A Method for Developing Model to Text Transformations

Alessandro Tiso, Gianna Reggio, Maurizio Leotta, Filippo Ricca

Abstract:

In the field of business process development, model transformations play a key role, for example for moving from business process models to either code or inputs for simulation systems, as well as to convert models expressed with notation A into equivalent models expressed with notation B. In the literature, many cases of useful transformations of business process models can be found. However, in general each transformation has been developed in an ad-hoc fashion, at a quite low-level, and its quality is often neglected. To ensure the quality of the transformations is important to apply to them all the well-known software engineering principles and practices, from the requirements definition to the testing activities. For this reason, we propose a method, MeDMoT, for developing nontrivial Model to Text Transformations, which prescribes how to: (1) capture and specify the transformation requirements; (2) design the transformation, (3) implement the transformation and (4) test the transformation. The method has been applied in several case studies, including a transformation of UML business processes into inputs for an agent-based simulator.

Digital Object Identifier (DOI):

https://doi.org/10.1145/3167132.3167370

https://doi.org/10.1145/3167132.3167370

A Method for Developing Model to Text Transformations

Alessandro Tiso
Università di Genova
Genova, Italy
alessandro.tiso@unige.it

Gianna Reggio
Università di Genova
Genova, Italy
gianna.reggio@unige.it

Maurizio Leotta
Università di Genova
Genova, Italy
maurizio.leotta@unige.it

Filippo Ricca
Università di Genova
Genova, Italy
filippo.ricca@unige.it

ABSTRACT

In the field of business process development, model transformations play a key role, for example for moving from business process models to either code or inputs for simulation systems, as well as to convert models expressed with notation A into equivalent models expressed with notation B. In the literature, many cases of useful transformations of business process models can be found. However, in general each transformation has been developed in an ad-hoc fashion, at a quite low-level, and its quality is often neglected. To ensure the quality of the transformations is important to apply to them all the well-known software engineering principles and practices, from the requirements definition to the testing activities. For this reason, we propose a method, *MeDMoT*, for developing non-trivial Model to Text Transformations, which prescribes how to: (1) capture and specify the transformation requirements; (2) design the transformation, (3) implement the transformation and (4) test the transformation. The method has been applied in several case studies, including a transformation of UML business processes into inputs for an agent-based simulator.

CCS CONCEPTS

• **Software and its engineering → Unified Modeling Language (UML)**;

KEYWORDS

Model-driven, UML, Model to Text Transformations.

ACM Reference Format:
Alessandro Tiso, Gianna Reggio, Maurizio Leotta, and Filippo Ricca. 2018. A Method for Developing Model to Text Transformations. In *SAC 2018: Symposium on Applied Computing, April 9–13, 2018, Pau, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3167132.3167370

1 INTRODUCTION

In the field of business process development, model transformations play a key role, for example for moving from business process models to either code or inputs for simulation systems, as well as to convert models expressed with notation A into equivalent models expressed with notation B (e.g. in a formal notation suitable for verification). In the literature many cases of useful transformations of business process models can be found, e.g. [2]. However, in general each transformation has been developed in an ad-hoc fashion, at a quite low-level (e.g. in terms of rules mapping meta-classes into meta-classes), and assuring the quality of the transformation is generally not considered. To obtain transformations of good quality it is important to apply all the well-known software engineering principles and practices, from the requirements specification till the testing activities. Indeed, in the last years, software engineering principles have been applied to software development to make the software production more systematic and quantifiable, and hence more effective. For this reason, we believe that the same software engineering principles should be applied to the model transformation development. This means that we must take care of all the development phases such as: requirements definition, design, implementation and testing; and that there is a need of a (possibly integrated) set of methods/techniques/notations to support the developers during all those phases, and thus covering the whole development process.

Currently, to the best of our knowledge, in the literature most of the works about model transformation development deal with the implementation phase, and we can cite only few works on requirement and testing as [1]. Furthermore, almost all proposals deal with the development of Model to Model Transformations, being [1] an exception since it considers Model to Text Transformations.

