Proceedings of the 2011 Winter Simulation Conference
S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, eds.

METAMODELING AND MODEL TRANSFORMATIONS IN MODELING AND SIMULATION

Deniz Cetinkaya
Alexander Verbraeck

Systems Engineering Group, Faculty of Technology, Policy and Management
Delft University of Technology
Jaffalaan 5, 2628BX, Delft, THE NETHERLANDS

ABSTRACT

Metamodeling and model transformations are the key concepts in Model Driven Development (MDD) approaches as they provide a mechanism for automated development of well structured and maintainable systems. However, neither defining a metamodel nor developing a model transformation is an easy task. In this paper, we provide an overview of metamodeling and model transformations in MDD and discuss about the use of a MDD approach in Modeling and Simulation (M&S). In order to support the development of successful model transformations, we define the criteria for the evaluation of model transformations.

1 INTRODUCTION

In Model Driven Development (MDD), models are the primary artifacts of the development process and they represent the system on different levels of abstraction. A model is an abstract representation of a system defined in a modeling language, where a modeling language is a means of expressing systems in a formal and precise way by using diagrams, rules, symbols, signs, letters, numerals, etc..

Metamodeling is widely accepted as a superior way to describe models in MDD context. A metamodel is a definition of a modeling language, which should be defined in a modeling language as well (Sprinkle et al. 2007). A sound metamodel describes models in a well-defined manner. Model transformations are used to create new models based on the existing information throughout the MDD process (Sendall and Kozaczynski 2003). Instead of creating the models from scratch during the different development life-cycle stages and activities, model transformations enable the reuse of information that was once modeled. Different models represent different views of the system.

MDD is increasingly gaining the attention of both industry and research communities in Modeling and Simulation (M&S) field (Mészáros, Levendovszky, and Mezei 2009, Wang, Albani, and Barjis 2011). Metamodeling term has been used in simulation for many years in a different context. Metamodeling referred to constructing simplified models when simulation models were quite complex (Barton 1998). In this context, a metamodel is known as a surrogate model. Surrogate models mimic the complex behavior of the underlying simulation model. Metamodeling in MDD context and model transformations have been introduced to simulation lately (Vangheluwe and Lara 2002).

In this paper, we provide an overview of metamodeling and model transformations in MDD and give clear definitions for the key terms. After that, we discuss about the use of MDD approach in M&S. Then, we define the criteria for the evaluation of model transformations in Section 4. Lastly, we present conclusions and future directions in Section 5.

2 BASIC MDD CONCEPTS

MDD is based on a number of principles that involve the concepts of model, metamodel, meta-metamodel and model transformations in order to produce a model that contains enough details for (semi)automatic generation of executable code. Due to the fact that some of the terms may have different meanings in different contexts, we give clear definitions and detailed explanation for basic MDD concepts in this section.

2.1 Metamodeling

As stated in the previous section, a model is an abstract representation of a system defined in a modeling language. A modeling language consists of the abstract syntax, concrete syntax and semantics. The abstract syntax describes the vocabulary of concepts provided by the modeling language and how they can be connected to create models. It consists of the concepts, the relationships and well-formedness rules, where well-formedness rules state how the concepts may be legally combined. The concrete syntax presents a model either in a diagram form (visual syntax) or in a structured textual form (textual syntax). Semantics of a modeling language is the additional information to explain what the abstract syntax actually means (Harel and Rumpe 2004).

Metamodeling is the process of complete and precise specification of a domain specific modeling language, which in turn can be used to define models of that domain (Achilleos, Georgalas, and Yang 2007). A Domain Specific Modeling Language (DSML) is a modeling language dedicated to a particular problem domain and/or a particular solution technique. DSMLs are closer to the problem domain and concepts than general purpose modeling languages such as UML. A metamodel describes the abstract syntax of a modeling language and it is defined in a metamodeling language. A metamodeling language is a superior language to describe new modeling languages and it is also defined with a metamodel. The metamodel of the metamodeling language is called as meta-metamodel. Model, metamodel, and meta-metamodel form a three-level metamodeling pattern as shown in Figure 1. The MDD pattern can be applied several times and a hierarchy of abstraction layers can be built, where models at layer *n* are specified using the language defined as a metamodel at layer *n+1* (Achilleos, Georgalas, and Yang 2007).

