Delivering the Next Generation of Model Transformation Languages and Tools

Vlad Acretoaie

Dept. of Applied Mathematics and Computer Science
Technical University of Denmark
rvac@dtu.dk

1 Introduction

The **research question** addressed by this thesis is the definition and implementation of a model transformation language focused on usability. Work on this topic has started in January 2013, as a continuation of an M.Sc. thesis on the topic of model querying [2].

The **procedure** used to address this question involves the proposal of a new by-example model transformation language based on the existing Visual Model Query Language (VMQL). The language will target research gaps identified by a Structured Literature Review currently in progress. Empirical validation of the new language's usability will play a central role in the project.

The **expected outcome** is a working implementation of the proposed model transformation language, accompanied by a corpus of empirical evidence to support its claims of superior usability.

2 Motivation

Creating and maintaining models as part of software development projects is currently a common practice. Models come into play in various development activities, such as domain analysis, solution architecture design, or implementation documentation. The importance of models in the software development process can vary. While some software development paradigms view models predominantly as communication artifacts, others such as Model Based and Model Driven Development (MB/MDD), Domain-Specific Languages (DSLs), and Business Process Management (BPM) place models in a more central role, sometimes with the intention of having models completely replace code.

Regardless of the importance they assign to models, all of these software development approaches require dedicated tools for model manipulation. An early attempt to answer this requirement has been proposed in the form of model operators [5] for matching, differencing, merging, splitting, and composing models. These were all conceived as high-level algebraic operators on data models. A more practically-oriented solution has been proposed by the MDSD community in the form of tools for model transformation, querying, constraint verification, version control and transformation. Out of these, model transformation is widely

regarded as the cornerstone of MDSD [23], as it facilitates a path from domain models to executable implementations. This philosophy is endorsed, for example, in the context of OMG's Model Driven Architecture (MDA) [20].

With these aspects in mind, we focus on the task of specifying and executing model transformations. We intend to address two application areas for model transformations: MDSD and domain or requirements analysis. A critical observation we make is that domain and requirements analysis models are often created by modelers with a non-technical background and limited programming experience. Such modelers require any model transformation tool to be first and foremost usable. Starting from previous work on VMQL, a by-example model query and constraint language [24, 25], we are currently developing the Visual Model Transformation Language (VMTL): a by-example model transformation language with usability at its core. We believe that our work will contribute to alleviating the problem of insufficient adoption of models as primary artifacts in the software development process [22].

3 Model Transformations

A model transformation takes as inputs one or more source models together with a transformation definition consisting of a set of rules. It produces one or more target models obtained by applying these rules to the source model(s). Models may be expressed in one of the standard modeling languages, e.g. UML [21] or BPMN [19], or in a Domain Specific Language (DSL). The components of a traditional model transformation are illustrated in Fig. 1, adapted from [7]. Note that the source and target meta models may or may not coincide (leading to *endogenous* or *exogenous* transformations, respectively [17]), but the transformation definition must explicitly refer to both meta models.

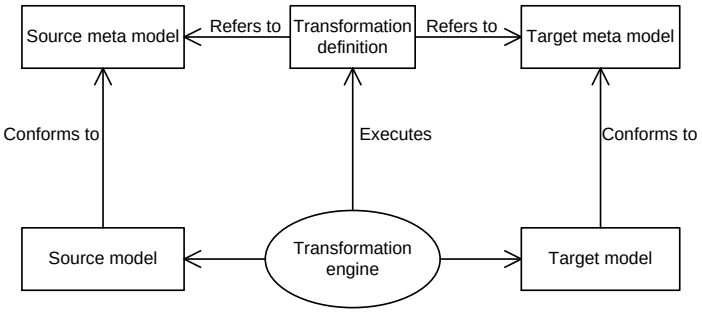


Fig. 1. Structure of a traditional model transformation

A considerable number of model transformation languages have been proposed so far (see [7, 17] for surveys). Some of them, such as the ATLAS Transformation Language (ATL, [11]), the Epsilon Transformation Language (ETL, [13]),

and Kermeta [18] benefit from relatively mature publicly available implementations. However, they all share a fundamental characteristic, in that they are in essence domain specific imperative programming languages with a textual syntax (though it should be noted that ATL and ETL also support declarative constructs). Furthermore, specifying transformations using these languages requires extensive knowledge of the modeling language's meta model.

