A Comparison of Taxonomies for Model Transformation Languages

Gabriel Tamura[1,2] and Anthony Cleve[2]

[1] University of Los Andes, TICSw Group, Cra. 1 N° 18A-10, Bogotá, Colombia
[2] INRIA Lille - Nord Europe, University of Lille 1, France
{gabriel.tamura,anthony.cleve}@inria.fr

Abstract. Since the introduction of the MDE/MDA/MDD ideas for software systems development several years ago, a number of different (meta)modeling and model transformation languages have been proposed. Although the OMG's QVT standard specification has somewhat stabilized the number of new model transformation languages, it is likely that new ones will continue to appear, following different paradigms and approaches. However, the evolution towards the consolidation of models as a unifying, foundational and consistent concept for software-intensive system development, requires the realization of a set of ideal characteristics for the specification of model transformation languages. Several works have been proposed for defining characterization and classification schemes, and the set of these ideal characteristics. In this paper we present a comparison of characterization and classification schemes for model transformation languages, and an analysis of the OMG's QVT specification with respect to this set of ideal characteristics. Although the MOF 2 QVT specification presents a good coverage of these characteristics, it still lacks a formal foundation, from which it could obtain several benefits.

1 Introduction

In MDE/MDA/MDD [17][25], software systems development is based on the concept of model. A model is defined as an abstract representation of a system, and, as an abstraction, it captures some characteristics of the system, while hiding others. Which characteristics are captured and which are hidden or omitted, depends on the focus or aspects in which the modeler is interested about the system. Therefore, for a given system, there can be many models, one capturing its visual interface, one for the constraints in its flow of control, another one for its security on operations, and others for capturing even nonfunctional aspects of it, such as safety or scalability, for instance. In turn, abstract representation of models, that is, a model of models, is called a meta-model.

Having a high level representation of one aspect of a system in independent models, allows modelers to better reason about that particular aspect of the system, so better designs, with better support for evolution and maintainability, would be expected.

However, in order for this approach of software system development to be feasible from the industrial point of view, the designed models have to be manipulated and operated in such a way that, from them, it becomes possible to safely derive or generate the source code and, finally, to obtain a running application of the system. These manipulations and operations are specified in model transformation languages.

Model transformations are thus, in the MDE/MDA/MDD approach, fundamental, because they make it possible, for example, on one side, to have a whole system representation, by merging its different models, and, on the other side, to organize the system models in several levels of abstraction. As proposed by the MDA: the top-layer captures the computational-independent model of a system, then the next layer captures the platform-independent model, and the platform-specific model is represented in the lowest layer. The ultimate objective is to derive the system source code by successive model transformations, from the top abstract representation, passing through the intermediate layered models. Models in each layer add successive design and implementation details, until the source code is produced.

The consolidation of the MOF specification [22] and the subsequent QVT RFP [27] by the OMG brought a high interest on the subject, and a considerable number of model transformation languages were created, with distinct approaches . The number of languages and the diversity of the approaches followed, made it necessary to analyze the nature of model transformation languages itself and, in consequence, several works on their characterization and classification were conducted, among them, reviews and recommendations, taxonomies and surveys, like [8], [16], [6] and [7].

The goal of this paper is twofold: (i) to compare the characterization and classification schemes proposed in the cited works. Even though they share some common characteristics, they differ from each other thereby following different organization and classification structures; and (ii) to compare the relative state of realization of characteristics identified as important or ideal on the MOF's QVT transformation language.

The remainder of this paper is organized as follows: in section 2 we compare the cited characterization and classification schemes for model transformation languages; section 3 presents an analysis of the OMG's QVT Specification in its 2005 and 2008 versions, with respect to the characteristics and classification schemes of the previous section; and section 4 concludes the paper and anticipates further work.

