Engineering Bidirectional Model Transformations (Short Paper)

Thomas Buchmann[1], Bernhard Westfechtel[1]

[1]*Applied Computer Science I, University of Bayreuth, D-95440 Bayreuth, Germany*

Abstract
Bidirectional transformations have been studied in a wide range of application domains. In model-driven software engineering, they are required for roundtrip engineering processes. We present a pragmatic approach to engineering bidirectional model transformations that assists transformation developers by domain-specific languages, frameworks, and code generators and provides for conciseness, expressiveness, and scalability. We also discuss different variants of transformation development processes as well as their advantages and drawbacks.

Keywords
Model-driven software engineering, roundtrip engineering, bidirectional transformation

1. Background

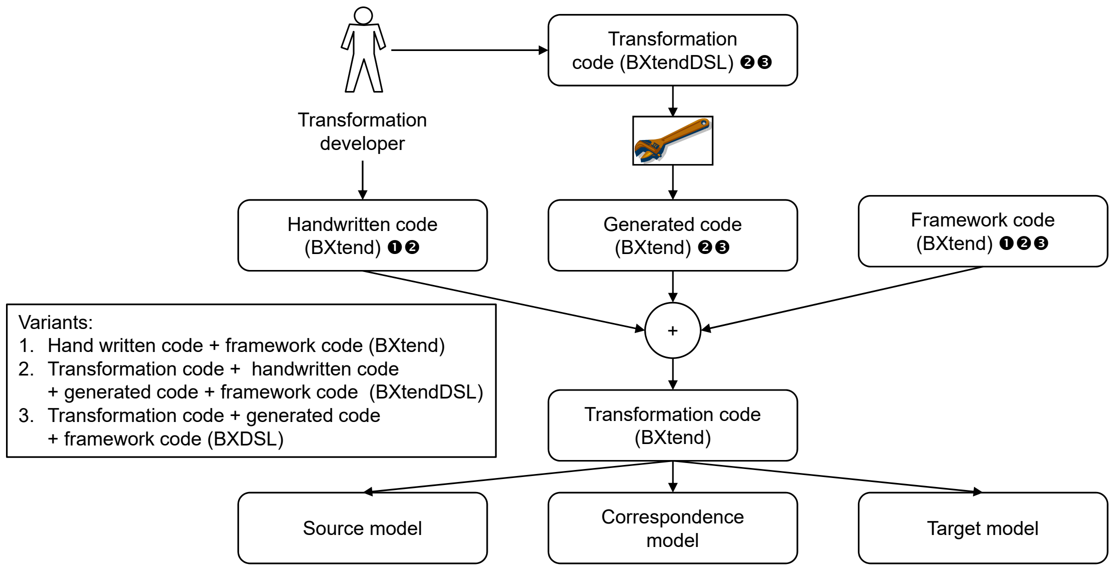
Bidirectional transformations (bx) occur in different application domains, including e.g. databases, programming languages, and software engineering [1]. Programming bidirectional transformations in a conventional programming language is both laborious and error-prone: Both transformation directions have to be programmed separately, and consistency of forward and backward transformations has to be checked by testing.

In response to these problems, a wide variety of bx approaches have been developed in research [2]. In *functional approaches*, a bidirectional transformation is defined by a function operating in one direction; the opposite direction is derived automatically [3, 4]. In *relational approaches*, a bidirectional transformation is defined by a set of relations between source and target elements [5, 6]. In *grammar-based approaches*, a set of grammar rules defines consistent pairs of source and target models [7, 8].

A recurring theme driving bx research are *roundtrip properties*, also referred to as *bx laws* [9]. Roundtrip properties are constraints on the interplay of forward and backward transformations. The goal of many bx approaches consists in the construction of bidirectional transformations that are *provably correct* with respect to roundtrip properties.