Some model transformation languages are based on the theoretical foundations of graph transformation [9], including AGG [27], AToM^3 [8], FUJABA [31], GrGEN.NET [10], Henshin [3], VIATRA2 [29], and VMTS [16]. These are declarative languages, with various types of syntax: some are purely textual [10, 29], some introduce their own visual notation [27, 8, 31, 3], while some adapt the modeling language's abstract syntax visualized as UML Class Diagrams [16]. They specify transformation rules as pairs of left-hand side (LHS) and right-hand side (RHS) patterns, where each occurrence of the LHS pattern in the source model is replaced by the RHS pattern, thus obtaining the target model. From a usability perspective, specifying declarative patterns implies a softer learning curve than mastering a full-blown imperative programming language. However, modelers must still learn the syntax of the transformation language.

Using the concrete syntax of the source and target modeling languages to express LHS and RHS patterns can overcome this drawback. This avenue has been explored by the Patterns in Concrete Syntax (PICS, [4]) approach. Although it affords more compact and (according to the authors) more readable transformation specifications, PICS is only exemplified for refactoring UML Class Diagrams, and no tool implementation is mentioned. A similar approach originates in the Aspect Oriented Modeling (AOM) field, where MATA [32] is an aspect composition language based on graph transformation rules expressed in concrete syntax. MATA is also limited in scope, directly addressing only UML Class Diagrams, Sequence Diagrams, and State Machine Diagrams.

Recently, several approaches adopting the Model Transformation By-Example (MTBE, [28]) paradigm have been proposed [14]. Like PICS and MATA, these approaches use the concrete syntax of the source and target models to define transformation rules, and thus propose a change to the overall model transformation mechanism (see Fig. 2). Namely, the transformation definition directly references the source and target models. Modelers define transformations by specifying several source and target model pairs, from which the underlying transformation engine deduces correspondences between source and target model elements. These correspondences are then formalized as rules in an established model transformation language such as ATL. In *correspondence-based* MTBE approaches [15, 30], modelers explicitly specify the correspondences between source and target model elements via graphical model annotations. In *demonstration-based* MTBE approaches [6, 26], modelers exemplify the transformation rules by performing edit operations on the source model. By recording these operations or mapping model element IDs, the transformation engine deduces generalized transformation rules. In both correspondence and demonstration-based approaches, modelers can manually edit the generated transformation rules.

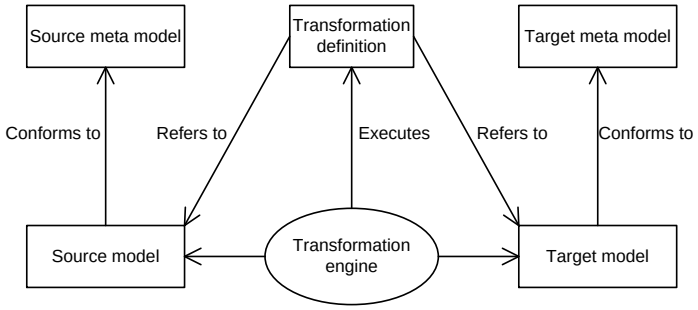


Fig. 2. Structure of a model transformation by-example

4 Outline of the Proposed Approach

Considering the overview of model transformation approaches presented in Section 3, we conclude that the recent MTBE direction presents considerable exploration opportunities. Furthermore, we envision our solution taking the demonstration-based direction, since it is suited to the vision and existing constructs of VMQL.

To exemplify the usage of VMTL, consider the *Pull Up Attribute* refactoring (i.e. endogenous transformation) illustrated in Figure 3 on a simple model depicting networking protocols. To trigger this refactoring, a model must satisfy the following preconditions:

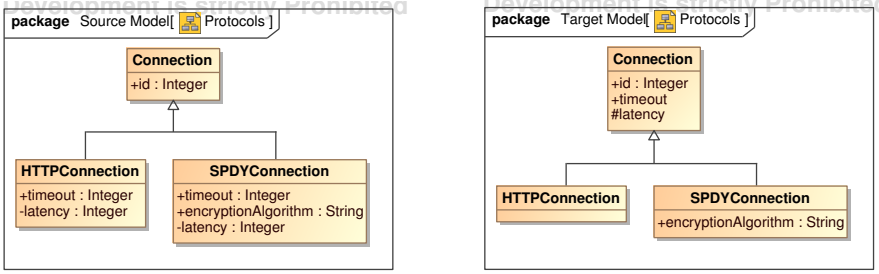
- *Pre1:* A superclass must have at least two distinct direct subclasses.
- *Pre2:* The subclasses must have one or more attributes sharing the same name, type, and visibility.
- *Pre3:* The superclass must not have an attribute with the same name as the one shared by the subclasses.