2 Characterization and Classification Schemes for Model Transformation Languages

Since the introduction of MDE/MDA/MDD, there have appeared many proposals of (meta)-modeling languages, like MOF (OMG) [19], ecore (Eclipse) [24], metaGME (GME) [3], HUTN [23] and EOL (Epsilon) [14]; and even more proposals for model transformation languages, like QVT [10], ATL [11], openArchi-

tectureWare [20], Kermeta [18], YATL (Yet Another Transformation Language) [21], VIATRA (VIsual Automated model TRAnsformations) [5], GReAT (Graph Rewriting and Transformation Language) [1], AGG (Attributed Graph Grammar System) [26] and Fujaba/TGG (Triple Graph Grammars [4].

This may resemble, at scale, the 1970's boom of programming languages era, in which a "Babel tower" was formed with a bunch of programming languages that, in many cases, appeared with no well-defined semantics or even not well fitted-for-purpose syntax, and then shortly disappeared.

To prevent, at least in part, a similar waste of effort in the development of model transformation languages, some initiatives, from the industry and the academic worlds, identified the importance of characterizing the nature of model transformation languages. As a result, some classification schemes and sets of characteristics and features have been proposed, differing in some organizational structures and characterization aspects.

In the following subsections we compare the proposals presented by:

1. The OMG MOF 2.0 QVT Submissions pre-review and recommendations towards the final standard [8], of the Zurich and Hursley IBM Research and Development Laboratories, which is based on a pre-review of the eight submissions presented in response to the OMG's QVT RFP [27]. The paper is presented under the practitioner and industrial-prospective perspective, and besides unifying a set of base terminology for queries, views, and transformations, makes a comparative analysis of the submissions. This analysis has as reference a set of common transformation scenarios of the industrial world, and the original QVT RFP requirements. The paper also highlights characteristics of each submission and gives a set of recommendations for the final QVT standard, so it does not address directly a classification scheme.
2. The Taxonomy of Model Transformations [15], which resulted from the second working group of the Dagstuhl Seminar on Language Engineering for Model-Driven Software Development in 2005, with the participation of more than 15 recognized authors on the subject. This is a prospective paper, that presents a more conceptual and academic perspective about model transformation approaches and languages, gives some base definitions, and addresses both the set of ideal or important characteristics of model transformation languages, as well as a classification scheme for model transformation approaches. The ideal characteristics are derived from a set of identified functional and non-functional requirements, whereas the classification scheme is based on the existing commonalities and variabilities of model transformation approaches, so it can be seen as a base for a taxonomy.
3. The Feature-based survey of model transformation approaches by Czarnecki and Helsen [6]. The perspective of this paper is as an inventory of features, rather than a prospective vision. Consequently, it presents an exhaustive classification scheme based on a feature model, that is, in terms of concrete trees of nodes, in which every node represents possible design choices for model transformations. It also presents the base definitions on the subject, analyzes the classification of several model transformation languages as well

as their highlights and drawbacks, and gives some directions for the applicability of the classification scheme. However, it does not directly address a set of ideal characteristics or decision choices.

4. The State-of-the-art on model-transformation based software development (in french) [7], of Diaw et al. The perspective of this paper is also of the inventory type, so it makes a compendium of the concepts of model-driven architecture and related approaches for system development. For the model transformation approaches, its classification scheme includes several dimensions or axes, and presents a characterization based on the observable properties of model transformations. The model transformation approaches are classified also in categories according to their realization, such as utilities, libraries or languages.

The contributions and approaches followed by each of them, as well as the corresponding structural organization of their work, are described as summaries in the next sections, identified as *sources*. These summaries, of course, are given under the subjective visions of the authors of the present paper.

2.1 Source 1: The OMG MOF 2.0 QVT submissions pre-review and recommendations.

The main reference for the analysis of characteristics presented in the IBM contribution in 2003, is the OMG's QVT RFP, in its mandatory and optional requirements. The mandatory requirements can be summarized as [27]:

- Transformation definitions shall describe relationships between a source MOF metamodel S, and a target MOF metamodel T, which can be used to generate a target model instance conforming to T from a source model instance conforming to S. The source and target metamodels may be the same.
- The abstract syntax for transformation, view and query definition languages shall be defined as MOF (version 2.0) [19] metamodels.
- The transformation definition language shall be declarative in order to support transformation execution in an incremental way, that is, changes in a source model may be transformed into changes in a target model immediately.
- The proposals should be implementation independent, address security issues where needed and specify the degrees of internationalization.