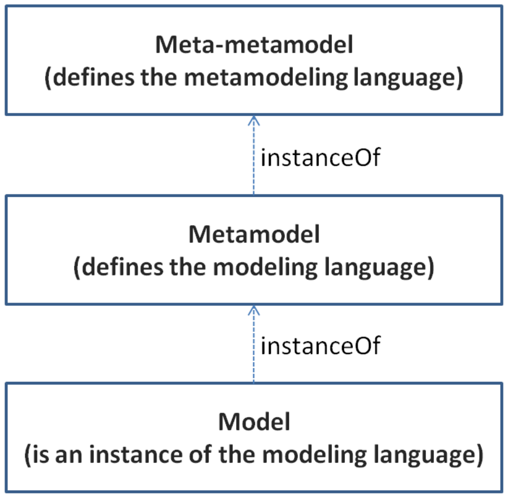
Meta-metamodel
(defines the metamodeling language)
instanceOf
Metamodel
(defines the modeling language)
instanceOf
Model
(is an instance of the modeling language)

Figure 1: Metamodeling pattern in MDD.

MDD approach is used by different technologies like Model Driven Architecture (MDA), Model Integrated Computing (MIC), and many others. Hence, there are several metamodeling languages, yet the most common ones are Meta Object Facility (MOF), Eclipse ECore, MetaGME, and Kermeta meta-metamodel. The MOF Specification is the foundation of Object Management Group's (OMG) industry-standard environment where models can be created, integrated and transformed into different formats. OMG's MDA relies on the MOF to integrate the modeling steps and provide the model transformations (OMG 2003). The earlier version of the MOF meta-metamodel is a part of the ISO/IEC 19502:2005 standard (ISO/IEC 2005). The MOF meta-metamodel is referred as the *MOF Model* and MOF specification uses numbered layers instead of using the *meta-* prefix. Although the classical MOF Specification is based on four layers, the later versions allow more or less meta-levels than this, depending on how MOF is deployed. However, the minimum number of layers is two. Figure 2 shows the metamodeling layers in MOF (ISO/IEC 2005). The information layer is comprised of the specific runtime data that the modeler wishes to describe. The MOF Model is self-describing, i.e. it is formally defined using its own metamodeling constructs. In 2006, OMG introduced the Essential MOF (EMOF) which is a subset of MOF with simplified classes (OMG 2006). The primary goal of EMOF is to allow defining metamodels by using simpler concepts while supporting extensions.

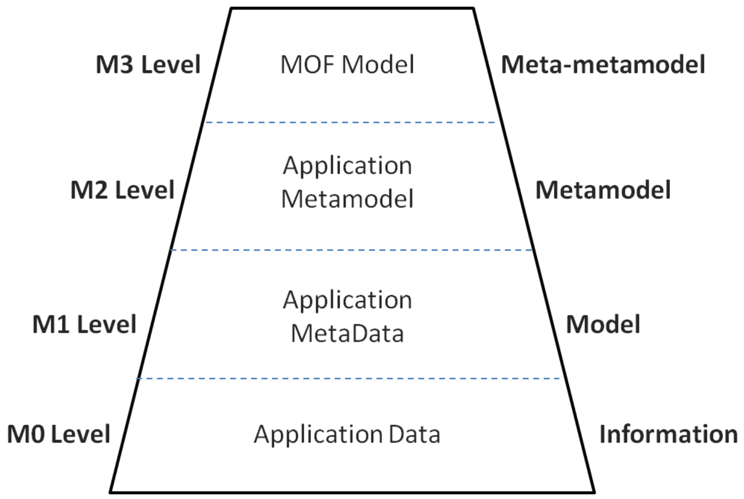

Figure 2: Metamodeling layers in MOF.

Eclipse Ecore meta-metamodel is the main part of The Eclipse Modeling Framework (EMF) project (EMF 2011). The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. Ecore is based on the EMOF specification, but renames the metamodeling constructs like EClass, EAttribute, EReference or EDataType. An Ecore metamodel needs to have a root object and a tree structure represents the whole model.

MetaGME is the top level meta-metamodel of the MIC technology (ISIS 2011). MIC defines a technology for the specification and use of the DSMLs. MIC refines and facilitates MDD approaches and provides an open integration framework to support formal analysis tools, verification techniques and model transformations in the development process. MIC allows the synthesis of application programs from models by using customized Model Integrated Program Synthesis (MIPS) environments.

Kermeta is a metamodeling language which allows describing both the structure and the behavior of models (IRISA 2011). Kermeta is built as an extension to EMOF and it provides an action language for specifying the behavior of models. Kermeta development environment provides a comprehensive tool support for metamodeling activities, such as an interpreter, a debugger, a text editor with syntax highlighting and code auto completion, a graphical editor and various import/export transformations. Kermeta is fully integrated with Eclipse and it is available under the open source Eclipse Public License (EPL).