In this paper we present a portion of a new method (*MeDMoT*) covering all the phases of the development of Model to Text Transformations, providing specific techniques and notations for each phases. We have pragmatically designed *MeDMoT* so that it should:

- be able to support the development of transformations of reasonable size, not only small toy examples;
- be quite lightweight, avoiding to force the transformation developers to use formal notations (difficult to learn and to use) and to produce huge and detailed documentation of the transformation;
- be able to support the development of any kind of Model to Text Transformations, not only from models to code written in some programming language, but, e.g. also from models to various textual artifacts required in the software systems development.
- encompass all the good principles and practices of software engineering, for example the importance of high-level abstract requirements, modularization, test definition before coding, etc.

SAC 2018, April 9–13, 2018, Pau, France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167370

MeDMoT considers Model to Text Transformations, and prescribes how to develop a transformation guiding to: (1) capture and specify the transformation requirements, and (2) design, (3) implement, and (4) test the transformation.

MeDMoT has reached a certain level of maturity, indeed, it has been used to develop non trivial M2TTs such as PSOM2F (see Sect. 4), and UML models of Desktop applications → Java (AutoMARS) [4]. AutoMARS is a model driven application generator developed following *MeDMoT* that starting from a UML model representing a detailed design of an application can generate complete (excluding the GUI) Java desktop applications built using up-to-date technologies (like the Spring framework, JPA with Hibernate and Maven). All the details of the application can be in the source model, including operation method behaviour and constraints like class invariants, preconditions and post-conditions of operations. Also, automated test suite (JUnit test suite) for the generated Java application can be written directly on the source model. The source model is checked against a set of well-formedness constraints, helping to avoid the most common errors. A user may then concentrate himself/herself on producing and checking input models, thus increasing the level of abstraction used in the development of software.

Due to space reasons, here we present only the part of *MeDMoT* concerning the design of a transformation. The others have been already presented in [4, 6–8].

In Sect. 2 we give a short overview of *MeDMoT*, whereas the transformation design is described in Sect. 3, and Sect. 4 presents a case study. The conclusions are in Sect. 5.

2 *MEDMOT* OVERVIEW

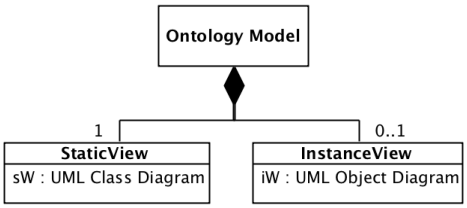
MeDMoT (**Me**thod for **D**eveloping **Mo**del **T**ransformations) aims to support the development of transformations from models to text, precisely it considers **M**odel to **T**ext **T**ransformations (shortly M2TTs) of the kind shown in Fig. 1 where:

– *Source Universe* is the set of models defined by a metamodel (e.g. the set of all the UML models), and – *Source* is a sub-set of *Source Universe* containing all models assumed to be correct inputs of the transformation (e.g. all UML models made by a class diagram and an activity diagram and a set of constraints).

– *Target Universe* is a set of **S**tructured **T**extual **A**rtifacts (shortly STAs) having a specific form, where an STA is a set of text files, written using one or more concrete syntaxes, disposed in a well-defined structure (physical positions of the files in the file system). – *Target* is the subset of *Target Universe* containing all the STAs assumed to be a correct result of the transformation.

We consider STAs instead of plain text, because in many cases the relative positions of text files in the file system have a specific meaning. For example, the position of configuration files and resource files in a Java enterprise application cannot be random but some prescribed file positions must be respected.

Figure 1: A generic Model to Text Transformation *mtt*



Let sW and iW be the two parts of an ontology model
– every class in sW must be stereotyped by either ≪category≫ or ≪instances≫
– every association in sW must have named ends and must be anonymous
.....

Figure 2: U-OWL Source Metamodel

– *mtt* is a function from *Source* into *Target*.

Both the elements in the source and in the target of the considered transformation have associated a semantics and the transformation developer should precisely know them. For example, if the source consists of UML models of Desktop applications and the target of Java Desktop applications, the developer must know which are the meanings of the elements composing the UML models belonging to the source, and of the Java constructs and of any API and frameworks that will be used.

Referring to Fig. 1, *MeDMoT* prescribes that a requirement specification for an M2TT is composed of: – the definition of the transformation domain *Source*; – the definition of the transformation codomain *Target*, and – the characterization of a relation (R) between domain and codomain.