However, our *empirical evaluations* [8, 10, 11] demonstrate limitations of bx approaches with respect to *expressiveness*, i.e., the capability to solve a given bx problem. These limitations follow from the conditions that transformations have to satisfy in order to guarantee roundtrip properties. Additional shortcomings were observed with respect to *conciseness* — the ability

Tenth International Workshop on Bidirectional Transformations (BX 2022), July 08, 2022, Nantes, France
Thomas.Buchmann@uni-bayreuth.de (T. Buchmann); Bernhard.Westfechtel@uni-bayreuth.de (B. Westfechtel)



Transformation developer
Transformation code (BXtendDSL) ❷❸
Handwritten code (BXtend) ❶❷
Generated code (BXtend) ❷❸
Framework code (BXtend) ❶❷❸
+
Variants:
1. Hand written code + framework code (BXtend)
2. Transformation code + handwritten code + generated code + framework code (BXtendDSL)
3. Transformation code + generated code + framework code (BXDSL)
Transformation code (BXtend)
Source model
Correspondence model
Target model

Figure 1: Layered approach to bidirectional transformations

to provide for short solutions with respect to size metrics — and *scalability* — the ability to perform transformations efficiently on large data sets.

2. Contribution

Based on our experience gained from empirical evaluations, we have been developing *frameworks* for *engineering bidirectional model transformations* [12]. Our goal is to support transformation developers in the specification and implementation of bidirectional transformations in the context of *model-driven software engineering* [13]. Thus, the artifacts that are subject to transformations are *models* of software systems; the Eclipse Modeling Framework [14] serves as the underlying technological ecosystem. In particular, we focus on *roundtrip engineering* (e.g., between model and code), where the source model and the target model are considered as peers and updates may be propagated in both directions. While roundtrip properties are guaranteed to a certain extent, we have addressed primarily conciseness, expressiveness, and scalability.

In our work, bidirectional transformations share the following features: (1) Bidirectional transformations are *symmetric* inasmuch as source and target model are considered as peers (rather than asymmetric transformations, working on sources and views). (2) Each transformation execution is *directed*, i.e., it reads the source model and updates the target model (or vice versa). (3) Each transformation is *executed on demand* only (no live propagation of changes). (4) Transformations are *correspondence-based*, a correspondence model is stored persistently to allow for precise change propagations. (5) Finally, transformations are *state-based*, i.e., they rely on model states only rather than on operational deltas.

As illustrated in Figure 1, we provide a *layered approach* to developing bidirectional model transformations. This approach, which is called *BXTendDSL*, combines declarative with imperative programming and is labeled as number ❷ in Figure 1. Before, we briefly present its precursor *BXtend* [15], which entirely relies on *imperative programming* (number ❶). Finally, we discuss a potential *purely declarative approach* (*BXDSL*), which is subject to current and future work (number ❸).

2.1. BXtend

The acronym BXtend is composed from BX (bidirectional transformations) and Xtend, an object-oriented programming language that is based on Java [16]. The BXtend framework [15] provides an *internal domain-specific language (DSL)* for bidirectional transformations. An internal DSL offers an application programming interface in a host language; it is easier to implement than an external DSL and does not require the transformation developer to learn a new language. Xtend was selected as a host language because it offers high-level support for object-oriented, procedural, and functional programming.

Programming bidirectional transformations from scratch is laborious. BXtend reduces this effort considerably by providing a generic framework on top of which specific code for implementing transformation rules has to be written. The framework includes an implementation of a correspondence model as well as algorithms for incremental model transformations in both directions. The evaluations which we conducted so far confirm conciseness, expressiveness, and scalability. In the Families to Persons benchmark published in [8], the BXtend solution is the only one that passes all test cases. Furthermore, BXtend proves to be scalable. Even conciseness is reasonable although both transformation directions have to be programmed manually.

2.2. BXtendDSL

The BXtend framework shows two shortcomings: First, a transformation definition written in the BXtend internal DSL contains redundant code, i.e., similar code fragments in forward and backward direction. Second, roundtrip properties may be ensured by testing only. In response to these problems, we developed BXtendDSL [12, 17] that adds a declarative layer on top of the imperative layer. At the declarative layer, the transformation developer specifies the transformation in a purely declarative language (reflected in the suffix of the acronym); here, we decided to provide an *external DSL* with a short and intuitive syntax. After code generation, the transformation developer employs the internal DSL to complete the transformation definition.

The external DSL is a small and light-weight *relational language* that is based on rules describing relations between corresponding source and target patterns. Intentionally, the DSL is *computationally incomplete*: In all of the transformation cases that we have studied so far, the declarative code needs to be supplemented with imperative code that offers the required flexibility to solve the transformation case at hand. In this way, the external DSL can be kept small, avoiding the reimplementation of functionality that is available in the BXtend internal DSL anyway. Accordingly, *code generation* is *partial*, as e.g. in the EMF code generator [14], which generates incomplete code from a structural metamodel.

