Towards a Taxonomy for Bidirectional Transformation

Romina Eramo, Romeo Marinelli, and Alfonso Pierantonio

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica (DISIM),
Università degli Studi dell'Aquila, Italy
name.surname@univaq.it

Abstract. In Model Driven Engineering, bidirectional transformations are considered a core ingredient for managing both the consistency and synchronization of two or more related models. However, current languages still lack of a common understanding of their semantic implications hampering their applicability in practice. This paper illustrates a set of relevant properties pertaining to bidirectional model transformations. It is a first step towards a taxonomy that can help developers to decide which bidirectional language or tool is best suited to their task at hand. This study is based on the existing literature and characteristics of existing approaches.

1 Introduction

Bidirectionality is an important feature of model transformations: often it is assumed that during development only the source model of a transformation undergoes modifications, however in practice it is necessary for developers to modify both the source and the target models of a transformation and propagate changes in both directions [31, 37]. In this context, bidirectional model transformations have arrived as a key mechanism, in fact they describe not only a forward transformation from a source model to a target model, but also a backward transformation that reflects the changes on the target model to the source model so that consistency between two models is maintained.

Many typical situations, that arise when both models are modified by humans, demand bidirectional transformations; for instance, a conceptual model is transformed into a platform specific model; a subview of selected data is modified and contextually the entire database is updated; many kinds of integration between systems or parts of systems, which are modeled separately, must be consistent (e.g., a database schema must be kept consistent with the application that uses it) [31]. Hence, bidirectional transformations have many potential applications in software development, including model synchronization, round-trip engineering, software evolution by keeping different models coherent to each other, multiple-view software development.

In this paper we propose a taxonomy for bidirectional model transformations. It represent a first step towards a complete set of objective criteria, that designers may consider in order to select the approach that is more suitable for their needs. The study is based on the discussions of the working group of the Dagstuhl seminars on Language Engineering for Model-Driven Software Development [6, 20] and the existing taxonomies for model transformations (e.g., [23, 7]).

The paper is organized as follows. Section 2 introduces the background. Section 3 proposes a taxonomy of bidirectional model transformations. Section 4 discusses how this taxonomy can be applied on existing approaches. Finally, Section 5 describes related work and Section 6 draws some conclusion and future work.

2 Background

Despite its relevance, bidirectionality has rarely produced anticipated benefits as demonstrated by the lack of a language comparable to what ATL[1] represents for unidirectional transformations. However, a number of languages and tools have been proposed due to the intrinsic complexity of bidirectionality; each one is characterized by a set of specific properties pertaining to a particular applicative domain [32].

The attempt to advocate a language rather than another can not neglect the important differences in circumstances. In fact, each language or tool presents specific characteristics that make it more suitable for a certain circumstance; as a consequence, a solution which seems good with one set of assumptions made about the situation can be infeasible with another. Understanding these characteristics is useful for distinguishing between existing approaches; in particular, in order to choose the most suitable approach, designers need to know how factors, that may influence their choice, affect bidirectional approaches.

Several works have been proposed for defining characterization and classification schemes of model transformation, others illustrate the state-of-the-art. Many of the proposed aspects characterizing model transformations are considered in this study, many other aspects need to be explored to shift the focus on intrinsic characteristics of bidirectional model transformations. Taking inspiration from [23], we consider some fundamental issues as following:

- What needs to be transformed into what;
- Which mechanisms can be used for bidirectional transformation;
- What are the application domains;
- What are the characteristics of a bidirectional transformation;
- What are the quality requirements for a bidirectional language or tool;
- What are the success criteria for a bidirectional language or tool.

These issues demand for the definition of pertaining concepts, terms and base characteristics for bidirectional transformation. To this end, in the next section a taxonomy for bidirectional transformation is proposed to provide specific and intrinsic features and requirements.

3 Taxonomy

The proposed taxonomy is based on a set of features that are divided into three main categories: General Requirements (GR), Functional Requirements (FR) and Non Functional Requirements (NFR). They are described in detail in the following.

[1] http://www.eclipse.org/atl/

3.1 General Requirements (GR)

This section defines general requirements that concerns generic and important aspects for bidirectional model transformations. These requirements are useful to understand what are some important characteristics of a bidirectional approach or tool.