The optional requirements concerns the support for:

- Bi-directionality of transformation definitions (ideally single bi-directional definitions).
- Mechanisms for reuse and extension of generic transformation definitions.
- Traceability of transformation definitions/executions.
- Transactional definitions to specify which parts of a transformation definition are identified as suitable for commit or rollback during execution.

Based on the commonalities of the submissions and the reference set of transformation scenarios, the IBM contribution identified some additional requirements and proposed 12 recommendations. The additional requirements, which were supplemented with a benchmark for bidirectional transformations of Kent and Smith [13] are in sum:

- MOF meta-modeled definition for the declarative language for transformation definitions, i.e., a transformation specification should be a MOF model itself.
- Support for N-to-M transformation definitions.
- Inverse transformations should be definable such that $T^{-1}(T(M)) = M$.
- The use of additional transformation data, not contained in the source model, received for instance as parameters, should be possible.
- Conditions on rich well-formed mappings. Mapping definitions should facilitate the (automatic) construction (derivation) of tools, which, for instance, given source and target models and a mapping between them, decide if the source/target pair is a valid instance of the mapping.

On the other hand, the most divergent recommendations, with respect to the QVT RFP, can be summarized as:

- Support a hybrid language for transformation definitions.
- Provide a simple declarative specification language, and graphical (visual) as far as possible. In this, there should be a fair balance in expressiveness vs. brevity, to favor the easiness in construction, comprehension, and maintainability.
- For the queries, provide only *declarative* constructions.
- Support packaging, composition and reuse of transformation definitions, to specify more complex ones, and to build large transformations systems in a well organized and maintainable way.

2.2 Source 2: The Taxonomy of Model Transformations.

The contribution of the Dagstuhl Seminar on Language Engineering for Model-Driven Software Development in 2005 is structured around five fundamental questions:

1. What needs to be transformed into what?
2. What are the important characteristics of a model transformation?
3. What are the success criteria for a transformation language or tool?
4. What are the quality requirements for a transformation language or tool?
5. Which mechanisms can be used for model transformation?

For every question, the pertaining concepts, terms, base characteristics for classification, and ideal (or most promising) choices are given.

The classification scheme proposed by this source is based on the existing commonalities and variabilities of model transformation approaches. In short, the characteristics for its classification scheme are:

- Endogenous vs. exogenous: same or different domain metamodels;
- Horizontal vs. vertical: same or different levels of abstraction;
- Technological space of the source and target domains;
- Preservation: structure, behavior, correctness;
- Re-use schemes: generic, higher-order, groupings, composition, decomposition of transformations;
- Directionality and multiplicity: 1-to-1, M-to-N, mono or bidirectional;
- Declarative vs. operational.

For the set of ideal characteristics, or most promising options, the paper identifies a set of functional and non-functional requirements. They are summarized as follows:

The functional requirements:

- Create/read/update/delete transformations (CRUD);
- Applicability criteria: conditions to apply a given set of transformations;
- Customisation or reuse of transformations;
- Guarantee correctness of the transformations;
- Deal with incomplete or inconsistent models;
- Group, compose and decompose transformations;
- Ability to test, validate and verify transformations;
- Specify generic and higher-order transformations;
- Specify bidirectional transformations;
- Support for traceability and change propagation.

The non-functional requirements:

- Usability and usefulness: useful, in the sense that it has to serve a practical purpose; usable, in that it should be intuitive and efficient in its use.
- Verbosity versus conciseness, in the same sense as in the IBM contribution.
- Scalability.
- Standardization: conformance to other standards.

The set of characteristics identified as important:

- Automation: as far as possible, manual intervention in the overall process should be reduced to the minimum. This depends, naturally, on the kind of transformation.
- Mechanisms for scalability and re-use: generic, higher-order, groupings, composition, decomposition and inheritance of transformations.
- Verifiability, testability.
- Formal (mathematical) and sound foundation: this allows verification, enforcement and preservation of properties such as syntactic and semantic correctness in transformation definitions, but also in the automation of the transformation process itself. In this point, the paper favors declarative over operational approaches.