In the external DSL, a transformation is defined by a sequence of *rules* that may *depend* on each other. The DSL allows to declare $m : n$ *dependencies* between *source and target objects*, as well as $m : n$ dependencies at the level of *structural features* (attributes and references).

Furthermore, the transformation developer may define *extension points* that have to be filled with imperative code if the mapping cannot be handled by generated code alone.

Furthermore, the external DSL guarantees *roundtrip properties* under certain restrictions. Roughly following [18], *correctness* means that a transformation restores consistency among the participating models, and *hippocraticness* implies that models that are already mutually consistent are not updated. Correctness and hippocraticness are guaranteed under *well-behavedness conditions* that are defined by means of OCL [19] constraints on transformation definitions in the declarative DSL.

Our benchmark evaluations [12] reconfirm conciseness, expressiveness, and scalability. Compared to BXtend, both expressiveness and scalability are not affected adversely. Furthermore, solutions in the layered framework (comprising the manual code written on both layers) are considerably more concise than the corresponding BXtend solutions (and solutions in other tools/languages).

2.3. BXDSL

BXtendDSL offers two main advantages over BXtend: conciseness and guarantee of roundtrip properties (under well-behavedness conditions). However, it exhibits two shortcomings: First, the transformation developer has to switch between different levels of abstraction. Second, the well-behavedness conditions are restrictive; each of the transformation cases we have studied so far violates at least one of these conditions, implying the need for imperative programming. Thus, the question arises whether it is possible to design a declarative language (with the fictitious name *BXDSL*) that is computationally complete (and thus does not require imperative code as a supplement) and relaxes the well-behavedness conditions such that a wider range of transformations can be specified that are provably correct.

So far, we have not achieved these goals. Currently, we are working on an extension of BXtendDSL so that more work can get done at the declarative layer [20]. However, the new language version is not powerful enough to solve bx transformations completely, and still requires complementary imperative code, thus retaining the layered approach described above. Based on the experiences we have gained so far in bidirectional transformations, we consider it unlikely that a single bx language may be designed that guarantees round-trip properties without restrictions on the use of the language. Furthermore, the fictitious language would have to include unidirectional language constructs, as they are already present in BXtendDSL and were proposed e.g. in [21] as extensions to the relational bx language QVT-R [5].

3. Conclusion

In this paper, we summarized our research in the bx domain. Our approach aims at engineering bidirectional model transformations by offering domain-specific languages, frameworks, and code generators. In this way, we intend to reduce the amount of work that transformation developers have to invest. Our main focus lies on the quality attributes conciseness, expressiveness, and scalability. We also discussed different variants of transformation development processes. So far, we consider the layered approach of BXtendDSL the best choice in terms of concisensess, expressiveness, and scalability. Altogether, our research complements other bx research that primarily focuses on roundtrip properties and provably correct transformations. These issues have been addressed to a limited extent in BXtendDSL, as well, but have not been the major driving force of our research, which rather follows the modest goal of making the life of bx transformation developers easier.

References

[1] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, J. F. Terwilliger, Bidirectional transformations: A cross-discipline perspective, in: R. F. Paige (Ed.), Proceedings of the Second International Conference on Theory and Practice of Model Transformations (ICMT 2009), volume 5563 of *Lecture Notes in Computer Science*, Springer-Verlag, Zurich, Switzerland, 2009, pp. 260–283.

[2] S. Hidaka, M. Tisi, J. Cabot, Z. Hu, Feature-based classification of bidirectional transformation approaches, Software and Systems Modeling 15 (2016) 907–928.

[3] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, A. Schmitt, Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem, ACM Transactions on Programming Languages and Systems 29 (2007) 17:1–17:65.

[4] H. Ko, T. Zan, Z. Hu, BiGUL: a formally verified core language for putback-based bidirectional programming, in: M. Erwig, T. Rompf (Eds.), Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2016, St. Petersburg, FL, USA, January 20 - 22, 2016, ACM, 2016, pp. 61–72. URL: https://doi.org/10.1145/2847538.2847544. doi:10.1145/2847538.2847544.

[5] OMG, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, formal/2015-02-01 ed., Needham, MA, 2015.