Complexity. Transformations can be considered as small (e.g., refactoring), or heavy-duty (e.g., compilers and code generators) [23]. The large difference among them requires an entirely different set of tools and techniques. Considering the complexity of transformations is out of our scope; it demands for more effort and can be treated, for instance, by using metrics.
Level of automation. A bidirectional transformation can be performed in a completely automatic way (*fully-automated*), otherwise it may need to a certain amount of manual intervention (*human-in-the-loop*). In particular, manual intervention is needed to address and resolve ambiguity, incompleteness and inconsistency in the requirements, that may be partially expressed in natural language [23].
Visualization. It refers to the way in which models, metamodels and model transformations are presented to the user; it can be *graphical* or *textual*. Some existing tools allow developers to create artifacts in a completely graphical way (e.g., [28]), others require transformations are written entirely in textual way (e.g., [3]).
Level of industry application. Some existing tools are used in both academic and industrial world (e.g., [28]), others are used exclusively in academic setting (e.g., [5]), in which the usefulness is due to particular issues related to the bidirectionality.
Maturity level. The main existing approaches for bidirectional transformations are implemented by means a tool (*practical approaches*), often available on internet (e.g., [28, 26]). However, other approaches, despite the theoretical development (e.g., [9, 10]) have not been yet implemented by meas of a tool (*theoretical approaches*).

3.2 Functional Requirements (FR)

Functional requirements refer to the product capabilities and define how a bidirectional approach behaves to meet user needs. These requirements define important characteristics of a bidirectional approach that contribute to the success of such a language or tool. Furthermore, part of these requirements concerns the source and target artifacts of the transformations and the mechanisms that can be used.

Correctness. The simplest notion of correctness is the *syntactic correctness*: given a well-formed source model, it must guarantee that the target model produced by the transformation is wellformed. A significantly more complex notion is the *semantic correctness*: does the produced target model must have the expected semantic properties [23]. In other words, if the target model conforms to the target metamodel specification and wellformednes rules, then the model transformation is syntactically correct; whereas if the model transformation preserves the behavior of the source model, then it is semantically correct [11].
Inconsistency management. It concerns the ability to deal with incomplete or inconsistent artifacts. In fact, in the early phases of the software development life-cycle, requirements may not yet be fully understood; this often gives rise to ambiguous, incomplete

or inconsistent specifications, demanding for mechanisms for inconsistency management. These mechanisms may be used to detect inconsistencies in the transformation or in transformed models [23, 22, 24].

Modularity. It is the ability to compose existing transformations into new composite ones. Decomposing a complex transformation into small steps may require mechanisms to specify how these smaller transformations are combined [29, 4].

Traceability. It is the property of having a record of links between the source and target elements of a transformation as well as the various stages of the transformation process. Traceability links can be stored either in the target model or separately [29, 4, 8]. To support it, tools need to provide mechanisms to maintain an explicit link between the source and target models [23, 22, 24].

Change Propagation. Bidirectional approaches have to correctly propagate changes occurring in models from a direction to another. In order to support change propagation, bidirectional approaches may provide mechanism for an incremental updates or consistency management. Unfortunately, some approaches require to translate the source model into some standardized format (e.g., XML) before the transformation execution, and to translate again to obtain the target model. A clear disadvantage of such an approach concerns the difficult to synchronize models when changes are made [23, 22, 24]. Whereas, [35] propose a change propagating transformation language that supports the preservation of target changes by back propagating them toward the source. On the one hand, conflicts may arise each time the generated target should be merged with the existing one; on the other hand, the back propagation poses some problems related to the invertibility of transformations.

Incrementality. It refers to the ability to update existing target models on the base on changes made in the source models, and vice versa [7]. In particular, incremental transformations synchronize two models by propagating modifications such that information not covered by the transformation can be preserved and the computational effort can be minimized. In contrast, a classical batch transformation synchronizes two models taking a source model as input and computes the resulting target model from scratch, and vice versa [17]. This property can be achieved, for example, by using traceability links. When any of the source models are modified and the transformation is executed again, the necessary changes to the target are determined and applied. At the same time, the target elements may be preserved. In general, incremental transformations are able to avoid loss of information (e.g., elements discarded by the mapping).