2.3 Source 3: The Feature-based survey of model transformation approaches

The paper of Czarnecki and Helsen, 2006, presents a wide coverage of model transformation systems and languages, with a thorough classification scheme based on a model of features [12], and on a set of so-called major categories for model transformation approaches.

The advantage of the model of features is its conciseness and concreteness for visually representing the design options and choices for a given domain, in this case, model transformation approaches.

The contribution of Czarnecki and Helsen is then structured over a top-level feature diagram, that corresponds to the first level of the hierarchy of their classification scheme. Then, for every major characteristic, the next level of sub-characteristics is developed. The top-level characteristics are:

- Specification: of pre- and post-conditions; executable, non-executable.
- Transformation Rules: domain languages, form and structure, application conditions, intermediate structures, parameterization, reflection, aspects.
- Rule Application Control: location of application (e.g. selection by pattern matching); scheduling.
- Rule Organization: packaging, modularization and re-use mechanisms.
- Source-Target (meta)models relationship.
- Incremental; directionality; tracing: in the same sense of the two other contributions analyzed in the previous sections.

With respect to the major categories for model transformation approaches, they distinguish between model-to-text and model-to-model transformations. The sub-categories for model-to-text, for which the output is source code, are the following strategies:

- Visitor-based
- Template-based
- Variations or combinations of the two above

And the sub-categories for model-to-model approaches, are the following:

- Direct manipulation: based on some internal representation for models, and a set of operations to manipulate it, in the form of APIs, for example.
- Structure-driven: based on transformation frameworks in which the designer only specifies the transformation rules, and the framework uses a strategy for its application.
- Operational: similar to the direct manipulation approach, but with more support for the transformations. In general, these approaches result from extending (meta)modeling tools and/or combining them with enriched programming languages.
- Template-based: use model template specifications with embedded metacode (e.g. OCL variations) to evaluate the variable parts of the template instances. Model template specifications are usually expressed in the concrete syntax of the target language, closely representing the result of the transformation.

- Relational: this category regroups declarative approaches based on mathematical relations, including executable and non-executable specifications. In general, the relational approaches are side-effect-free and, in contrast to the imperative direct manipulation approaches, create target elements implicitly. Relational approaches can naturally support multidirectional rules, and provides mechanisms for unification-based matching and rule application based on backtracking in its pure or domain-specific more efficient versions.
- Graph-transformation: based mainly on the theoretical work on graph rewriting over extensions or variations of typed-attributed labeled graphs, as formal representations of (meta)models. As noted also by the contribution of Dagstuhl, this approach has as advantages its graphical representation of rule transformations, which are close to the graphical representation of models, and the well-known mathematical properties already developed in the graph theory.
- Hybrid: this category groups approaches which result from the combination of two or more of the previous approaches. The combination can be done in a coarse-grained fashion, as separate components, or in a more fine-grained fashion, at the level of the transformation rules.

2.4 Source 4: The State-of-the-art on model-transformation based software development

The contribution of the Diaw et al. work is structured on characteristics of model-driven architecture and related approaches for system development, in a comprehensive way. In this contribution, two categories of characteristics can be identified: (i) conceptual aspects; and (ii) tool classification.

In the first category, conceptual aspects, it considers:

1. Major model transformation approaches: by direct programming, by templates and by models itself.
2. Transformation types: 1-to-1, M-to-N, in-place.
3. Transformation axis or dimensions: processes, metamodeling, parameterization.
4. Transformation properties: reversibility, traceability, re-usability, modularity (which ideally should be based on formal approaches).

All of these, with the exception of the transformation dimensions, are present in different categories of the contributions analyzed in previous sections, some of them possibly with different names. The difference in the transformation dimensions with respect to the other proposals, can be identified as:

1. Processes: by functionally identifying the transformation in the phases or global process of engineering. However, the sub-classification characteristics refers to the usual horizontal vs. vertical transformations.
2. Metamodeling: characterizes the role of metamodels in the transformations. For instance, metamodels can be used as a type system, or as a base for operations such as difference, merge, composition.