A modeling tool provides ways to develop models based on a modeling language. In most cases, the graphical modeling tools are extended with extra tools such as a model transformer, a mapper, a compiler, an interpreter, a consistency checker or a debugger (where a tool is an instrument to perform some task). In this case, the complete tool set is called as a modeling environment. A metamodeling environment is a tool used to develop a modeling tool by defining its metamodel in a predefined meta-metamodel. The resulting tool may either work within the metamodeling environment, or less commonly be produced as a separate standalone program. Metamodeling environments are known as Domain Specific

Modeling (DSM) environments in MDD literature. Common DSM environments are Microsofts DSL Tools for Software Factories, MetaEdit+, XMF-Mosaic, DOmain Modelling Environment (DOME), Generic Modeling Environment (GME), AndroMDA and Eclipse Generative Modeling Technologies (GMT) project. Achilleos et al. (2007) and Amyot et al. (2006) present detailed discussion about various metamodeling environments.

2.2 Model Transformations

Once the models are constructed conforming to their certain metamodels, we can talk about the model transformations. A model transformation is the process of converting a source model into a target model according to a set of transformation rules. The rules are defined with a model transformation language (Mens and Gorp 2005). A transformation rule consists of two parts: a left-hand side that accesses the source model; and a right-hand side that expands on the target model. The source model conforms to the source metamodel and the target model conforms to the target metamodel. During the transformation, the source model remains unchanged. Figure 3 shows a model transformation pattern adapted from MDA (OMG 2003).

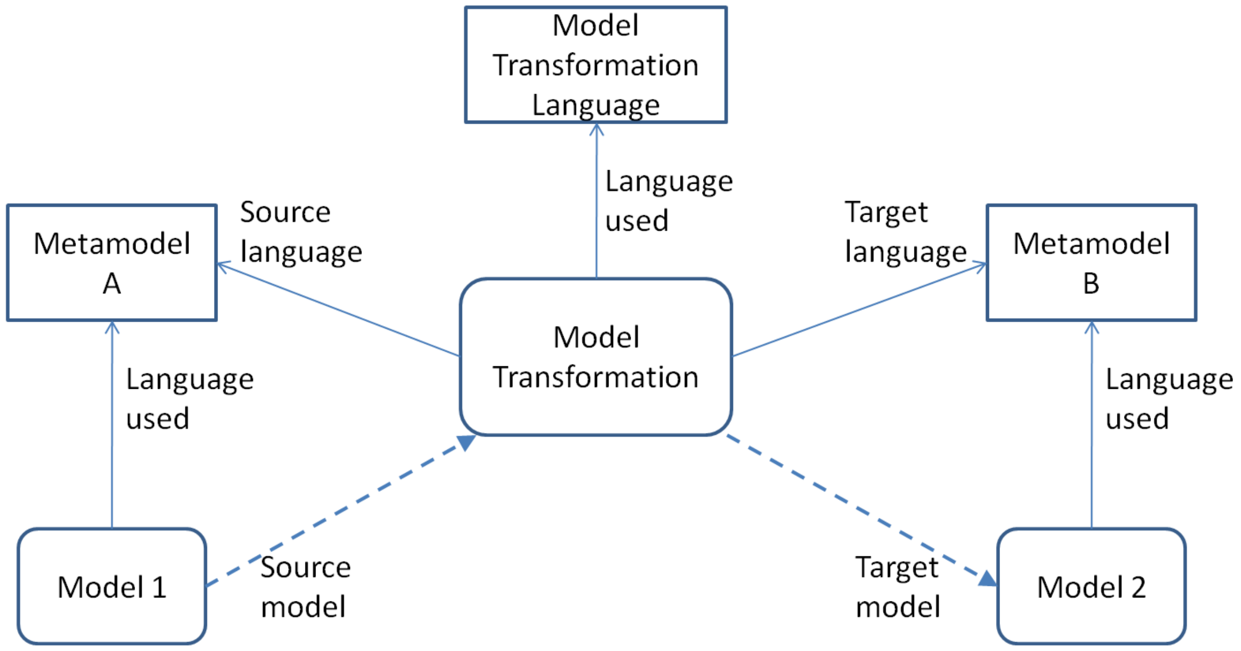

Figure 3: Model transformation pattern in MDD.

If the source and target metamodels are identical, the transformation is called as endogenous; otherwise it is called as exogenous (Mens and Gorp 2005). If the level of abstraction does not change, the transformation is called as horizontal transformation. If the level of abstraction does change, the transformation is called as vertical transformation.