Uniqueness. It refers to the number of solutions generated by the bidirectional approach [4]. If the transformation is non-deterministic, it may exist more than one way to keep models in a consistent way. Most of the existing tools generate a single solution. Some recent approaches are able to generate all the possible solutions according to a non-deterministic specification (e.g., [5]).

Termination. A model transformation provides termination, if it always terminates and leads to a result [4].

Symmetric/Asymmetric behavior. Transformations are asymmetric when they work between a concrete set of models and a strict abstraction of this, and the abstract set of models contains strict less information [31]. For instance, [15, 36] are asymmetric trans-

formations since two models that need to be synchronized must be one an abstraction of the other. On the contrary, the transformation is symmetric.

Type of Artifact. The distinction concerns the kinds of artifacts being transformed [23]. *Program transformation*s involve programs (i.e., source code, bytecode, or machine code); *Model transformation*s involve software models. According to [23], the latter term encompasses the former since a model can range from abstract analysis models, over more concrete design models, to very concrete models of source code. Hence, model transformations also include transformations from a more abstract to a more concrete model (e.g., from design to code) and vice versa (e.g., in a reverse engineering context). Model transformations are obviously needed in common tools such as code generators and parsers.

Data Model. It refers to the way in which data are represented into the tool. In particular, data can be represented by means of a graphs or trees.

Endogenous/Exogenous transformations. Models need to be expressed in some modeling language (e.g., UML for design models, or programming languages for source code models). The syntax and semantics of the modeling language itself is expressed by a metamodel (e.g., the UML metamodel). Based on the language in which the source and target models of a transformation are expressed, a distinction can be made between endogenous and exogenous transformations [23]. *Endogenous transformations* are transformations between models expressed in the same language. *Exogenous transformations* are transformations between models expressed using different languages.

Transformation Mechanisms. The major distinction between transformation mechanisms is whether they rely on a *declarative* or an *operational* (or *imperative*) approach. Declarative approaches focus on the what aspect, i.e., they focus on what needs to be transformed into what by defining a relation between the source and target models. Operational approaches focus on the how aspect, i.e., they focus on how the transformation itself needs to be performed by specifying the steps that are required to derive the target models from the source models. Declarative approaches (e.g., [1]) are attractive because particular services such as source model traversal, traceability management and automatic bidirectionality can be offered by an underlying reasoning engine.

There are several aspects that can be made implicit in a transformation language: (i) navigation of a source model, (ii) creation of target model and (iii) order of rule execution. As such, declarative transformations tend to be easier to write and understand by software engineers. Operational (or constructive) approaches (e.g., [30]) may be required to implement transformations for which declarative approaches fail to guarantee their services. Especially when the application order of a set of transformations needs to be controlled explicitly, an imperative approach is more appropriate thanks to its built-in notions of sequence, selection and iteration. Such explicit control may be required to implement transformations that reconcile source and target models after they were both heavily manipulated outside that transformation tool.

Declarative approaches include, but are not limited to, all of the following approaches: functional programming, logic programming and graph transformation.

- *Functional programming*. Such an approach towards model transformation is appealing, since any transformation can be regarded as a function that transforms some input (the source model) into some output (the target model). In most func-

tional languages, functions are first class, implying that transformations can be manipulated as models too. An important disadvantage of the functional approach is that it becomes awkward to maintain state during transformation.

- *Logic programming*. A logic language (e.g., Prolog or Mercury) has many features that are of direct interest for model transformation: backtracking, constraint propagation (in the case of constraint logic programming languages), and unification. Additionally, logic languages always offer a query mechanism, which means that no separate query language needs to be provided.
- *Graph transformation*. It is a set of techniques and associated formalisms that are directly applicable to model transformation [16]. It has many advantages. It is a visual notation: the source, target and the transformation itself can be expressed in a visual way. It offers mechanism to compose smaller transformations into more complex ones.

In-place/Out-of-place transformations. If the number of involved models is only one, the source and target model are the same and all changes are made *in-place*. Other endogenous transformations create model elements in one model based on properties of another model (regardless of the fact that both models conform to the same metamodel). Such transformations are called *out-place* [24]. Note that exogenous transformations are always out-place.

3.3 Non Functional Requirements (NFR)

Non functional requirements may be considered as the quality attributes for a bidirectional transformation.