3. Parameterization: it is referred to the degree in which the transformation can be automated, on the presence of internal or external parameters.

For the taxonomy itself, the paper of Diaw et al. refers basically to a classification scheme based on the characteristics of endogenous-exogenous vs. horizontal-vertical.

In the second category, tool classification, it considers more pragmatic and implementation aspects, including the degree of coverage of the aforementioned conceptual aspects:

1. Generic tools: by family, for example, XML based and graph based.
2. IDE integrated tools: tools which offer a complete framework for for model development.
3. Specific model transformation languages/tools: usually to be integrated in other development frameworks.
4. Specific metamodeling tools: metamodeling tools in which the model transformation amounts to execution of a meta-program. In general, the definition of the metamodel can be made at the same level of the (meta)tools used to manipulate it.

3 Analysis of the QVT Standard as a Model Transformation Language

As can be evidenced by comparing the 2005 and 2008 OMG's QVT specifications [28][10] with the OMG's QVT RFP, several of the recommendations analyzed in the source contributions for the present paper, were adopted.

The main characteristics of the QVT specification are:

1. MOF-metamodeled abstract syntax, with textual and graphical concrete syntaxes.
2. Hybrid model-to-model transformation approach, distinguished from model-to-text mechanisms, for which there is another sub-standard.
3. Adoption of OCL, a purely declarative language, for queries.
4. Incremental and multi-directional.
5. Automatic management of traceability in transformations.
6. Pattern-matching by object template expressions in relations.

The hybrid model transformation language of the MOF 2.0 QVT specification [10] follows a combined coarse-grained and fine-grained approach, as it really proposes three well distinguished languages: core, relations, and operational. But in some parts of the operational language, it is possible to use combinations of the relational or the core languages. Besides this, it allows the use of "black-box" alternatives in these two languages.

The core and relations languages amount for the declarative part of QVT, and they serve as a basis to define the execution semantics of the imperative part.

3.1 The Declarative Part

The declarative part of QVT comprises the core and relations languages, each having different levels of abstraction, and thus, organized in a two layer architecture.

The core metamodel and language are defined by extending EMOF and OCL, forming a low-level consistent base, which follows the recommendations of self-definition properties. This combined base supports traceability in terms of explicit trace classes defined as MOF models, as well as automatic trace instance creation and deletion.

The relations metamodel and language specification conforms to a high-level relations language that supports object template creation and complex object pattern matching, in a similar way as supported in some functional object-oriented programming languages. As it is supported by the core language, automatic traceability is also supported by the relations language.

3.2 The Operational Part

MOF 2.0 QVT supports two mechanisms for using imperative styles of transformation: (i) the standard Operational Mappings language; and (ii) the non-standard Black-Box MOF Operation implementations. In QVT, each relation defines a class to be instantiated to trace between model elements being transformed, and which has a one-to-one mapping to an operation signature implementation.

In the operational mapping language, the model navigation must be done explicitly, as well as the creation of model elements. For that, in terms of the standard, it extends the declarative power with imperative constructs for flow of control, and with a version of OCL with side-effects.

3.3 Discussion

The MOF 2.0 QVT standard is the result of an evolution of several years, in which, directly or indirectly, the model-driven community has participated. Proof of this are the analyzed contributions themselves, and the changes that the standard has experimented since its original RFP.

In this evolution, which is representative of the evolution of the model transformation languages, the relation between theoretical computer science and software engineering has played a significative role. In the last years, there have been published many works on, for instance, the application of graph theory, graph grammars and graph rewriting to the specification of (meta)models and model transformation. In particular, this relation can be drawn from the analogy with the evolution of programming languages, whose specification started with intuitive approaches for structural and behavioral definitions. Then, a formal and sound theory of programming languages was developed, based on the Chomsky hierarchy [2]. This theory provided formal notations for the definition of any textual programming language, in terms of its lexical and syntactical rules of

formation, and mechanisms for defining its meaning, via operational or denotational semantics. Moreover, the equivalence of these notations with well known computational structures and their corresponding formal properties allowed a robust and safe automatic generation of considerable parts of the implementation of programming languages.