According to the type and complexity of the process, the model transformation can be named as model to model (M2M) transformation, model to text (M2T) transformation, model merging, model linking, model synthesis, model mapping or model to code transformation (code generation). In a MDD approach, usually there is a chain of several M2M transformations and a final model-to-code transformation. M2M transformations are used with the different intermediate models, which are needed for expressing different views of a system, optimization, tuning or debugging purposes (Czarnecki and Helsen 2003).

As the name implies, a M2M transformation converts a source model into a target model, which can be instances of the same or different metamodels. However, a M2T transformation converts a source model into a text file. M2T transformation is generally used for final code generation or supportive document creation. Although there are different solutions for transforming models to text, this is not the case for transforming models to models (Czarnecki and Helsen 2003). Since M2M transformation is a relatively young area, very limited and often ad hoc transformation capabilities are offered by the modeling environments available on the market.

Well known M2M transformation languages are ATL (ATLAS Transformation Language) and QVT (Query/View/Transformation). MOF 2.0 QVT Specification is a set of model transformation languages defined by the OMG (OMG 2011). The QVT specification has a hybrid declarative/imperative nature and it defines three related transformation languages: Relations, Operational Mappings, and Core. Relations metamodel and language supports complex object pattern matching and object template creation with a high-level user-friendly approach. Operational Mappings can be used to implement one or more Relations from a Relations specification, when it is difficult to provide a purely declarative specification of how a Relation is to be populated. Core metamodel and language is defined using minimal extensions to EMOF and Object Constraint Language (OCL) on the lower level.

ATL is one of the most popular model to model transformation languages (Bézivin et al. 2003). Once the target metamodel and source metamodel are available, an ATL transformation file can produce a target model from a source model. During the transformation, an attribute of a target model instance receives a return value of an OCL expression that is based on a source model instance. The ATL Integrated Development Environment (IDE), that is developed on top of the Eclipse platform, provides a number of tools such as a text editor with syntax highlighting, a debugger and an interpreter that aims to ease development of ATL transformations.

There are many other model transformation languages (Scheidgen 2006). As well as, some projects like Kermeta, Fujaba and GReAT (these projects apply a model driven development approach) provide M2M transformation methods. Detailed explanation about the model transformation languages and methods is given in (Dehayni et al. 2009) and (Huber 2008, Czarnecki and Helsen 2003).

At this point, it is important to note that the possibility of defining various model transformations for the same source model. This is illustrated in Figure 4 by extending the ATL 'Families to Persons' example given in (ATL 2007). The Family model conforms to the Family metamodel whereas the Person model conforms to the Person metamodel. Families2Persons.atl is a simple illustration of M2M transformation. If we define another model transformation file as Families2Mothers.atl, then it can produce a different target model from the same source model. Furthermore, it is also possible to change the target metamodel, and so the model transformations. However, defining precise metamodels and choosing the best model transformation are the challenging activities of MDD.

3 MDD APPLIED INTO SIMULATION

The research on MDD in simulation field has emerged in the last decade. De Lara and Vangheluwe (2002) present a tool for metamodeling and model transformations for simulation, namely AToM3. They introduce the combined use of multi-formalism modeling and metamodeling to facilitate computer assisted modeling of large systems. The usage of the tool is presented in (Vangheluwe and Lara 2002).

Duarte and de Lara (2009) show the application of MDD to agent based simulation. They present the design of a DSML for modeling and simulation of multi agent systems. The language is made of four different diagram types, that are used to define agents types, their behavior, their sensors and actuators and the initial configuration. A customized modeling tool developed in Eclipse and a code generator for a Java-based agent simulation language are presented as well.

Iba, Matsuzawa, and Aoyama (2004) propose a simulation model development process and present an example for agent based social simulations. The proposed process consists of three major phases: conceptual modeling, simulation design and verification. In the conceptual modeling phase, the modeler