Extensibility and Modifiability. The extensibility of a tool refers to the ease in which it can be extended with new features. The modifiability of an artifact refers the ability of a bidirectional transformation to be modified and adapted to provide different or additional features [25].

Usability and Utility. The language or tool should be useful, which means that it has to serve to a practical purpose. On the other hand, it has to be usable too, which means that it should be intuitive and efficient to use [23].

Scalability. It is the ability to cope with large and complex transformations or transformations of large and complex software models without sacrificing performance [23].

Robustness. It is the ability to manage invalid models. If unexpected errors are handled and invalid source models are managed, then the approach provides robustness [4].

Verbosity and Conciseness. *Conciseness* means that the transformation language should have as few syntactic constructs as possible. From a practical point of view, however, this often requires more work to specify complex transformations. Hence, the language should be more *verbose* by introducing extra syntactic sugar for frequently used syntactic constructs. It is always a difficult task to find the right balance between these two conflicting goals [23, 25].

Interoperability. It refers to the ability of a tool to integrate itself with other tools, used within the (model-driven) software engineering process [25]. Sometimes designers wish to use and interchange models among different modeling tools and modeling languages. A typical example is the translation of some tool-specific representation of UML models

into XMI (and vice versa), OMG's XML-based standard for model interchange. This facilitates exchanging UML models between different UML modeling tools [22].
Reference Platform (Standardization). It indicates whether the transform tool is compliant to all the relevant standards (e.g., XML, UML, MOF) [25]. Many tools can export and import models in a standard form, typically XML; an external tool can then take the exported model and transform it [29]. Most of them are implemented within EMF [5, 12, 13, 21]. Others adopt an algebraic approach [18, 3] and work on files or data strings, which are not directly integrated with the environment EMF.
Verifiability and validity. It concerns the ability to test, verify and validate models and transformations. Since transformations can be considered as a special kind of software programs, systematic testing and validation techniques can be applied to them to ensure that they have the desired behavior [23, 4]. Verification of (sets) of model transformations is needed to assure that they produce well-formed and correct models, and preserve (or improve) desirable properties such as (syntactical or semantical) correctness, consistency, and so on [22].

Existing approaches and tools involve testing (e.g., test case generation for model transformations), model checking, or analysis performed only on executing programs (e.g., run-time monitoring) [2]. In contrast, [2] and [14] propose an approach to analyze model transformation by means of a logic environment, able to verify if the transformation satisfy certain properties (correctness, well-formedness, non-determinism, etc.). Furthermore, existing functional approaches perform model transformation analysis [18, 27], in order to evaluate the transformation validity (generally, in terms of correctness) before its execution.

4 Applications

The proposed taxonomy can be applied for assessing characteristics of the existing bidirectional approaches. As already said, each language or tool provides different characteristics, and designers can choose the desired approach by using the set of evaluation criteria proposed in this taxonomy. The application of this taxonomy on the existing approaches aims to emphasize strengths and weaknesses.

The most common paradigms used in the existing approaches are the declarative and the functional paradigms. Among the existing declarative approaches, we are interested to consider: (i) TGGs [28], a bidirectional languages based on graph grammars; (ii) QVT-Relations [26], a declarative bidirectional language part of the QVT standard; (iii) JTL [5], a constraint-based bidirectional transformation language. Moreover, we are interested in considering some functional approaches, that are: (i) Lenses [3], an asymmetric bidirectional programming language between a concrete structure and a correspondent abstract view, (ii) GRound-Tram [18], a functional language-based modeling framework; (iii) BiFlux [27], an integrated modeling framework for developing bidirectional model transformations based on graph query language.

The proposed taxonomy represents a a preliminary work; in fact, the real benefit is represented by the result of the application over the existing work. The application of the selected features will highlight weaknesses and criticality of the bidirectional approaches .

5 Related Work

This work is essentially based on the existing works which propose taxonomies and characterizations for model transformation and bidirectionality.