From the analysis that we have made in sections 2 and 3, we can observe that the MOF 2.0 QVT standard presents a good coverage of the characteristics that were identified as ideal or important by the contributions considered, but it still lacks a formal foundation for its structural definition and its semantics. As happened in the programming languages evolution, a formalism based on attributed grammars allowed not only the safe automation of program translation, but also to raise the level of reasoning about language design.

A second instance worth to discuss in this evolution, though not as fundamental as the former one, is related to the graphical vs. textual notation for transformation definitions. As noted by several works, like [9] and others, there are arguments for and against to the graphical vs. visual dichotomy. On one hand, there is a clear consensus about the advantages of the visual specification of (meta)model structures. This consensus is based mainly on two factors: (i) it is easier and faster for a designer to understand a visual representation of a model structure and the relationships among its parts; and (ii) there are automated tools that help designers to specify the kind of structures and the intra/inter-relationships among structural parts of models with minimum effort. On the other hand, there are the positions about the expressiveness of the textual notations, for which, in some cases, are said to be richer than the graphical ones. This is a controversial issue, which is related to the usability of the notation, and of course, to the level of training of the user. Nonetheless, it would appear that, having a graphical notation for (meta)model definitions, the natural option would be to have also a graphical notation for model transformation definitions.

Aside from the discussion, this dichotomy is resolved in the MOF 2.0 QVT standard by supporting the two options. The textual, represented in the operational and declarative textual languages, and for the graphical, introducing a graphical concrete syntax for the relations language.

4 Conclusions and Further Work

The big advances and the diversity of uses and applications that the MDE/MDA/MDD have shown in the last years that it is a promising system development approach, which is currently evolving towards a comprehensive, more structured and well-defined model-driven software engineering.

However, even though there have been several works on the "unification" of model driven concepts and definitions, the analysis made in this paper reveals that differences still remain in the use and meaning of the terms. Moreover, for the definition of model transformation languages, a key factor for the MDE/MDA/MDD success, there exists no general formal description language yet. And, since model transformation languages operate on models, it would be ideal to

have a formal language, technology independent, for defining also models. Having this formal language, a metalanguage in fact, it would then be possible to define model transformation languages in a more straightforward way.

OMG's MOF and QVT are claimed to be the de facto standards, for both the metamodeling language and for the model transformation language, and they satisfies and fulfills many of the requirements and important characteristics that have been identified in the contributions analyzed in the present paper. Nonetheless, they both still lack a formal foundation for their definition and semantics, a very important characteristic identified in at least three of the four contributions.

The existence of such a formalism would provide mechanisms to proof fundamental properties on the model transformation languages, such as correctness and confluence, and therefore, would enable a safer automation, of at least important parts of the transformation process. It could also serve as a sound basis for the realization of several other important identified characteristics, such as verification, testing, and re-use mechanisms in terms of higher-order definitions and other constructs. Besides this, having a common formal foundation for (meta)model definition and for model transformation definitions should allow to reason more consistently about the overall process of model transformation, contributing to consolidate the MDE/MDA/MDD as a solid, unifying and foundational approach to software development.

Acknowledgments

This work was carried out during the tenure of an ERCIM "*Alain Bensoussan*" Fellowship Programme by the second author.

References

1. Aditya Agrawal. Graph rewriting and transformation (great): A solution for the model integrated computing (mic) bottleneck. *Automated Software Engineering, International Conference on*, 0:364, 2003.
2. Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.
3. Jean Bézivin, Christian Brunette, Régis Chevrel, Frédéric Jouault, and Ivan Kurtev. Bridging the generic modeling environment (gme) and the eclipse modeling framework (emf). pages 1–9, 2005.
4. Sven Burmester, Holger Giese, Jörg Niere, Matthias Tichy, Jörg P. Wadsack, Robert Wagner, Lothar Wendehals, and Albert Zündorf. Tool integration at the meta-model level: the fujaba approach. *STTT*, 6(3):203–218, 2004.
5. György Csertán, Gábor Huszerl, István Majzik, Zsigmond Pap, András Pataricza, and Dániel Varró. Viatra " visual automated transformations for formal verification and validation of uml models. In *ASE '02: Proceedings of the 17th IEEE international conference on Automated software engineering*, page 267, Washington, DC, USA, 2002. IEEE Computer Society.

6. Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. In *IBM Systems Journal*, volume 45, pages 621–645. 2006.
7. Samba Diaw, Redouanne Lbath, and Bernard Coulette. Etat de l'art sur le développement logiciel basé sur les transformations de modèles. Technical report, University of Toulouse 2, 2009.
8. Tracy Gardner, Catherine Griffin, Jana Koehler, and Rainer Hauser. A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard, July 2003.
9. Roy Grønmo, Birger Møller-Pedersen, and Gøran K. Olsen. Comparison of three model transformation languages. In *ECMDA-FA '09: Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*, pages 2–17, Berlin, Heidelberg, 2009. Springer-Verlag.
10. The Object Management Group. Meta object facility (mof) 2.0 query/view/transformation. Specification Version 1.0, Object Management Group, April 2008.
11. Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. Atl: a qvt-like transformation language. In Peri L. Tarr and William R. Cook, editors, *OOPSLA Companion*, pages 719–720. ACM, 2006.
12. Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-oriented domain analysis (foda) - feasibility study. Technical report, The Software Engineering Institute, 1990.
13. Stuart Kent and Robert Smith. The bidirectional mapping problem. In *In Proceedings of the UNIGRA'03, Uniform Approaches to Graphical Process Specification Techniques (Satellite Event for ETAPS 2003)*, volume 82, pages 151–165. Electronic Notes in Theoretical Computer Science, June 2003.
14. Dimitrios Kolovos, Richard Paige, and Fiona Polack. The epsilon object language (eol). volume 4066, pages 128–142. 2006.
15. Tom Mens, Krzysztof Czarnecki, and Pieter Van Gorp. 04101 discussion – a taxonomy of model transformations. In Jean Bezivin and Reiko Heckel, editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
16. Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125 – 142, 2006. Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005).
17. Joaquin Miller and Jishnu Mukerji. *MDA Guide Version 1.0.1.* Object Management Group, Framingham, Massachusetts, June 2003.
18. Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In *in: International Conference on Model Driven Engineering Languages and Systems (MoDELS), LNCS 3713 (2005*, pages 264–278. Springer, 2005.
19. OMG. Meta object facility (mof) core specification version 2.0. Technical Report formal/06-01-01, January 2006. OMG Available Specification.
20. openArchitectureWare. openarchitectureware user guide version 4.3.1. pages 1–257, 2008.
21. Octavian Patrascoiu. Yatl: Yet another transformation language. In *University of Twente, the Nederlands*, pages 83–90, 2004.
22. John D. Poole. Model-driven architecture: Vision, standards and emerging technologies. In *ECOOP 2001, Workshop on Metamodeling and Adaptive Object Models*, April 2001.

23. Louis M. Rose, Richard F. Paige, Dimitrios S. Kolovos, and Fiona Polack. Constructing models with the human-usable textual notation. In Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Völter, editors, *MoDELS*, volume 5301 of *Lecture Notes in Computer Science*, pages 249–263. Springer, 2008.
24. Bernhard Schätz. Formalization and rule-based transformation of emf ecore-based models. pages 227–244, 2009.
25. Douglas C. Schmidt. Model driven engineering. pages 25–31. IEEE, 2006.
26. Gabriele Taentzer. AGG: A graph transformation environment for modeling and validation of software. In *Proc. Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 3062 of *Lecture Notes in Computer Science*, pages 446–453. Springer-Verlag, 2004.
27. The Object Management Group. Request for Proposal: MOF 2.0 Query / Views / Transformations RFP. pages 2–32, 2002.
28. The Object Management Group. MOF QVT Final Adopted Specification, 2005.