concepts, knowledge, tools and required skills in the given context. Examples include: Model Driven Architecture (MDA) or XML-based Languages.

Endogenous transformation: The transformation in which the input and output modeling languages (more specifically metamodels) are same. This transformation can also be termed as rephrasing transformation [5].

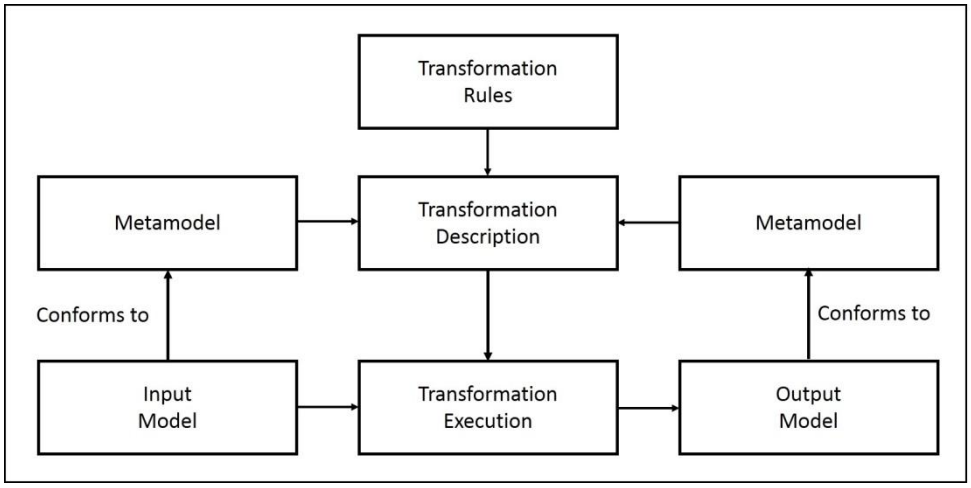

Fig 2: Model Transformation Process

Exogenous transformation: The transformation in which the input and output modeling languages (more specifically metamodels) are different. This transformation is also called translational transformation [5].

Bidirectional Transformation: The transformation of source model to target model and vice versa can be achieved.

Declarative MT Approach: They define the relation between the source and target model. "What needs to be transformed into what" [9].

Operational MT Approach: They specify *how* the transformation has to be executed rather than the *what* aspect.

Rule scheduling and control: There may be more than one transformation rule for a given transformation process. The order in which the transformation rules are executed or scheduled must also be defined. The schedule can be controlled implicitly or explicitly.

Model Transformation Languages: Programming languages can be used for performing model transformations. But a dedicated domain-specific programming language would be much preferable as the technical space would reduce. These domain specific languages are nothing but model transformation languages.

There are many popular model transformation languages. Few of them are mentioned below: Atlas Transformation Language (ATL)[10], Query/View/Transformation (QVT)[6], etc.

Eclipse Modeling Framework (EMF): It is a modeling framework and provides code generation facility. A structured data model aids to build the tools and applications.

2.2 Uses of Model Transformation

2.2.1 Automatic Code Generation from models:

One of the important uses of model transformation is to autogenerate code from models. These tools are termed as code generators. Most companies appear to experience productivity increases of between 20-30% [4] by incorporating automatic code generation from models in their software development process. As mentioned in Section 1, the Embedded Coder tool from Mathworks is a perfect example in this context. Another example include an open source code generator, Acceleo, which uses any EMF models to auto-generate code like Java, PHP, Python etc. [8] uses Acceleo tool to generate XML-like text document for colored Petri nets (CPN) from UML behavioral State machine diagrams (SMDs).

2.2.2 Improving Quality of Models:

Model transformations are used for improving the quality of models. Here, the models are at same level of abstraction and are modified for the non-functional properties. This can be categorized as an example of Endogenous transformation. [7] Proposes a tool which identifies and rectifies the MATLAB Simulink models if they are not complaint with the MAAB (MathWorks Automotive Advisory Board) standards. These guidelines need to be followed so that the same standards are followed across by all MATLAB Simulink users. Some violations handled includes identifying and converting three inputs to the Simulink product block into two cascaded product blocks with two operators each. It also checks for the line intersections which are corrected by changing the port order of inputs/outputs.