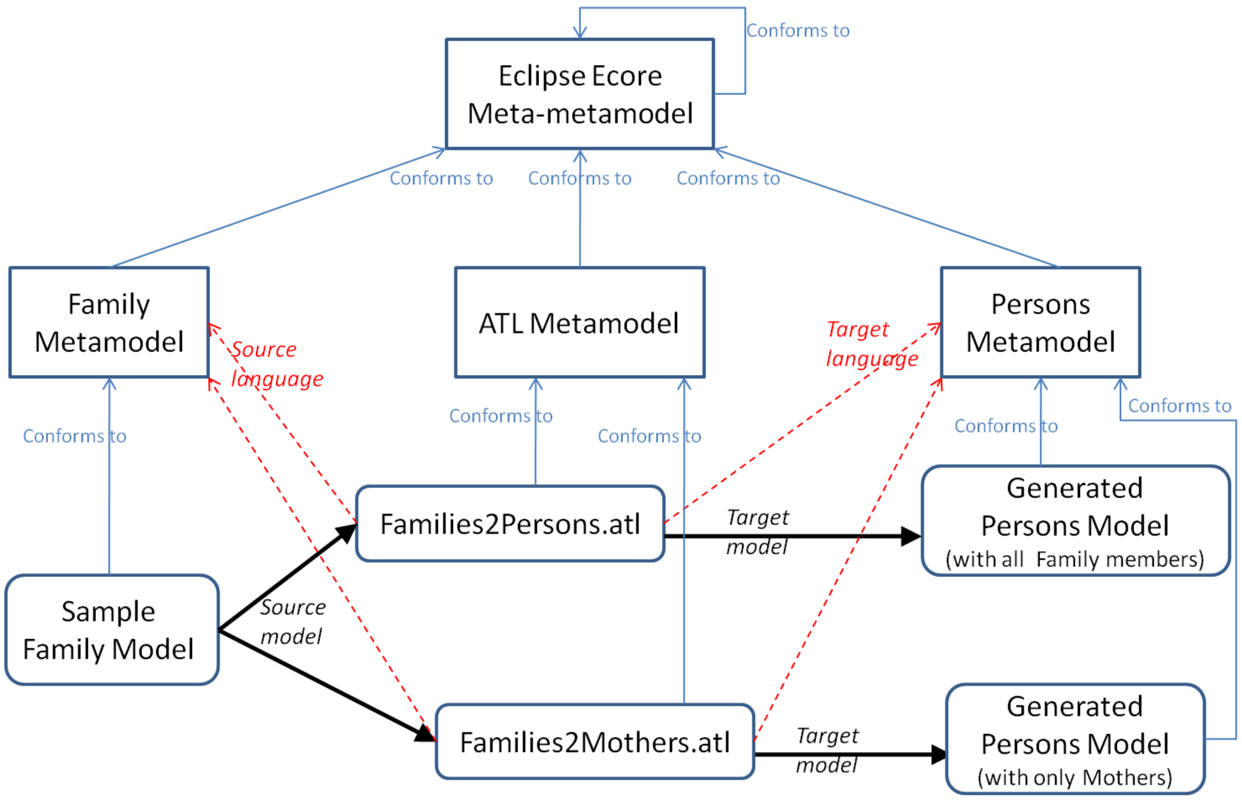

Figure 4: Defining various model transformations for the same source model.

analyzes the target world and describes the conceptual model. In the simulation design phase, the modeler designs and implements the simulation model, which is executable on the provided simulation platform. The modeler translates the conceptual models into simulation models according to the suggested 'Foundation Model Framework'. In the verification phase, the modeler runs the simulation and inspects whether the simulation program is coded rightly. If necessary, the modeler returns to the first or second phase and modifies the models.

Guiffard et al. (2006) provide a study that aims at applying a model driven approach to the M&S in military domain. This work has been carried out in the context of a larger research program (High Performance Distributed Simulation and Models Reuse) sponsored by the DGA (Delegation Generale pour l'Armement) of the French Ministry of Defense. The paper presents a prototype implementation as well. The prototype demonstrates that the automated transformation from a source model to executable source code is possible. The authors state that the amount of work needed for writing correct and complete set of transformation rules is extremely large.

Topcu, Adak, and Oguztuzun (2008) propose the Federation Architecture Metamodel (FAMM) for describing the architecture of a High Level Architecture (HLA) compliant federation. FAMM supports the behavioral description of federates based on live sequence charts and it is defined with metaGME. FAMM formalizes the standard HLA Object Model and Federate Interface Specification.

D'Ambrogio et al. (2010) introduce a model driven approach for the development of DEVS simulation models. The approach enables the UML specification of DEVS models and automates the generation of DEVS simulations that make use of the DEVS/SOA implementation. An example application for a basic queuing system is also presented. In its present form, the behavior of the atomic models are not implemented and the approach only produces the core skeleton of Java classes that implement DEVS models.

Cetinkaya, Verbraeck, and Seck (2010) motivate the use of a MDD approach to move the focus of M&S from coding to modeling. The presented work suggests an MDD applied M&S life-cycle with five stages: (1) Problem Definition, (2) Conceptual Modeling, (3) Specification, (4) Implementation and (5) Experimentation. As a follow up study, Cetinkaya, Verbraeck, and Seck (2011) propose a MDD framework for modeling and simulation (MDD4MS). MDD4MS framework defines three metamodels for Simulation Conceptual Model (CM), Platform Independent Simulation Model (PISM) and Platform Specific Simulation Model (PSSM). CM is an instance of Simulation Conceptual Modeling Metamodel, PISM is an instance of Simulation Model Specification Metamodel, and PSSM is an instance of Simulation Model Implementation Metamodel. In order to obtain the continuity throughout the M&S life-cycle, the model transformations are defined as: CM_2_PISM transformation is a partial M2M transformation from CM to PISM; PISM_2_PSSM transformation is another partial M2M transformation from PISM to PSSM; and PSSM_2_SM transformation is a full model to text M2T transformation from PSSM to SM (Simulation Model: executable source code). A tool architecture and an Eclipse based prototype implementation for the MDD4MS framework are presented as well.