After the *Pull Up Attribute* refactoring is applied, the following postconditions must hold true:

- *Post1:* The common attribute is removed from the subclasses.
- *Post2:* The common attribute is added to the superclass.
- *Post3:* If the common attribute's visibility in the subclasses was *private*, its visibility is modified to *protected* in the superclass (so that it remains accessible to the subclasses). Otherwise, the attribute maintains its visibility.

All preconditions hold true for the source model in Figure 3a. Applying the described *Pull Up Attribute* refactoring to this model generates the target model in Figure 3b, for which all postconditions hold true.

A modeler would specify this refactoring using VMTL as shown in Figure 4. The refactoring definition contains two VMTL *rules*, each consisting of a *source pattern* and a *target pattern*. The semantics of applying VMTL rules is straight forward: each instances of a source pattern matched in the source model is



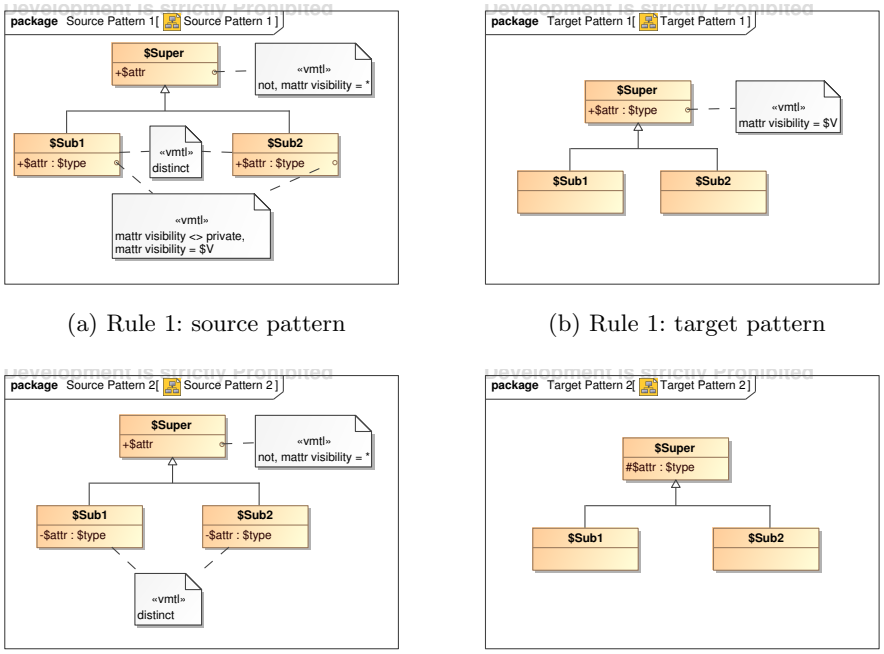
(a) Source model (b) Target model

Fig. 3. The *Pull Up Attribute* refactoring applied to a networking protocols scenario

replaced by an instance of the corresponding target pattern (considerations such as rule scheduling and rule application conditions are deferred for now). Rule 1 handles all scenarios except those in which the subclasses' common attribute is *private*, which are handled by Rule 2. We provide an explanation of Rule 1, whose source and target patterns are shown in Figure 4a and Figure 4b, respectively.

The source pattern in Figure 4a captures the listed preconditions. The pattern is itself a valid UML Class Diagram annotated with `<<vmtl>>` stereotyped comments. Each comment is anchored to a model element and contains a conjunction of VMTL *constraints* (the conjunction operator is a comma). Constraints alter the way in which patterns are matched in models. For instance, the `distinct` constraint anchored to classes `$Sub1` and `$Sub2` states that these classes will only be matched to distinct classes of the source model - therefore satisfying precondition *Pre1*. Strings starting with the `$` character are VMTL variables, and are bound to String values in the model at execution time. The `not, mattr visibility = *` constraint anchored to the `$attr` attribute of the `$Super` class states that, in order for the pattern to match, an attribute with this name must not exist in the matched class, regardless of its visibility (since an attribute with the same name exists in the two subclasses). Thus, precondition *Pre3* is satisfied. Finally, precondition *Pre2* is satisfied by the `$attr` and `$type` variables appearing in both subclasses (meaning that they must be bound to the same source model elements), in conjunction with the `mattr visibility <> private, mattr visibility = $V` constraint. This constraint states that the visibility of the attributes it is anchored to must not be *private*, and also stores the value of the visibility meta attribute in the variable `$V`. The same variable appears in the target pattern in Figure 4b as part of the `mattr visibility = $V` constraint, which assigns the value of the variable `$V` to the *visibility* meta attribute of attribute `$attr`. This means that the visibility of the pulled up attribute must not change. Note that all other variables appearing in the target pattern have also appeared in the source pattern, meaning that they are bound to the same values.