Important aspects of bidirectional transformations have been discussed during the working group of the Dagstuhl seminars on Language Engineering for Model-Driven Software Development in 2005 and 2011 [6, 20]. In general, model transformations can be characterized by different orthogonal concerns (see [7] for a detailed classification). Czarnecki and Helsen in [7] present a survey of model transformation techniques with a particular emphasis on rule-based approaches such as those based on graph transformations. They mention directionality, but do not focus on it. In [23] the authors propose a taxonomy for model transformation, in particular they consider functional requirements that contribute to the success of the tool or language and non-functional requirements or quality requirements. In [24], the taxonomy is applied to graph transformations, in particular AGG [33] and Fujaba[2]. In [22] the same taxonomy is presented again, but emphasizing the importance of other concepts such as composition, interoperability and bidirectionality. [25] proposes a taxonomy based only on non-functional features, referring to languages and artifacts. [29] puts emphasis on standardization and languages for model transformation. [4] considers existing taxonomies and proposes a set of most important features. Finally, [34] performs a comparison among existing taxonomies.

The above mentioned works consider general model transformation. A specific taxonomy for bidirectional transformations has not yet been proposed, however there have been several works analyzing characteristics and semantic issues of bidirectional model transformations. Among them, in [32] the QVT-R bidirectional transformation language is illustrated and semantic issues and open questions about bidirectionality are discussed. [31] explores the landscape of bidirectional model transformations until the 2007. [19] proposes a survey on TGGs tools.

6 Conclusion

In Model Driven Engineering, bidirectional transformations are considered a core ingredient for managing both the consistency and synchronization of two or more related models. However, current languages still lack of a common understanding of its semantic implications hampering their applicability in practice. This paper proposed a set of relevant properties pertaining bidirectional model transformations. It is based on the existing literature and the characteristics of existing approaches. This work aims to represent a first step towards a taxonomy that can be used, among others, to help developers in deciding which bidirectional transformation language or tool is more suitable for different types of tasks. In order to do this, as future work, we plan to extend the taxonomy with other features for bidirectionality and apply it to existing languages and tools for bidirectionality.

[2] http://www.fujaba.de/

References

1. D. H. Akehurst and S. Kent. A Relational Approach to Defining Transformations in a Metamodel. In *Procs of the 5th Int. Conf. on The UML*, pages 243–258. Springer-Verlag, 2002.
2. K. Anastasakis, B. Bordbar, and J. M. Küster. Analysis of model transformations via Alloy. In *ModeVVa'07*, pages 47–56, 2007.
3. A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, and A. Schmitt. Boomerang: Resourceful lenses for string data. In *Proc. of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL 2008, pages 407–419, 2008. http://www.seas.upenn.edu/ harmony/.
4. D. Cetinkaya and A. Verbraeck. Metamodeling and Model Transformations in Modeling and Simulation. In *Proceedings of the Winter Simulation Conference*, WSC '11, pages 3048–3058, 2011.
5. A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. JTL: A Bidirectional and Change Propagating Transformation Language. In *3rd Int. Conf. on Software Lang. Engineering (SLE)*, SLE 2010. http://jtl.di.univaq.it.
6. K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective–GRACE meeting. In *ICMT2009*, volume 5563 of *LNCS*, pages 260–283. Springer, 2009.
7. K. Czarnecki and S. Helsen. Feature-based Survey of Model Transformation Approaches. *IBM Systems J.*, 45(3), 2006.
8. M. Dehayni, K. Barbar, A. Awada, and M. Smaili. Some model transformation approaches: a qualitative critical review. *Journal of Applied Sciences Research*, 5(11):1957–1965, 2009.
9. Z. Diskin, Y. Xiong, and K. Czarnecki. From state-based to delta-based bidirectional model transformation. In *3rd Int. Conf. on Model Transformation*. Springer, 2010.
10. Z. Diskin, Y. Xiong, and K. Czarnecki. From state- to delta-based bidirectional model transformations: the asymmetric case. In *Journal of Object Technology*, volume 10, 2011.
11. H. Ehrig and C. Ermel. Semantical correctness and completeness of model transformations using graph and rule transformation. In *ICGT*, pages 194–210, 2008.
12. EMoflon. http://www.moflon.org/.
13. EMorF. http://www.emorf.org.
14. R. Eramo, R. Marinelli, A. Pierantonio, and G. Rosa. Towards analysing non-determinism in bidirectional transformations. In *Procs. of AMT 2014*, 2014.
15. J. Foster, M. Greenwald, J. Moore, B. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3), 2007.
16. A. Gerber, M. Lawley, K. Raymond, J. Steel, and A.Wood. Transformation: The Missing Link of MDA. In *1st Int. Conf. on Graph Transformation*, pages 90–105, 2002.
17. H. Giese and R. Wagner. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling*, 8(1):21–43–43, 2009.
18. S. Hidaka, Z. Hu, K. Inaba, H. Kato, and K. Nakano. Groundtram: An integrated framework for developing well-behaved bidirectional model transformations. In *ASE 2011*, 2011. http://www.biglab.org/.
19. S. Hildebrandt, L. Lambers, H. Giese, J. Rieke, J. Greenyer, W. Schäfer, M. Lauder, A. Anjorin, and A. Schürr. A survey of triple graph grammar tools. In *BX 2013*, volume 57, pages 1–18. EC-EASST, 2013.
20. Z. Hu, A. Schürr, P. Stevens, and J. F. Terwilliger. Bidirectional transformation "bx" (dagstuhl seminar 11031). *Dagstuhl Reports*, 1(1):42–67, 2011.
21. MediniQVT. http://projects.ikv.de/qvt/.