MDD approach brings great advantages to M&S. It provides ways to formally define the models and modeling languages. Since models are defined in a good manner and free of implementation details, conceptual modelers and domain experts can understand the models more easily and they can play a direct role in simulation model development process. The simulation model implementation becomes more efficient and error-free since repetitive code can be generated automatically. Most existing MDD research takes place with metamodeling and modeling environments. Although the research studies look very promising, especially model transformations have not been sufficiently studied yet in simulation field.

Based on our review of the current literature, we define the following open problems in the MDD applied M&S research:

- Using different diagramming techniques together to describe the simulation problem accurately and completely, including the dynamics of the simulation model.
- Defining combined M2M transformations for a set of source models.
- Adding component based approach in order to increase the portion of the auto-generated code.
- Adding library approach and search facilities in order to allow reusing already defined components (implemented building blocks).
- Developing guidelines and procedures for the validation of the generated simulation model.
- Evaluation of the model transformations and choosing the best model transformation.

We believe that these problems may form the basis for a research agenda in this field. As an early attempt to address the last problem we define a set of criteria for analyzing model transformations in the next section.

4 CRITERIA FOR MODEL TRANSFORMATIONS

This section provides the evaluation criteria for model transformations extending the work of Mens and Gorp (2005). The goal of model transformations is to automatically generate different views of a system from a source model and to support the code generation. Both the target model and the transformation rules can be evaluated according to some basic principles derived from software engineering.

- *Correctness*: The correctness of a model transformation is analyzed in two ways: syntactic and semantic correctness (Mens and Gorp 2005). If the target model conforms to the target metamodel specification, then the model transformation is syntactically correct. If the model transformation preserves the behavior of the source model, then it is semantically correct (Ehrig and Ermel 2008).

- *Completeness*: In order to preserve and reuse the information in the source model, the transformation needs to be complete. For each element in the source model, if there is a corresponding element in the target model then the model transformation is complete (Mens and Gorp 2005).
- *Uniqueness*: If the model transformation generates unique target models for each source model, then it provides uniqueness.
- *Termination*: If the model transformation always terminates and leads to a result, then it provides termination.
- *Readability*: If the transformation rules are human-readable and understandable, then the model transformation provides readability.
- *Efficiency*: The model transformations need to be performed efficiently. The efficiency of a transformation can be evaluated by analyzing how many transformation steps are necessary, and how many functions are applied during each specific transformation.
- *Maintainability*: The degree of the effort spent for changing, extending and reapplying a model transformation defines the maintainability.
- *Scalability*: The ability to cope with large models without sacrificing performance defines the scalability (Mens and Gorp 2005).
- *Usability*: Once implemented, if the model transformation can be performed easily then it is usable.
- *Reusability*: Reusability can be measured with the possibility of adapting and reusing a model transformation via various reuse mechanisms such as parameterization or using templates (Mens and Gorp 2005).
- *Accuracy*: If all possible errors are handled and the model transformation can manage with the all incomplete source models, then it provides accuracy.
- *Robustness*: If most of the unexpected errors can be handled and the model transformation can manage with the all invalid source models, then it provides robustness. The difference from the accuracy is, an incomplete source model can be viewed by the associated modeling tool, whereas an invalid source model cannot be viewed by the associated modeling tool.
- *Modularity*: Modularity can be defined as the ability to compose existing transformations into new composite ones. Decomposition of a complex transformation into smaller steps may require a mechanism to specify how these smaller transformations need to be combined (Sendall and Kozaczynski 2003).
- *Validity*: Since transformations can be considered as a special kind of software programs, systematic testing and validation techniques can be applied to transformations to ensure that they have the desired behavior (Mens and Gorp 2005).
- *Consistency*: Model transformations need to be consistent and unambiguous. If the model transformation detects and possibly resolves the internal contradictions and inconsistencies, then it is consistent (Mens and Gorp 2005).
- *Parsimony*: A target model needs to be as parsimonious and lightweight as possible. Simplicity and lack of redundancy can define the level of parsimony.
- *Traceability*: Traceability is the property of having a record of links between the source and target elements as well as the various stages of the transformation process. Traceability links can be stored either in the target model or separately (Dehayni et al. 2009).
- *Reversibility*: A model transformation from S to T is reversible if there is a model transformation from T to S. This property can be useful in canceling the effects of a transformation (Dehayni et al. 2009).