Since the transformation specification in Figure 4 introduces no new visual notation elements compared to the notation of the host modeling language (in



(a) Rule 1: source pattern

(b) Rule 1: target pattern

(c) Rule 2: source pattern

(d) Rule 2: target pattern

Fig. 4. VMTL rules describing the *Pull Up Attribute* refactoring

this case UML Class Diagrams), it can arguably be understood and created by any modeler. Furthermore, the expressiveness of VMTL is likely to exceed that of existing demonstration-based approaches, since none of these permits annotations to the source and target example models.

Figure 5 presents an overview of the proposed transformation process. A transformation takes as input a set of *source and target patterns*, where each source pattern has a corresponding target pattern. The patterns are processed by the *transformation generation engine*, which produces a *transformation definition* in a Prolog-based internal format. The purpose of this internal format is to enable the application of VMTL to several modeling languages, including but not limited to UML and BPMN. Apart from the rules themselves, the transformation definition stores additional information such as the rule scheduling policy. This definition is provided to a *transformation execution engine*, which also takes as input the *source model* (i.e. the model onto which the transformation definition will be applied). After converting the source model into a Prolog-based representation, the transformation execution engine produces the *target model* by matching instances of source patterns in the source model and replacing them with the corresponding instances of target patterns.

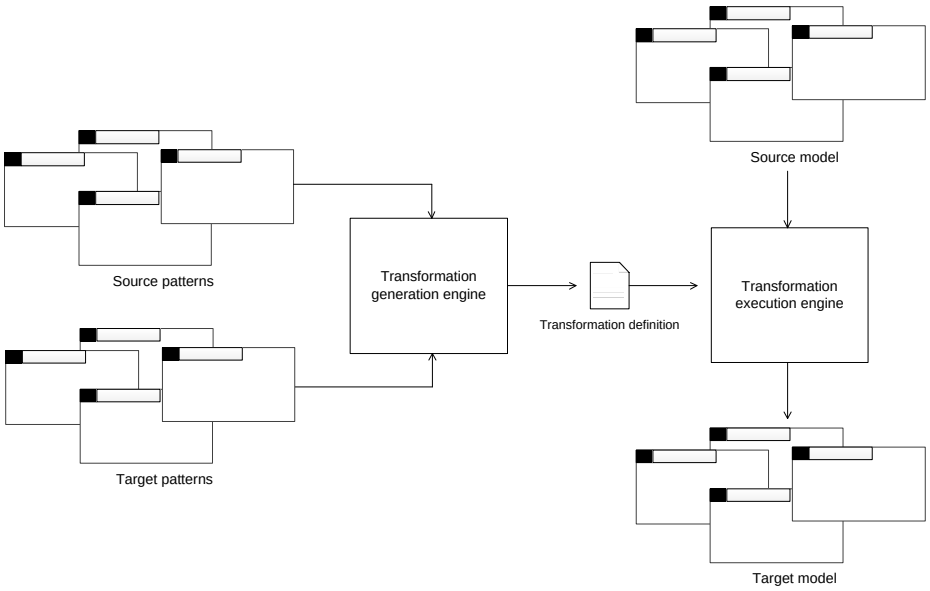


Fig. 5. An overview of the VMTL transformation process

5 Current Status and Next Steps

5.1 Current Status

As a starting point for the development of VMTL, we are in the process of conducting a Structured Literature Review (SLR) of the field of model transformation languages. Compared to a regular literature review, an SLR brings the additional benefit of a rigorous and extensively documented review process, making the results repeatable and thus increasing confidence in their validity [12]. The impetus behind this effort is twofold. First, we wish to provide a clear motivation for the direction chosen for VMTL. Second, the most recent comprehensive reviews of model transformation approaches date back to 2006 [7, 17], and were not performed under the rigorousness of an SLR.