An M2TT *mtt* will *respect the requirements* iff $s\ R\ mtt(s)$ for all $s \in Source$. The relation R should be expressed in terms of the semantics of the transformation domain and codomain, and not just relating metaclasses in the metamodel of the domain with strings appearing in the codomain.

The source of a transformation is a set of models conform to a metamodel constrained by a set of well-formedness rules, which precisely define the set of acceptable input models.

We show, as an example, the definition of the sources of U-OWL, that are UML extended by a profile models representing ontologies. The metamodel is shown in Fig. 2.

The UML profile is composed by two stereotypes: ≪category≫ and ≪instances≫. A class stereotyped by ≪category≫ represents a category of the ontology, whose instances will be defined in the InstancesView. A class stereotyped by ≪instances≫ defines simultaneously a category and its instances represented by its literals.

An ontology model is composed by a class diagram (the StaticView) defining the structure of the ontology in terms of categories, and possibly by an object diagram (the InstancesView) describing the information about the instances of the ontology.

The model transformation target is always a class of Structural Textual Artifacts with an associated semantics. A simple way to describe the target is defining its structure in terms of files and folders, and the concrete syntaxes relative to the various files composing it. The U-OWL target are the text files describing an ontology using the RDF/XML for OWL concrete syntax.

The design phase will be presented in detail in Sect. 3. For what concerns the implementation phase, we have selected some tools

and languages belonging to the Eclipse Modeling Project[1], offering a complete tools infrastructure for MDD (see [4]). Moreover, we have defined how to design and perform the testing phase that is very important for the quality of the transformation (see [7, 8]), using the requirement and the design specifications to generate the test cases.

In this paper, we will use U-OWL as a running example. U-OWL transforms UML models representing simple ontologies into the corresponding OWL ontologies, where the considered UML models of ontologies have been developed by some of the authors in the context of an industrial project.

3 TRANSFORMATION DESIGN

MeDMoT prescribes to structure the M2TTs to develop as a chain of transformations of different types, some from model to model, and the last from model to text, to help to modularize the transformation and to decompose the work of the developers in smaller and simpler tasks.

- **Well-Formedness Check** The input model is verified by a Model to Model Transformation returning a *diagnostic model* representing the violations of the well-formedness rules that constrain the transformation source.
- **Simplification** If the input model is well-formed, then it is simplified by one or more Model to Model Transformations performed in a well-defined order. The simplification has been introduced to facilitate the following task (see Sect. 3.1).
- **Text Generation** Finally, the simplified model is transformed into an STA using a M2TT (see Sect. 3.2).

3.1 Simplification

The simplification of the input models before transforming them into STAs is useful to reduce the complexity of the M2TT by a "divide et impera" approach.

In general, the modelling notations contain a lot of derived constructs and shortcuts, useful to improve the readability of the models and to facilitate their production, by means of the simplification they can be eliminated, and so their presence does not affect the production of the STAs. For example, a UML state machine having entry/exit and /internal transitions can be transformed into an equivalent one without those constructs, so if we have to transform state machines into Java code by a simplification we can eliminate such constructs, and thus the text generation will be simpler (i.e., less constructs to consider).

The simplification may be composed by one or more transformations, each of them will take care of simplify a single feature of the input models. These transformations must be executed in a well-defined order.

After having investigated different solutions we decided to present the *abstract design* of the simplifying transformations *by examples*, as well as the abstract design of the subsequent transformation for text generation. *MeDMoT* requires to give for each transformation an informal description and a set of examples of application, that we call transformation case or just cases. Each "*transformation case*" presents a generic example of application of the transformation. Technically, it is a pair, where the first component is an input model template, and the second is the result of the transformation applied on it. The abstract design of a simplifying transformation is then presented in the following way:

- Name: informal description

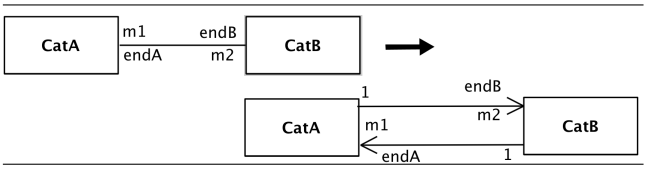
$$\text{source}_i \longrightarrow \text{target}_i$$

where source_i, target_i ($i = 1, \ldots, n$) are templates for source/target models, i.e. models where variables may appear; obviously such variables are implicitly typed by model elements, e.g. class name, list of attributes, and association end multiplicity in case of UML models.