As stated earlier, there are many ways to define a model transformation for a source model. There is no general rule or guidance that can be provided to define a good solution. However, based on the defined criteria, model transformations can be analyzed and results can be used to choose the best model transformation.

5 CONCLUSION

This paper presented an overview of MDD and related research in simulation field. The key elements, open problems and the evaluation criteria for a successful model transformation are explained. A MDD approach requires languages for the specification of models, for the specification of metamodels and for the definition of model transformations. For a successful code generation, it is important to select the most convenient languages and to use the efficient and scalable tools.

Some future directions can be listed as follows: The trade offs between the evaluation criteria need to be examined. A special care needs to be taken into account about code generation by the developers. Auto-generated and non-generated code should be clearly differentiated. Increasing the metamodel expressiveness by the use of additional model restrictions can help with better model transformations. As a future work, we will evaluate some model transformation examples based on the proposed evaluation criteria.

REFERENCES

Achilleos, A., N. Georgalas, and K. Yang. 2007. "An Open Source Domain-Specific Tools Framework to Support Model Driven Development of OSS". In *Proceedings of the 3rd European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2007)*, edited by D. H. Akehurst, R. Vogel, and R. F. Paige, Volume 4530 of *Lecture Notes in Computer Science*, 1–16: Springer.

Amyot, D., H. Farah, and J.-F. Roy. 2006. "Evaluation of Development Tools for Domain-Specific Modeling Languages". In *Proceedings of the 5th Workshop on System Analysis and Modeling: Language Profiles*, edited by R. Gotzhein and R. Reed, Volume 4320 of *Lecture Notes in Computer Science*, 183–197: Springer.

ATL 2007. "ATL Basic Examples and Patterns: Families to Persons - A simple illustration of model to model transformation". Accessed July 14, 2011. http://www.eclipse.org/m2m/atl/basicExamples_Patterns/. ATLAS Group, INRIA and University of Nantes, France.

Barton, R. R. 1998, December. "Simulation Metamodels". In *Proceedings of the 1998 Winter Simulation Conference*, edited by D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, 167–176. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Bézivin, J., G. Dupé, F. Jouault, G. Pitette, and J. Rougui. 2003. "First experiments with the ATL model transformation language: Transforming XSLT into XQuery". In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*.

Cetinkaya, D., A. Verbraeck, and M. D. Seck. 2010, December. "Applying a Model Driven Approach to Component Based Modeling and Simulation". In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 546–553. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Cetinkaya, D., A. Verbraeck, and M. D. Seck. 2011. "MDD4MS: A Model Driven Development Framework for Modeling and Simulation". In *Proceedings of the 2011 Summer Computer Simulation Conference (SCSC 2011)*. The Hague, Netherlands.

Czarnecki, K., and S. Helsen. 2003. "Classification of Model Transformation Approaches". In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*.

D'Ambrogio, A., D. Gianni, J. Risco-Martín, and A. Pieroni. 2010. "A MDA-based Approach for the Development of DEVS/SOA Simulations". In *Proceedings of the 2010 Spring Simulation Multiconference*. NY, USA: ACM.

De Lara, J., and H. Vangheluwe. 2002. "AToM3: A Tool for Multi-formalism and Meta-modelling". In *Proceedings of the Fundamental Approaches to Software Engineering*, edited by R.-D. Kutsche and H. Weber, Volume 2306 of *Lecture Notes in Computer Science*, 174–188: Springer.

Dehayni, M., K. Barbar, A. Awada, and M. Smaili. 2009. "Some Model Transformation Approaches: a Qualitative Critical Review". *Journal of Applied Sciences Research* 5 (11): 1957–1965.

Duarte, J. N., and J. de Lara. 2009. “ODiM: A Model-Driven Approach to Agent Based Simulation”. In *Proceedings of the 23rd European Conference on Modelling and Simulation (ECMS 2009)*.

Ehrig, H., and C. Ermel. 2008. “Semantical Correctness and Completeness of Model Transformations Using Graph and Rule Transformation”. In *Proceedings of the 4th International Conference on Graph Transformations (ICGT’08)*, edited by H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, Volume 5214 of *Lecture Notes in Computer Science*, 194–210: Springer.

EMF 2011. “Eclipse Modeling Framework Project”. Accessed July 14, 2011. http://www.eclipse.org/modeling/emf/. The Eclipse Foundation.