A second line of current work is motivated by the desire to make VMTL applicable across modeling languages (i.e. not only to UML). To this extent, we are currently exploring the possibility of using VMQL as a query language for BPMN models. As VMQL lies at the basis of our future transformation language, this will provide a readily available path to applying VMTL to BPMN and other modeling languages.

5.2 Next Steps

In terms of future work required to bring the VMTL vision to fruition, we envision the following major development stages:

1. *Language definition.* We will extend VMQL with transformation-specific language constructs. As guidance for this process, we will maintain a set of transformation scenarios which the new language should be able to address.
2. *Implementation.* We will provide an implementation for VMTL. The implementation should provide support for usability-enhancing features such as stepwise execution and rollback of transformations, and should be integrated into an industrial strength modeling environment. We expect the implementation to be based on Prolog, similar to the implementation of VMQL [1].
3. *Validation.* We will determine via empirical studies if the usability advantage VMQL has demonstrated over OCL carries over to VMTL in relation to more established textual and visual model transformation languages.

5.3 Expected Challenges

Based on our experience with implementing VMQL and on the findings of our ongoing SLR effort, we expect to encounter the following challenges:

1. *Exogenous transformations support.* We expect to encounter some difficulties in handling exogenous transformations with a demonstration-based approach. The only existing solution addressing this challenge [15] relies on a state-based comparison between *context* and *extended* source and target models, respectively. The limited work in this area suggests that supporting exogenous transformations at the transformation language level may be a considerably more difficult task in the by-demonstration context.
2. *Necessity for an internal representation of transformation rules.* Existing MTBE approaches work by generating a transformation specification from the provided examples. This specification is internally represented using a traditional textual model transformation language, such as ATL. As a first step, we intend to adopt a simplified approach: recursively replacing instances of each source pattern with instances of the corresponding target pattern in the source model. This approach will be implemented by representing both patterns and source/target models as Prolog fact databases. Executing a transformation will then reduce to adding or removing facts from the source model to obtain the target model. It remains to be seen whether this execution model will suffice, or a more elaborate internal representation of transformation rules will be required. It may be the case that the *transformation generation engine* and the *transformation definition* mentioned in Figure 5 can be omitted. In any event, users will not be required to interact with, or even be aware of, the Prolog-based internal model representation.
3. *Execution times for large models.* Based on similar setbacks encountered with our existing VMQL implementation, we expect efficiency of the transformation engine to be a critical factor when dealing with large models. Overcoming such setbacks will largely rely on producing a Prolog implementation of an efficient model matching algorithm.

6 Conclusion

We have presented our roadmap to delivering a next generation model transformation language - VMTL. An ongoing survey of existing model transformation languages has suggested that usability is an often overlooked aspect in this area, which has prompted us to pursue a by-example approach to model transformations. The approach is based on our experience with by-example model queries and constraints. In this context, we have exemplified a usage scenario for VMTL. After outlining the planned implementation architecture, we have specified current work areas and future steps to be taken. Finally, we have listed what we believe to be the most important challenges lying ahead.

References

1. Acretoaie, V., Störrle, H.: MQ-2: A Tool for Prolog-based Model Querying. In: Workshop Proc. 8th European Conference on Modelling Foundations and Applications (ECMFA'12), pp. 328–331, 2012.
2. Acretoaie, V.: An implementation of VMQL. M.Sc. thesis, Technical University of Denmark, 2012.
3. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: advanced concepts and tools for in-place EMF model transformations. In: Proc. 13th International Conference on Model Driven Engineering Languages and Systems (MODELS'10), pp. 121–135. Springer (2008).
4. Baar, T., Whittle, J.: On the Usage of Concrete Syntax in Model Transformation Rules. In: Proc. 6th International Andrei Ershov Memorial Conference (PICS'06), pp. 84–97. Springer (2007).
5. Bernstein, P., Halevy, A., Pottinger, R.: A Vision for Management of Complex Models. In: SIGMOD Record 29(4), 55–63 (2000).
6. Brosch, P., Langer, P., Seidl, M., Wieland, K., Wimmer, M., Kappel, G., Retschitzegger, W., Schwinger, W.: An Example Is Worth a Thousand Words: Composite Operation Modeling By-Example. In: Proc. 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09), pp. 271–285. Springer (2009).
7. Czarnecki, C., Helsen, S.: Feature-based survey of model transformation approaches. In: IBM Syst. J. 45(3), 621–645 (2006).
8. de Lara, J., Vangheluwe, H.: AToM3: A Tool for Multi-formalism and Metamodelling. In: Proc. 5th International Conference FASE 2002, pp. 174–188. Springer (2002).
9. Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg G. (eds.): Handbook on Graph Grammars and Computing by Graph Transformation. In: Applications, Languages and Tools, vol. 2. World Scientific (1999).
10. Gelhausen, T., Derre, B., Geiss, R.: Customizing GrGen.NET for Model Transformation. In: Proc. 3rd International Workshop on Graph and Model Transformation (GraMoT'08). 2008.
11. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I: ATL: A model transformation tool. In: Science of Computer Programming 72, 31–39 (2008).
12. Kitchenham, B.: Procedures for Performing Systematic Reviews. Joint Technical Report, Keele University / NICTA, 2004.