22. T. Mens. Model transformation: A survey of the state-of-the-art. In S. Gerard, J.-P. Babau, and J. Champeau, editors, *Model Driven Engineering for Distributed Real-Time Embedded Systems*. Wiley - ISTE, 2010.
23. T. Mens, K. Czarnecki, and P. V. Gorp. A Taxonomy of Model Transformations. In *Language Engineering for Model-Driven Software Development*, volume 04101 of *Dagstuhl Seminar Proceedings*, 2004.
24. T. Mens, P. V. Gorp, D. Varro, and G. Karsai. Applying a model transformation taxonomy to graph transformation technology. *Electronic Notes in Theoretical Computer Science*, 152:143–159, 2006.
25. S. Nalchigar, R. Salay, and M. Chechik. Towards a Catalog of Non-Functional Requirements in Model Transformation Languages. In *AMT @ MoDELS*, 2013.
26. Object Management Group (OMG). MOF QVT Final Adopted Specification, 2005. OMG Adopted Specification ptc/05-11-01.
27. H. Pacheco and Z. Hu. Biflux: A Bidirectional Functional Update Language for XML. In *BIRS workshop: Bidirectional transformations (BX)*, 2013. http://www.prg.nii.ac.jp/projects/BiFluX/.
28. A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In *in Proc. of the 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG '94), Herrsching (D*. Springer, 1995.
29. S. Sendall and W. Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Softw.*, 20(5):42–45, 2003.
30. J. Sprinkle, A. Agrawal, T. Levendovszky, F. Shi, and G. Karsai. Domain Model Translation Using Graph Transformations. In *10th IEEE Int. Conf. and Workshop on the Engineering of Computer-Based Systems*, pages 159–168. IEEE Computer Society, 2003.
31. P. Stevens. A Landscape of Bidirectional Model Transformations. In R. Lämmel, J. Visser, and J. Saraiva, editors, *GTTSE 2007, Braga, Portugal*, volume 5235 of *Lecture Notes in Computer Science*, pages 408–424. Springer, 2008.
32. P. Stevens. Bidirectional Model Transformations in QVT: semantic issues and open questions. *Software and Systems Modeling*, 8, 2009.
33. G. Taentzer. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In *Int. Workshop on Applications of Graph Transformations with Industrial Relevance, AGTIVE 2003*, volume 3062 of *LNCS*, pages 446–453. Springer-Verlag, 2004.
34. G. Tamura and A. Cleve. A Comparison of Taxonomies for Model Transformation Languages. *Paradigma*, 4(1):1–14, 2010.
35. L. Tratt. A Change Propagating Model Transformation Language. Technical report, Department of Computer Science, King's College London, TR-06-07, 2006.
36. A. Wider. Towards Combinators for Bidirectional Model Transformations in Scala. In *Proc. of the 4th Int. Conf. on Software Language Engineering*, SLE 2011, pages 367–377. Springer-Verlag, 2012.
37. Y. Xiong, D. Liu, Z. Hu, H. Zhao, M. Takeichi, and H. Mei. Towards Automatic Model Synchronization from Model Transformations. In *ASE 2007*, pages 164–173, 2007.