The proposed method results in an *abstract design* of a transformation (named user design in [4]) that it is easy to understand, e.g. it does not require the knowledge of the source metamodel, and thus it can be understood also by people that are familiar with the notation used for the source models but not with its metamodel (as the majority of the users of the UML and of the BPMN). *MeDMoT* provides also a *detailed design* of the transformations (named developer design in [4]), where the source and the target are considered at the metamodel level.

In the U-OWL case there are two simplifying transformations: to remove the bidirectional associations between categories, and to expand the classes stereotyped by ≪instances≫ (reported in [5]). In the following, the RemoveBidirectionalAssociations transformation is depicted.

- **RemoveBidirectionalAssociations**: transforms a bidirectional association into two oriented associations. The multiplicities of the ending roles are preserved.



3.2 Design of Text Generation Transformation

MeDMoT proposes to proceed in the following way to design the final component of a M2TT, i.e. **T**ext **G**eneration **T**ransformation (shortly TGT) that is a Model to Text Transformation. First, for each possible input the output of the transformation must be designed, taking advantage of any existing techniques and methods available for the target (e.g. design patterns for Java programs), and trying to meet the requirements. Then, TGT must be designed.

The developer needs to "design" the output of the transformation for each possible input model, obviously taking into account all the requirements expressed before. Thus, the transformation developer should, as first step, *design* the target element result of the transformation of each input model; all methods, techniques, know-how relative to the target should be used at this point. For example, in the U-Java case everything relative to design Java applications should be considered, e.g. to use a three-tiered architecture or taking advantage of the Hibernate framework.

In the U-OWL case, for example, we decided that the categories of an ontology should be realized by OWL classes (using the OWL construct *owl:class*), and that a generalization relationship between

[1] www.eclipse.org/modeling/

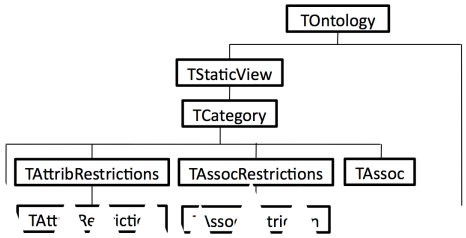


Figure 3: U-OWL: fragment of the TGT decomposition diagram

UML classes stereotyped ≪category≫ should be realized by the OWL construct *subClassOf* in RDF schema.

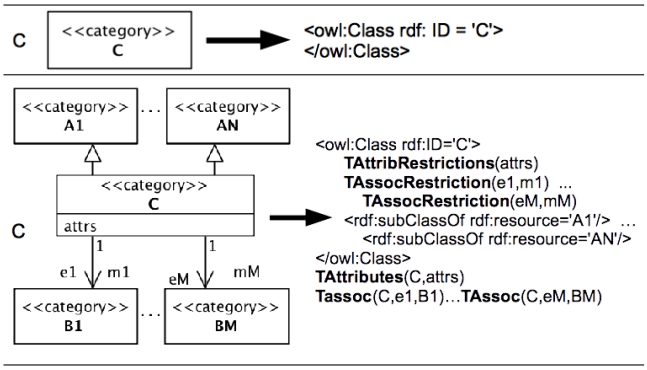
Since the TGT may be quite large and complex, it should be designed as composed by many sub-transformations, each of them taking care to transform parts of the input models. For example in the U-OWL case we have a transformation of the classes stereotyped by ≪category≫ (contained in the StaticView), and a transformation of the objects (contained in the InstancesView). The various sub-transformations may be arranged in a kind of call-tree, that we call functional *decomposition diagram*, where the nodes are labelled by the sub-transformations themselves and the children of a node labelled by S are the trees corresponding to those called by S. Fig. 3 shows a fragment of the decomposition diagram for the U-OWL case.

The design of each function composing the TGT is presented again by examples or better by transformation cases, as made before for the simplifying transformations in Sect. 3.1.