2.2.3 *Code to model conversion:*

This is a reverse engineering concept where model is generated from the code using some tools. Many companies refrain to adopt the MDE practice because of the huge legacy codes. [11] Has built a proof of concept of a tool which converts the legacy C codes to MATLAB Simulink models. C2M tool, as they mention, is a static C code analysis tool which converts the C code to an intermediate XML representation. The XML file is used to generate an m-script (.m (MATLAB) file) which is executed on MATLAB to generate Simulink models. The programming language 'C' is used to handle all the transformations instead of a standard model transformation language.

2.2.4 *Model to Model Transformation:*

This process involves the conversion from one modeling language to another depending upon the application. The model transformation here means the mapping of the two metamodels (input and output models) based on the transformation rules. Example: [6] converts models represented in SysML to UML. It illustrates a methodological template for Model Driven System Engineering (MDSE) for the development of a software-intensive system in the naval electronic warfare domain. The input model is represented using SysML modeling language which is converted to UML with the help of transformation rules specified using QVT transformation language. Another example can be Simulink to UML transformation for embedded control software application [13].

2.2.5 Model Merging:

Input to the transformation tool can be two or more metamodels represented using same or different modeling languages. The tool combines these metamodels to form one metamodel. Model merging is used in aspect oriented modeling or aspect weaving [5].

2.3 Current Trends and Challenges in Model Transformation

The applications mentioned below are the recent contributions made by different researchers in the field of MDE related to model transformation. The applications are reviewed in a way that the terms used in model transformations are highlighted so as to bridge the gap between the theoretical and practical aspects of MT. The challenges faced previously, work done in the direction to overcome those challenges and the future challenges that need to be addressed are also mentioned.

As mentioned in Section 2.2.3 Automatic code generation in MDE seems to increase productivity. However, modifications made to the auto-generated code are not reflected in models unless done manually. Moreover, extensive manual effort and time is needed to convert the legacy application codes to models. The above challenges are results of survey conducted by [2]. Proceeding towards overcoming these challenges, [12] have developed a proof of concept of tool which can automatically convert code to model. C2M tool description was elaborated in section 2.2.3. The model generated by C2M tool needs manual verification which can become tedious if models are huge. The further challenge in this direction could possibly include reducing the complexity of huge models.

For the application mentioned in Section 2.2.4 proposed by [6] elaborates the transformation of PIM represented using SysML to PSM which is represented using UML. The transformation approach is operational and implemented using

QVT. The need is for analysis and production of complex system for which the template is proposed. To blend model checking and validation with the template would be a challenging task ahead.

[7] Corrects for MAAB guidelines violations of MATLAB/Simulink models as briefed in 2.2.3. Graphical transformation rules are used to complete the transformation. The complexities of these rules are directly proportional to models. They use declarative transformation approach.

An MDE 'guru' needs to have good abstraction, compiler development and domain specific skills which follow huge training cost. Hence, the MDE and domain experts work together as a team which leads to the success of MDE practice in an organization. [14] Mentions that the domain experts give transformation examples easily than complete and consistent transformation rules. Model Transformation by Example (MTBE) is performed wherein the domain experts give the transformation examples from which transformation rules are extracted automatically. Operational transformation rules are generated but the limitation is that, MTBE approach cannot perform transformations in which new values are computed.

In order to trace the affected blocks in a complex model by the change in some parameter of the system, [11] have proposed automatic analysis of Simulink models. The affected blocks can be visualized in Simulink or external graphical editor. They compare two approaches for model analysis: 1) model transformation approach using EMF and 2) database method which uses database queries for model analysis. ATL transformation language is used in model transformation approach. They specify that hybrid approach of combining both the database and MT approach is better in comparison to model transformation alone.

3. CONCLUSION

Model driven engineering has potential features to replace the traditional software development process. Therefore, knowledge of Model Transformation becomes very essential. A complete roadmap of model transformation is discussed here. There are very few terminologies around which model transformation revolves. These terminologies were defined with some specific examples to get a clear understanding. The use of model transformation along with its practical implementation is also elaborated. It is observed that, embedded software applications development is one of the major application areas where model transformations play an important role.

Various applications were studied with a few mentioned in Section 2.3. Some of the most prominent challenges faced today include model checking and validation of complex models. It is observed that MATLAB Simulink is widely used in the automotive industry. Lot of work related to reducing the complexity of these models is being undertaken. Domain specific modeling is a current trend generally optimized for a particular application domain. Generalization of these models across different domains is a great challenge ahead.