[6] A. Cicchetti, D. Di Ruscio, R. Eramo, A. Pierantonio, JTL: A bidirectional and change propagating transformation language, in: B. Malloy, S. Staab, M. van den Brand (Eds.), Proceedings of the Third International Conference on Software Language Engineering (SLE 2010), volume 6563 of *Lecture Notes in Computer Science*, Springer-Verlag, Eindhoven, The Netherlands, 2010, pp. 183–202.

[7] A. Schürr, Specification of Graph Translators with Triple Graph Grammars, in: G. Tinhofer (Ed.), Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1994), volume 903 of *LNCS*, Springer-Verlag, Herrsching, Germany, 1994, pp. 151–163.

[8] A. Anjorin, T. Buchmann, B. Westfechtel, Z. Diskin, H. Ko, R. Eramo, G. Hinkel, L. Samimi-Dehkordi, A. Zündorf, Benchmarking bidirectional transformations: theory, implementation, application, and assessment, Software and Systems Modeling 19 (2020) 647–691. URL: https://doi.org/10.1007/s10270-019-00752-x. doi:10.1007/s10270-019-00752-x.

[9] F. Abou-Saleh, J. Cheney, J. Gibbons, J. McKinna, P. Stevens, Introduction to bidirectional transformations, in: J. Gibbons, P. Stevens (Eds.), Bidirectional Transformations - International Summer School, Oxford, UK, July 25-29, 2016, Tutorial Lectures, volume 9715 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 1–28. URL: https://doi.org/10.1007/978-3-319-79108-1_1. doi:10.1007/978-3-319-79108-1_1.

[10] T. Buchmann, B. Westfechtel, Using triple graph grammars to realize incremental round-

trip engineering, IET Software 10 (2016) 173–181. URL: http://digital-library.theiet.org/content/journals/10.1049/iet-sen.2015.0125.

[11] B. Westfechtel, Case-based exploration of bidirectional transformations in QVT relations, Software and Systems Modeling 17 (2018) 989–1029. doi:10.1007/s10270-016-0527-z.

[12] T. Buchmann, M. Bank, B. Westfechtel, BXtendDSL: A layered framework for bidirectional model transformations combining a declarative and an imperative language, J. Syst. Softw. 189 (2022) 111288. URL: https://doi.org/10.1016/j.jss.2022.111288. doi:10.1016/j.jss.2022.111288.

[13] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen, Model-Driven Software Development: Technology, Engineering, Management, John Wiley & Sons, Chichester, UK, 2006.

[14] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF Eclipse Modeling Framework, The Eclipse Series, 2nd ed., Addison-Wesley, Boston, MA, 2009.

[15] T. Buchmann, Bxtend - A framework for (bidirectional) incremental model transformations, in: S. Hammoudi, L. F. Pires, B. Selic (Eds.), Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2018, Funchal, Madeira - Portugal, January 22-24, 2018., SciTePress, 2018, pp. 336–345. URL: https://doi.org/10.5220/0006563503360345. doi:10.5220/0006563503360345.

[16] L. Bettini, Implementing Domain-Specific Languages with Xtext and Xtend, Packt Publishing, Birmingham, UK, 2016.

[17] M. Bank, T. Buchmann, B. Westfechtel, Combining a declarative language and an imperative language for bidirectional incremental model transformations, in: S. Hammoudi, L. F. Pires, E. Seidewitz, R. Soley (Eds.), Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2021, Online Streaming, February 8-10, 2021, SCITEPRESS, 2021, pp. 15–27. URL: https://doi.org/10.5220/0010188200150027. doi:10.5220/0010188200150027.

[18] P. Stevens, Bidirectional model transformations in QVT: Semantic issues and open questions, Software and Systems Modeling 9 (2010) 7–20.

[19] OMG, Object Constraint Language, formal/2014-02-03 ed., OMG, Needham, MA, 2014.

[20] O. Hacker, BXtendDSL 2: Weiterentwicklung einer hybriden Sprache für bidirektionale Modell-zu-Modell Transformationen, 2022. Master thesis (in German), University of Bayreuth, Germany.

[21] B. Westfechtel, A case study for evaluating bidirectional transformations in QVT Relations, in: J. Filipe, L. Maciaszek (Eds.), Proceedings of the 10th International Conference on the Evaluation of Novel Approaches to Software Engineering (ENASE 2015), SCITEPRESS, Barcelona, Spain, 2015, pp. 141–155.