Each transformation case presents an example of use of the considered function. On the left of the arrow there are templates for the parameters, and on the right there is the result of the function applied to such templates expressed using the graphical symbols representing a folder or a file and the concrete syntaxes of the various files composing the target STA. As for the refactoring transformations, the left side may contain conditions on the variables appearing in the templates, whereas the right side may contain calls to other sub-transformations.

Here, we give the abstract design of a sub-transformation of the U-OWL case, the remaining ones can be found in [5].

- **TCategory** (String, ClassDiagram) transforms a category into OWL classes, data properties, object properties and the needed restrictions over them.



4 A CASE STUDY

In a joint project university-industry, we developed a service oriented method for modelling business process based on the UML [3], and worked out a way to validate the modelled processes by means of the agent-based simulation. The key point of the proposed approach is a transformation named PSOM2F from the service-oriented UML models of business processes into inputs for an agent-based simulation tool.

5 CONCLUSIONS

In this paper, we briefly sketched a portion of a method, *MeDMoT*, to support the Model to Text Transformations (M2TTs) from models to different kinds of textual artifacts (not only code). This method covers all the phases that have to be carried out during the development of non-trivial M2TTs, namely: capture and specification of the requirements, design, implementation and testing. Due to space reasons, in this paper we described only the design phase. *MeDMoT* encompasses classical software engineering techniques and principles to help produce transformations of good quality in an effective way.

MeDMoT can be used in the context of the business process development, as shown in Sect. 4, since the transformations from models to text are at the basis of various approaches supporting the production of high-quality business processes, for example, transformations allow to obtain inputs for validation and simulation tools or code from the business process models.

Summarizing, *MeDMoT* has the following features: (a) deals in integrated way with all the M2TT developing phases; (b) provides guidelines on how to define the requirements; (c) guides the design of the M2TT; (d) helps in modularize the design of the M2TT; (e) provides a lightweight notation to be used for the M2TT design specifications; (f) gives guideline on how to implement the design M2TT (suggesting also the tools and the IDE that can be used).

REFERENCES

[1] L. Burgueño and M. Wimmer. 2013. Testing M2T/T2M Transformations. In *Proceedings of 16th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems 2013 (MoDELS 2013)*. Springer, 203–219. https://doi.org/10.1007/978-3-642-41533-3_13

[2] A. Delgado, I. Garcia-Rodriguez de Guzman, F. Ruiz, and M. Piattini. 2010. From BPMN business process models to SoaML service models: A transformation-driven approach. In *Proceedings of 2nd International Conference on Software Technology and Engineering (ICSTE 2010)*, Vol. 1. V1–314–V1–319. https://doi.org/10.1109/ICSTE.2010.5608855

[3] Gianna Reggio, Maurizio Leotta, Diego Clerissi, and Filippo Ricca. 2017. Service-oriented Domain and Business Process Modelling. In *Proceedings of 32nd ACM/SIGAPP Symposium on Applied Computing (SAC 2017)*. ACM, 751–758. https://doi.org/10.1145/3019612.3019621

[4] Alessandro Tiso. 2014. *MeDMoT: a Method for Developing Model to Text Transformations*. Ph.D. Dissertation. University of Genoa, Italy.

[5] Alessandro Tiso and Gianna Reggio. 2017. U-OWL requirements and design specifications. http://sepl.dibris.unige.it/2017-MeDMot.php.

[6] Alessandro Tiso, Gianna Reggio, and Maurizio Leotta. 2012. Early Experiences on Model Transformation Testing. In *Proceedings of 1st Workshop on the Analysis of Model Transformations (AMT 2012)*. ACM, 15–20. https://doi.org/10.1145/2432497.2432501

[7] Alessandro Tiso, Gianna Reggio, and Maurizio Leotta. 2013. A Method for Testing Model to Text Transformations. In *Proceedings of 2nd Workshop on the Analysis of Model Transformations (AMT 2013)*, Vol. 1077. CEUR Workshop Proceedings. http://ceur-ws.org/Vol-1077/amt13_submission_10.pdf

[8] Alessandro Tiso, Gianna Reggio, and Maurizio Leotta. 2014. Unit Testing of Model to Text Transformations. In *Proceedings of 3rd Workshop on the Analysis of Model Transformations (AMT 2014)*, Vol. 1277. CEUR Workshop Proceedings, 14–23. http://ceur-ws.org/Vol-1277/2.pdf