Guiffard, E., D. Kadi, J.-P. Mochet, and R. Mauget. 2006. “CAPSULE: Application of the MDA methodology to the simulation domain”. In *SISO EURO SIW*.

Harel, D., and B. Rumpe. 2004. “Meaningful Modeling: What’s the Semantics of ”Semantics”?”. *IEEE Computer* 37 (10): 64–72.

Huber, P. 2008. “The Model Transformation Language Jungle - An Evaluation and Extension of Existing Approaches”. Master’s thesis, Business Informatics Group, TU Wien.

Iba, T., Y. Matsuzawa, and N. Aoyama. 2004. “From Conceptual Models to Simulation Models: Model Driven Development of Agent-Based Simulations”. In *Proceedings of the 9th Workshop on Economics and Heterogeneous Interacting Agents*.

IRISA 2011. “Kermeta Workbench”. Accessed July 14, 2011. http://www.kermeta.org/. Triskell Project Team of IRISA (Institut de Recherche en Informatique et Systemes Aleatoires).

ISIS 2011. “Model Integrated Computing (MIC)”. Accessed July 14, 2011. http://www.isis.vanderbilt.edu/research/MIC. Institute for Software Integrated Systems, Vanderbilt University.

ISO/IEC 2005. “ISO/IEC 19502:2005, International Standard: Information technology - Meta Object Facility (MOF)”.

Mens, T., and P. V. Gorp. 2005. “A Taxonomy of Model Transformation”. In *Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005)*, Volume 152 of *Electronic Notes in Theoretical Computer Science*, 125–142.

Mészáros, T., T. Levendovszky, and G. Mezei. 2009. “Code Generation with the Model Transformation of Visual Behavior Models”. In *Proceedings of the 3rd International Workshop on Multi-Paradigm Modeling (MPM 2009)*, Volume 21 of *Electronic Communications of the EASST*.

OMG 2003. “Model Driven Architecture (MDA) Guide Version 1.0.1”. Accessed July 14, 2011. http://www.omg.org/mda/specs.htm.

OMG 2006. “Meta Object Facility (MOF) Core Specification, Version 2.0”. Accessed July 14, 2011. http://www.omg.org/spec/MOF/.

OMG 2011. “Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT)”. Accessed July 14, 2011. http://www.omg.org/spec/QVT/.

Scheidgen, M. 2006. “Model Patterns for Model Transformations in Model Driven Development”. In *Proceedings of the 4th Workshop on Model-Based Development of Computer-Based Systems*.

Sendall, S., and W. Kozaczynski. 2003. “Model Transformation: The Heart and Soul of Model-driven Software Development”. *IEEE Software* 20 (5): 42–45.

Sprinkle, J., B. Rumpe, H. Vangheluwe, and G. Karsai. 2007. “Metamodelling: State of the Art and Research Challenges”. In *Proceedings of the 2007 International Dagstuhl Conference on Model-based engineering of embedded real-time systems*, Volume 6100 of *Lecture Notes in Computer Science*, 57–76: Springer.

Topcu, O., M. Adak, and H. Oguztuzun. 2008. “A Metamodel for Federation Architectures”. *ACM Transactions on Modeling and Computer Simulation* 18 (3): 10:1–29.

Vangheluwe, H., and J. D. Lara. 2002, December. “Meta-Models are Models too”. In *Proceedings of the 2002 Winter Simulation Conference*, edited by E. Yücesan, C. H. Chen, J. L. Snowdon, and J. M. Charnes, 597 – 605. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wang, Y., A. Albani, and J. Barjis. 2011. “Transformation of DEMO Metamodel into XML Schema”. In *Proceedings of the First Enterprise Engineering Working Conference (EEWC 2011)*, Volume 79 of *Lecture Notes in Business Information Processing*, 46–60. Antwerp, Belgium.

AUTHOR BIOGRAPHIES

DENIZ CETINKAYA is a Ph.D. student at Delft University of Technology. She is in the Systems Engineering Group of the Faculty of Technology, Policy and Management. She received her M.Sc. in Computer Engineering from the Middle East Technical University, Turkey in 2005. She received her B.Sc. with honors in Computer Engineering from the Hacettepe University, Turkey in 2002. Her research focuses on component based modeling and simulation. Her e-mail address is d.cetinkaya@tudelft.nl.

ALEXANDER VERBRAECK is a full professor in Systems and Simulation in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology, and a part-time full professor in supply chain management at the R.H. Smith School of Business of the University of Maryland. He is a specialist in discrete event simulation for real-time control of complex transportation systems and for modeling business systems. His e-mail address is a.verbraeck@tudelft.nl.