4. REFERENCES

[1] Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. Computer, 39(2), 0025-31.

[2] Hutchinson, J., Whittle, J., & Rouncefield, M. (2014). Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. Science of Computer Programming, 89, 144-161.

[3] Embedded Coder: http://in.mathworks.com/products/datasheets/pdf/embedded-coder.pdf

[4] Whittle, J., Hutchinson, J., & Rouncefield, M. (2014). The state of practice in model-driven engineering. Software, IEEE, 31(3), 79-85.

[5] Biehl, M. (2010). Literature study on model transformations. Royal Institute of Technology, Tech. Rep. ISRN/KTH/MMK.

[6] Bocciarelli, P., D'Ambrogio, A., Caponi, E., Giglio, A., & Paglia, E. (2014). A Methodological Template for Model Driven Systems Engineering. In INCOSE Italia Conference on Systems Engineering (CIISE 2014) (pp. 48-58).

[7] Stürmer, I., Dziobek, C., & Pohlheim, H. (2008, April). Modeling Guidelines and Model Analysis Tools in Embedded Automotive Software Development. InMBEES (pp. 28-39).

[8] André, E., Benmoussa, M. M., & Choppy, C. (2014). Translating UML state machines to coloured Petri nets using Acceleo: A report. arXiv preprint arXiv:1405.1112.

[9] Mens, T., & Van Gorp, P. (2006). A taxonomy of model transformation.Electronic Notes in Theoretical Computer Science, 152, 125-142.

[10] Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). ATL: A model transformation tool. Science of computer programming, 72(1), 31-39.

[11] Merschen, D., Gleis, R., Pott, J., & Kowalewski, S. (2013). Analysis of Simulink Models Using Databases and Model Transformations. In Model-Based Methodologies for Pervasive and Embedded Software (pp. 69-84). Springer Berlin Heidelberg.

[12] Smitha, K. P., Ranadive, P., Boggarapu, N., & Rakesh, A. (2015). Automatic C to Simulink Model Converter (C2M) Tool (No. 2015-01-0164). SAE Technical Paper.

[13] Kamiyama, T., Soeda, T., Yoo, M., & Yokoyama, T. (2010). A Simulink to UML Transformation Tool for Embedded Control Software Design. In Proceedings of 2010 International Conference on Computer and Software Modeling (pp. 93-97).

[14] Saada, H., Dolques, X., Huchard, M., Nebut, C., & Sahraoui, H. (2012).Generation of operational transformation rules from examples of model transformations (pp. 546-561). Springer Berlin Heidelberg.

[15] Stürmer, I., & Travkin, D. (2007). Automated Transformation of MATLAB Simulink and Stateflow Models. In Proc. of 4th Workshop on Object-oriented Modeling of Embedded Real-time Systems (pp. 57-62).

[16] Silva, G. C., Rose, L., & Calinescu, R. (2014). A Qualitative Study of Model Transformation Development Approaches: Supporting Novice Developers. InProceedings of the 1st International Workshop in Model-Driven Development Processes and Practices (MD2P2) (pp. 18-27).

[17] France, R., & Rumpe, B. (2007, May). Model-driven development of complex software: A research roadmap. In 2007 Future of Software Engineering (pp. 37-54). IEEE Computer Society.

[18] Tamura, M., Kamiyama, T., Soeda, T., Yoo, M., & Yokoyama, T. (2012). A Model Transformation Environment for Embedded Control Software Design with Simulink Models and UML Models. In Proceedings of the International MultiConference of Engineers and Computer Scientists (Vol. 1).

[19] Zhang, L., Glab, M., Ballmann, N., & Teich, J. (2013, September). Bridging algorithm and ESL design: Matlab/Simulink model transformation and validation. In Specification & Design Languages (FDL), 2013 Forum on (pp. 1-8). IEEE.

[20] Favre, J. M. (2004, October). Towards a basic theory to model model driven engineering. In 3rd Workshop in Software Model Engineering, WiSME (pp. 262-271).

[21] Niere, J., Schäfer, W., Wadsack, J. P., Wendehals, L., & Welsh, J. (2002, May). Towards pattern-based design recovery. In Proceedings of the 24th international conference on Software engineering (pp. 338-348). ACM.

[22] Giese, H., Meyer, M., & Wagner, R. (2006). A prototype for guideline checking and model transformation in Matlab/Simulink. In Proc. of the 4th International Fujaba Days (pp.56-60).