13. Kolovos, D.S., Paige, R.F., Polack, F.A.: The Epsilon Transformation Language. In: Proc. 1st International Conference on Model Transformation (ICMT'08), pp. 46–60. Springer (2008).
14. Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., Wimmer, M.: Model transformation by-example: a survey of the first wave. In: Conceptual Modelling and Its Theoretical Foundations, 197-215 (2012).
15. Langer, P., Wimmer, M., Kappel, G.: Model-to-Model Transformations By Demonstration. In: Proc. 3rd International Conference on Model Transformation (ICMT'10), pp. 153–167. Springer (2010).
16. Levendovszky, T., Lengyel, L., Mezei, G., Charaf, H.: A Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS. In: Electron. Theor. Comp. Sci. 127(1), 65–75 (2005).
17. Mens, T., Van Gorp, P.: A Taxonomy of Model Transformation. In: Electronic Notes in Theoretical Computer Science (ENTCS) 152, 125–142 (2006).
18. Moha, N., Sen, S., Faucher, C., Barais, O., Jézéquel, J.M.: Evaluation of Kermeta for solving graph-based problems. In: Int. J. Software Tools for Technology Transfer (STTT) 12(3-4), 273–285 (2010).
19. The Object Management Group (OMG): Business Process Model and Notation (BPMN). Version 2.0, http://www.omg.org/spec/BPMN/2.0/. OMG (2011).
20. The Object Management Group (OMG): OMG MDA Guide. Version 1.0.1, http://www.omg.org/cgi-bin/doc?omg/03-06-01. OMG (2003).
21. The Object Management Group (OMG): Unified Modeling Language. Version 2.4.1, http://www.omg.org/spec/UML/2.4.1/. OMG (2011).
22. Selic, B.: What will it take? A view on adoption of model-based methods in practice. In: J. Software and Systems Modeling 11(4), 513–526 (2012).
23. Sendall, S., Kozaczynski, W.: Model Transformation - The Heart and Soul of Model-Driven Software Development. In: IEEE Software 20(5), 42–45 (2003).
24. Störrle, H.: VMQL: A Visual Language for Ad-Hoc Model Querying. In: J. Visual Languages and Computing 22(1), 3–29 (2011).
25. Störrle, H.: VMQL: Expressing Model Constraints Visually with VMQL. In: Proc. IEEE Symp. Visual Languages and Human-Centric Computing (VL/HCC'11), pp. 195–202. IEEE Computer Society (2011).
26. Sun, Y., White, J., Gray, J.: Model Transformation by Demonstration. In: Proc. 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09), pp. 712–726. Springer (2009).
27. Taentzer, G.: AGG: A Graph Transformation Environment for Modeling and Validation of Software. In: Proc. Second International Workshop AGTIVE 2003, pp. 446-453. Springer (2004).
28. Varró, D.: Model Transformation by Example. In: Proc. 9th International Conference on Model Driven Engineering Languages and Systems (MODELS'06), pp. 410–424. Springer (2006).
29. Varró, D.: The model transformation language of the VIATRA2 framework. In: Sci. Comp. Prog. 68(3), 187–2007 (2007).
30. Varró, D., Balogh, Z.: Model transformation by example using inductive logic programming. In: J. Software and Systems Modeling 8(3), 347–364 (2009).
31. Wagner, R.: Developing Model Transformations with Fujaba. In: Proc. 4th International Fujaba Days, pp. 79–82. University of Paderborn (2006).
32. Whittle, J., Jayaraman, P., Elkhodary, A., Moreira, A., Araújo, J.: MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation. In: Transactions on Aspect-Oriented Software Development VI - Special Issue on Aspects and Model-Driven Engineering, pp. 191-237. Springer (2009).